uCosminexus Application Server

System Design Guide

3021-3-J04-10(E)

Notices

■ Relevant program products

See the *Release Notes*.

■ Export restrictions

If you export this product, please check all restrictions (for example, Japan's Foreign Exchange and Foreign Trade Law, and USA export control laws and regulations), and carry out all required procedures.

If you require more information or clarification, please contact your Hitachi sales representative.

■ Trademarks

HITACHI, Cosminexus, HA Monitor, HiRDB, JP1, OpenTP1, TPBroker, XDM are either trademarks or registered trademarks of Hitachi, Ltd. in Japan and other countries.

AIX is a trademark of International Business Machines Corporation, registered in many jurisdictions worldwide.

AMD is a trademark (or registered trademark) of Advanced Micro Devices, Inc.

Intel is a trademark of Intel Corporation or its subsidiaries.

Linux is the registered trademark of Linus Torvalds in the U.S. and other countries.

Microsoft, SQL Server are trademarks of the Microsoft group of companies.

Microsoft, Windows are trademarks of the Microsoft group of companies.

Microsoft, Windows Server are trademarks of the Microsoft group of companies.

Microsoft is a trademark of the Microsoft group of companies.

Oracle, Java, and MySQL are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Red Hat is a registered trademark of Red Hat, Inc. in the United States and other countries.

Red Hat Enterprise Linux is a registered trademark of Red Hat, Inc. in the United States and other countries.

SPARC is a registered trademark of SPARC International, Inc. Products bearing SPARC trademarks are based on an architecture developed by Sun Microsystems, Inc

UNIX is a trademark of The Open Group.

Other company and product names mentioned in this document may be the trademarks of their respective owners.

This product includes software developed by the Apache Software Foundation (http://www.apache.org/).

■ Issued

Aug. 2022: 3021-3-J04-10(E)

■ Copyright

Preface

For details on the prerequisites before reading this manual, see the *Release Notes*.

■ Non-supported functionality

Some functionality described in this manual is not supported. Non-supported functionality includes:

- Audit log functionality
- Compatibility functionality
- Cosminexus Component Transaction Monitor
- Cosminexus Reliable Messaging
- Cosminexus TPBroker and VisiBroker
- Cosminexus Web Service - Security
- Cosminexus XML Security - Core functionality
- JP1 linkage functionality
- Management Server management portal
- Remote installation functionality for the UNIX edition
- SOAP applications complying with specifications other than JAX-WS 2.1
- uCosminexus OpenTP1 linkage functionality
- Virtualized system functionality
- XML Processor high-speed parse support functionality

■ Non-supported compatibility functionality

"Compatibility functionality" in the above list refers to the following functionality:

- Basic mode
- Check of JSP source compliance (cjjsp2java) with JSP1.1 and JSP1.2 specifications
- Database connection using Cosminexus DABroker Library
- EJB client application log subdirectory exclusive mode
- J2EE application test functionality
- Memory session failover functionality
- Servlet engine mode
- Simple Web server functionality
- Switching multiple existing execution environments
- Using EJB 2.1 and Servlet 2.4 annotation

Contents

Appendixes 382

Index 403

1 Purpose and Flow of Designing an Application Server System

This chapter describes the purpose of designing a Cosminexus Application Server system, the items that must be determined, and the flow of designing an Application Server system.

1.1 Purpose of designing an Application Server system

Cosminexus Application Server is a product used to build an *application server* that is an application execution environment conforming to Java and CORBA industry standards. The application server forms the base of a business system.

The requirements of a business system are as follows:

- **Realize a system with high reliability and availability**
  The system needs to be reliable and available for performing stable operations without any interruptions in the business system.
- **Ensure security**
  When a user (who manages or operates the system) builds and operates system operations and when an end user uses the system-provided services, various security-related threats are anticipated. For protecting the system from threats, the system needs to be designed in a physically safe configuration, and operators need to abide by the operation rules.
- **Achieve high processing performance**
  A prompt and assured response is required for processing the requirements from multiple clients such as Web clients and for processing mission-critical requirements in the business back-end system such as EJB clients.

To build a system that meets the above requirements, you must analyze the purpose and features of the system before building the system, estimate the resources to be used in the system, and determine the optimal system configuration. Furthermore, before starting the actual system operations, you must create procedures for ensuring security, and you actually verify and tune the operations in the expected runtime.

From the above tasks, Application Server system design aims at enabling the business systems running on Application Server to operate in an optimal state. This manual describes the following information that is to be determined and considered when designing a system for Application Server:

- How to determine a system configuration
- How to determine a secure system
- How to implement performance tuning

1.2 Flow of system design

This section describes the flow of designing an Application Server system.

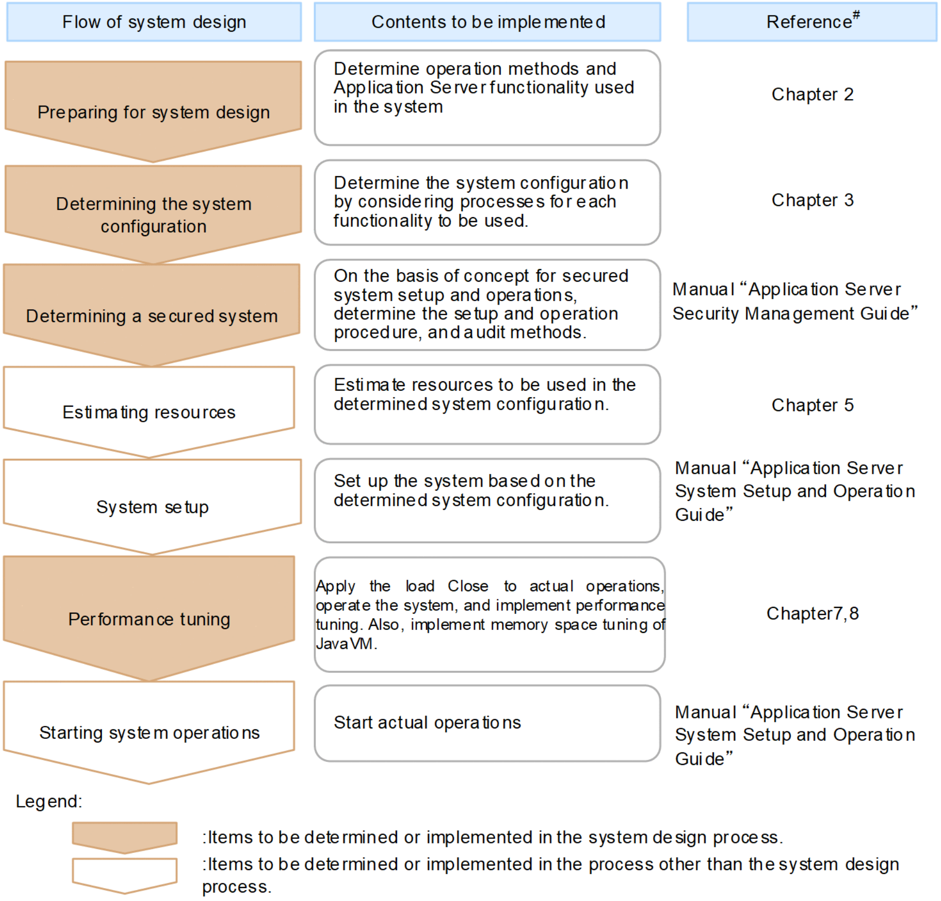
The concept of system design differs depending on whether the applications to be executed on that system are the applications that execute online processing (J2EE applications) or applications that execute batch processing (batch applications).

The flow of each system design is described below.

1.2.1 For applications that execute online processing (J2EE applications)

Design a system as per the flow shown in the following figure. For details about each procedure, see the chapter or manual described in the *Reference* column.

Figure 1–1: Flow of system design (for executing J2EE applications)



Flow of system design	Contents to be implemented	Reference#
Preparing for system design	Determine operation methods and Application Server functionality used in the system	Chapter 2
Determining the system configuration	Determine the system configuration by considering processes for each functionality to be used.	Chapter 3
Determining a secured system	On the basis of concept for secured system setup and operations, determine the setup and operation procedure, and audit methods.	Manual "Application Server Security Management Guide"
Estimating resources	Estimate resources to be used in the determined system configuration.	Chapter 5
System setup	Set up the system based on the determined system configuration.	Manual "Application Server System Setup and Operation Guide"
Performance tuning	Apply the load Close to actual operations, operate the system, and implement performance tuning. Also, implement memory space tuning of JavaVM.	Chapter7,8
Starting system operations	Start actual operations	Manual "Application Server System Setup and Operation Guide"

Legend:

▨ :Items to be determined or implemented in the system design process.

□ :Items to be determined or implemented in the process other than the system design process.

#

For the respective references, see the locations described in the following table.

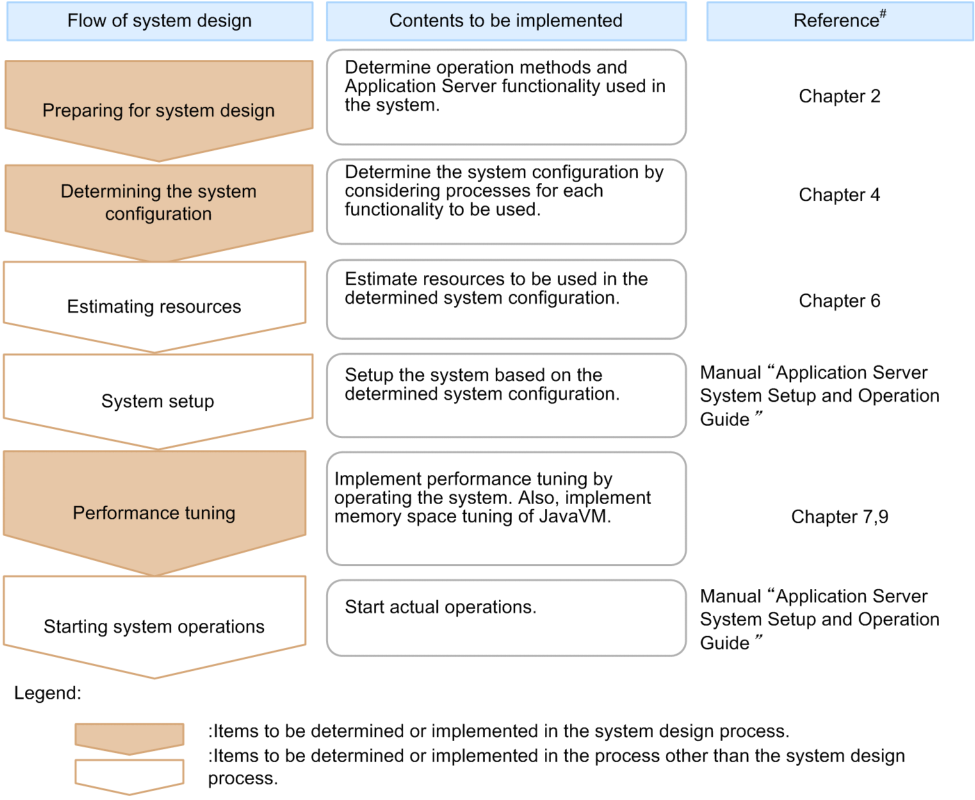
System design	Reference manual	Reference section
Preparing for system design	This manual	*Chapter 2*
Determining the system configuration		*Chapter 3*

System design	Reference manual	Reference section
Determining a secure system	*uCosminexus Application Server Security Management Guide*	*Chapter 4*
Estimating resources	This manual	*Chapter 5*
System setup	*uCosminexus Application Server System Setup and Operation Guide*	
Performance tuning	This manual	*Chapter 7*, *Chapter 8*
Starting system operations	*uCosminexus Application Server System Setup and Operation Guide*	

1.2.2 For applications that execute batch processing (batch applications)

Design a system as per the flow shown in the following figure. For details about each item, see the chapter or manual described in the *Reference* column.

Figure 1–2: Flow of system design (for executing batch applications)



Flow of system design	Contents to be implemented	Reference#
Preparing for system design	Determine operation methods and Application Server functionality used in the system.	Chapter 2
Determining the system configuration	Determine the system configuration by considering processes for each functionality to be used.	Chapter 4
Estimating resources	Estimate resources to be used in the determined system configuration.	Chapter 6
System setup	Setup the system based on the determined system configuration.	Manual “Application Server System Setup and Operation Guide”
Performance tuning	Implement performance tuning by operating the system. Also, implement memory space tuning of JavaVM.	Chapter 7,9
Starting system operations	Start actual operations.	Manual “Application Server System Setup and Operation Guide”

Legend:

▨ :Items to be determined or implemented in the system design process.

□ :Items to be determined or implemented in the process other than the system design process.

#

For the respective reference information, see the *Reference manual* and *Reference section* described in the following table.

System design	Reference manual	Reference section
Preparing for system design	This manual	*Chapter 2*
Determining the system configuration		*Chapter 4*
Estimating resources		*Chapter 6*

System design	Reference manual	Reference section
System setup	*uCosminexus Application Server System Setup and Operation Guide*	*4.6*
Performance tuning	This manual	*Chapter 7*, *Chapter 9*
Starting system operations	*uCosminexus Application Server System Setup and Operation Guide*	*4.6.3*

2 Preparing for System Design

This chapter explains the items that you must determine and prepare before you start designing a system.

2.1 Items to be determined before designing a system

This section describes the items that you must determine before designing an Application Server system.

Identify the following before you start the system design operations. Based on this determination result, you determine a system configuration and implement the performance tuning described in chapter 3 onwards.

- **Identifying the types of business**

  Identify the types of business to be executed on the system. Decide the format of the applications, configuration, and required software depending on the types of business.

- **Determining the application configuration based on the system purpose**

  Based on the purpose of the system, you determine the functionality to be used, identify the configuration of the applications to be executed, and prepare the necessary software.

- **Determining the operation method**

  In an Application Server system, you use an operation management process called the *Management Server* to run multiple server processes in a batch.

  When you determine the operation methods, apart from Application Server systems, you must determine how to operate the entire system including the programs other than Application Server.

2.2 Identifying the types of business

Identify the types of businesses to be executed on the system.

You can execute the following two types of businesses on Application Server systems:

- **Online business**
  This type of business processes requests sent from clients via a network.
- **Batch business**
  This type of business processes routine or periodic work in batch.

The format of an application to be executed or a server process that executes the application differs depending on the types of business. The following table describes the correspondence of the business types, application formats, and server processes that execute applications:

Table 2–1: Correspondence of the business types, application formats, and server processes that execute applications

Business type	Format of application	Server process that executes the application
Online business	J2EE application	J2EE server
Batch business	Batch application	Batch server

Tip

The contents of the system design to be implemented subsequently differ depending on the business type. Implement the required system design according to the business type. The following table describes the contents of the system design to be executed as per the business type and the corresponding reference sections in this manual:

Table 2–2: Contents of the system design to be executed according to the types of business and reference sections in this manual

Contents of system design		Type of business	
		Online business	Batch business
Preparing for system design	Determining the application configuration	*Section 2.5*	*Section 2.6*
	Determining the operation method	*Section 2.7*	
Determining the system configuration		*Chapter 3*	*Chapter 4*
Performance tuning		*Chapter 8*	*Chapter 9*
Tuning of JavaVM		*Chapter 7*, *Section 7.4*, *Section 7.11*	

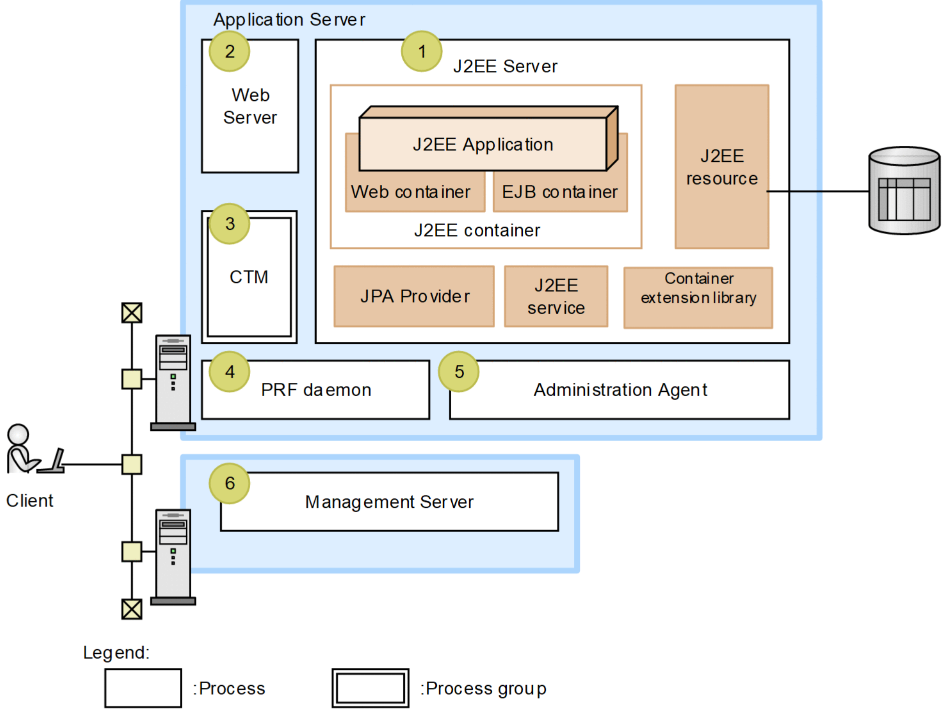
2.3 Determining the functionality to be used (when executing online processing)

This section describes the process configuration of Application Server for executing J2EE applications, and the configuration of a J2EE server.

2.3.1 Process configuration

The following figure shows the processes that configure Application Server for executing J2EE applications:

Figure 2–1: Processes that configure Application Server for executing J2EE applications



Reference note

Note that for building a system, match these processes with the requirements of the system, and then deploy one or multiple processes on each machine within the system.

Each process is described as follows. The numbers in the figure correspond to (1) to (6) in the following description:

(1) J2EE server

A J2EE server is a process that serves as a J2EE application execution platform. The multiple program modules such as J2EE applications, J2EE containers, J2EE services, and J2EE resources configure the J2EE server. Also, the J2EE containers are divided into EJB containers and Web containers according to the provided functionality. The details about the program modules configuring the J2EE server are described in *2.3.2 Configuration of a J2EE server*.

(2) Web server

A Web server is a process that executes the processing related to receiving requests from the Web browser and sending data to the Web browser. You must use the Web server for systems that access the J2EE application running on a J2EE server from the Web browser#. You can also access servlets, JSPs, or static contents included in the J2EE application from the Web browser.

In Application Server, you can use Cosminexus HTTP Server or Microsoft IIS as a Web server. Cosminexus HTTP Server is one of the component software of Application Server. For details about the functionality of Cosminexus HTTP Server, see the *uCosminexus Application Server HTTP Server User Guide*.

#

If you send and receive requests directly between the NIO HTTP server of the J2EE server and the Web browser without going through a Web server, you do not need to implement the Web server process. For details, see the manual *uCosminexus Application Server Web Container Functionality Guide*.

(3) CTM

A CTM is a group of processes used for scheduling the requests for Session Bean within the J2EE application. On using CTM, the requests from the client can be distributed and scheduled appropriately. As a result, you can control the load on the server, increase the system availability, and promote the business without delay.

You use multiple processes such as CTM daemons, CTM regulators, and CTM domain managers to implement the CTM functionality. Also, use the CORBA Naming Service as a Naming Service for implementing functions.

For details on the CTM functionality, see *3. Scheduling and Load Balancing of Requests Using CTM* in the *uCosminexus Application Server Expansion Guide*.

Tip

You can use CTM only with the products that contain Cosminexus Component Transaction Monitor in the component software. For details on the available products, see *2.2 Component software* in the manual *uCosminexus Application Server Overview*.

(4) PRF daemon (Performance tracer)

Application Server outputs trace information to a buffer when processing a request. You can also expand the trace target so that applications in addition to Application Server also output trace information to a buffer. A PRF daemon (performance tracer) is an I/O process that outputs the trace information (output to the buffer) to a file.

The trace information file output by the PRF daemon is useful for verifying bottlenecks of the system and determining troubleshoot efficiency.

For details about the PRF daemon functionality, see *7. Performance Analysis by Using Trace Based Performance Analysis* in the *uCosminexus Application Server Maintenance and Migration Guide*.

(5) Administration Agent

Administration Agent is an agent function that starts the logical server on each host and updates the setup file, instead of an administrator. Note that the logical server is a server or a cluster that is managed by Management Server.

(6) Management Server

Management Server is a process that issues instructions to Administration Agent deployed on each host of the management domain and executes operation management of the entire management domain.

(7) Other processes

In addition to the above (1) to (6) processes, the following processes are used depending on their functionality:

- User server
  A user server is any service or process defined by user. The user server can be defined as the logical server (logical user server), and thereby specific services and processes can be managed by Management Server. As a result, you can manage the services and processes in a batch in the Management Server in the same way as with the other logical servers.

2.3.2 Configuration of a J2EE server

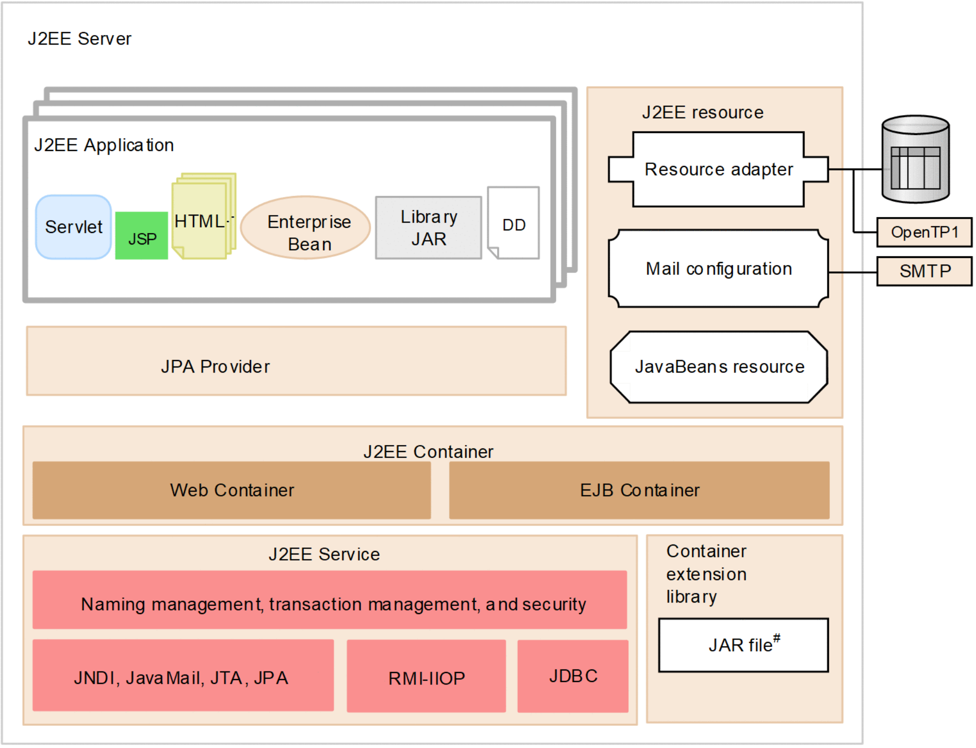
A *J2EE server* is a Java application executing the following six program modules:

- J2EE applications (such as servlets, JSPs, and Enterprise Beans)
- J2EE containers
- J2EE services
  Such as JNDI, JavaMail, JTA, JPA, RMI-IIOP, JDBC, Naming management, Transaction Management, and security
- J2EE resources
- JPA provider
- Container extension library

Servlets, JSPs, and Enterprise Beans configure the J2EE application. The user develops the J2EE application as per the business contents. Note that the program modules other than the J2EE applications are the modules provided by Application Server.

The following figure shows the structure of a J2EE server:

Figure 2–2: Structure of a J2EE server



Legend:

▭ :Scope of program module provided with the Application Server.

User prepares for the functionality that uses JAR file.

The following sections give an overview of each module of the J2EE server:

2.3.3 J2EE applications and J2EE component

One or more J2EE components configure the J2EE application. This subsection describes the J2EE applications and J2EE components.

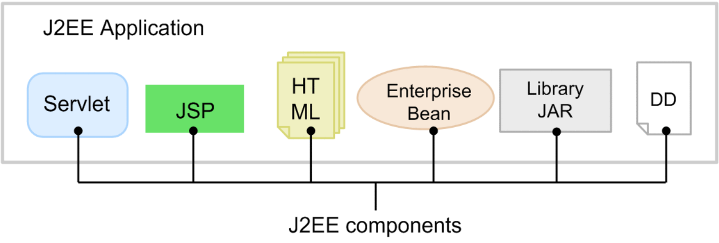
(1) Relation between J2EE application and J2EE components

The user application programs such as servlets, JSPs, and Enterprise Beans configure the J2EE application. The J2EE application runs in the J2EE container.

The servlets, JSPs, and Enterprise Beans configuring the J2EE application are called *J2EE components*.

The following figure shows the relation between the J2EE application and J2EE components:

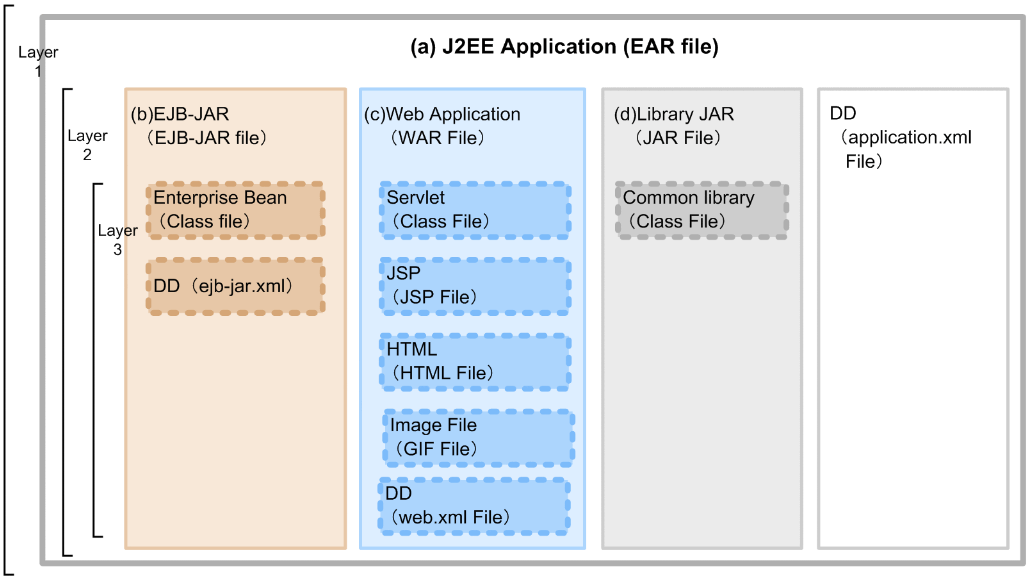
Figure 2–3: Relation between the J2EE application and J2EE components



(2) Structure of a J2EE application

A J2EE application has a structure with three layers. The following figure shows the structure of a J2EE application:

Figure 2–4: Structure of a J2EE application



Note The item number in the figure corresponds to the item number in the text.

The smallest units of the J2EE application are the files in layer 3 (files enclosed with a dotted line in the figure). Layer 3 includes files such as the class files and JSP files.

The files in layer 2 are the packages of the files in layer 3. For example, in the above figure, the EJB-JAR file in layer 2 is formed by packaging an Enterprise Bean and a DD (`ejb-jar.xml`) that belong in layer 3.

Furthermore, the respective files packaged in layer 2 form the J2EE application in layer 1.

The packaged files of layer 1 and layer 2 are as follows. The numbers in the figure correspond to the numbers in the following explanation:

Reference note

The file formats and DTDs of the DDs are predetermined by the respective layers.

A *DD* indicates the file wherein the definition information when deploying applications in the operating environment is coded. For an EJB-JAR file, the DD is `ejb-jar.xml`, for a Web application, the DD is `web.xml`, and for a J2EE application, the DD is `application.xml`.

Note that for using annotations in an Enterprise Bean, `ejb-jar.xml` is not required.

(a) J2EE application

Multiple EJB-JARs, Web applications, library JARs, and a single DD (`application.xml`) configure a J2EE application.

The **J2EE application** that can be executed on the J2EE server is in archive format and in exploded archive format.

- J2EE application in archive format

This is a J2EE application containing application entities such as EJB and servlet in the work directory of J2EE server. To import the J2EE application in archive format in the J2EE server and to make the application executable from client, you must assemble and deploy J2EE applications in EAR or ZIP format.

- J2EE application in exploded archive format

  This is a J2EE application containing application entities such as EJB and servlet in the file or directory based on the fixed rules that are external to the J2EE server. To import the J2EE application of exploded archive format in the J2EE server and to make the application executable from the client, you must deploy J2EE applications.

> **Reference note**
>
> - An *assemble* is an assembly operation used for positioning the EJB-JAR that is not operated independently, as one component of an application. In an assemble, the EJB-JAR is imported as one component to build the J2EE application. Furthermore, apart from the EJB-JAR, you can also include the WAR files, library JARs as a component of the J2EE application.
>
>   For J2EE application of exploded archive format, assembly in EAR format or ZIP format is not required.
> - A *deploy* means making the J2EE application executable from the client.

(b) EJB-JAR

An **EJB-JAR** is packaged in an EJB-JAR file format. Multiple Enterprise Beans and a single DD (`ejb-jar.xml`) configure an EJB-JAR. Note that when using annotations in Enterprise Bean, a DD (`ejb-jar.xml`) is not required.

(c) Web application

A **Web application** is packaged in a WAR file format. Multiple servlets, JSPs, HTML documents, and a single DD (`web.xml`) configure the Web application.

(d) Library JAR

A **library JAR** is packaged in a JAR file format. Multiple common libraries configure the Library JAR. A common library is a library where you can use J2EE components in the J2EE application in common. Apart from the files defined under the `<module>` tag of the DD (`application.xml`) in the J2EE application, the JAR files with the extension in lower case (`.jar`) are considered as the library JAR.

(3) Developing J2EE application and J2EE components

In a J2EE application, to use the functionality as an execution platform provided by Application Server, you must execute the application according to the functionality used. Furthermore, Application Server provides a Developer as a product for developing the J2EE application and J2EE components.

For details about how to develop the J2EE applications and J2EE components, see the *uCosminexus Application Server Application Development Guide*.

2.3.4 J2EE container

A **J2EE container** is a server platform for running the J2EE application. An EJB container and Web container configure the J2EE container.

A J2EE component runs on the J2EE container using the APIs provided by the Web container and EJB container. The Web container and EJB container provided by Application Server are compliant with `Java EE 6`. As a result, you can build a full-fledge key business application complying with Java EE quickly and easily.

(1) Web container

A **Web container** is a server platform used for executing servlets and JSPs. The Web container receives access from the Web client, and provides services according to a request.

The Web container provides servlets and APIs for JSP.

(2) EJB container

An EJB container is a server platform used for controlling the execution of Enterprise Bean, and provides various services for Enterprise Bean. The EJB container provides APIs for EJB.

2.3.5 J2EE service

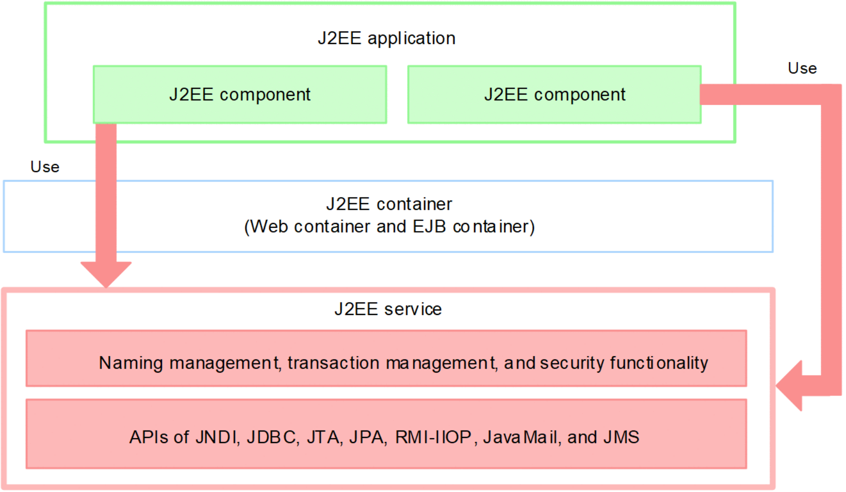
A **J2EE service** provides following functions and APIs:

1. Transaction Management, security management, and naming management functionality
2. APIs of JNDI, JDBC, JTA, JPA, RMI-IIOP, JavaMail, and JMS

The J2EE service is used as one of the component functionality of the J2EE container, and provides functions and APIs to J2EE components such as servlets, JSPs, and Enterprise Beans. The APIs of the J2EE service are used either directly by J2EE components or through the J2EE container.

The following figure shows the position of a J2EE service:

Figure 2–5: Position of a J2EE service


J2EE application
J2EE component
J2EE component
Use
Use
J2EE container
(Web container and EJB container)
J2EE service
Naming management, transaction management, and security functionality
APIs of JNDI, JDBC, JTA, JPA, RMI-IIOP, JavaMail, and JMS

In a J2EE service, apart from the functions of the component software of Application Server, you can also use the functions provided by products other than Application Server. The following table describes the software products or the component software of Application Server used for implementing the J2EE service:

Table 2–3: Products or component software used for implementing the J2EE service

Classification		Product or component software
Service	Naming management	Cosminexus Component Container# Cosminexus TPBroker#
	Transaction Management	

Classification		Product or component software
	Security	Cosminexus Component Container#
API	JNDI	
	JTA	
	JPA	
	JavaMail	
	RMI-IIOP	Cosminexus TPBroker#
	JDBC Standard Extension	HiRDB Type4 JDBC Driver Oracle JDBC Thin Driver SQL Server Driver for JDBC
	JDBC	
	JMS	Cosminexus RM# TP1/Message Queue - Access

#

Component software of Application Server.

2.3.6 J2EE resource

For a J2EE server, you can use resources such as the database, OpenTP1, SMTP server, and the JavaBeans resource. A J2EE resource is used to establish a connection with these resources.

For J2EE resources handled on Application Server, the resource adapters and a mail configuration used for establishing the connection with an external resource, are provided. Apart from these resources, there are JavaBeans resources that you can use as internal resources.

- Resource adapters
  The following are the resource adapters according to the types of resources used for establishing a connection:
  - **DB Connector**
    Used in establishing a connection with the database.
  - **DB Connector for Cosminexus RM and Cosminexus RM**
    Used in establishing a connection with the queue in the database.
  - **uCosminexus TP1 Connector**
    Used in establishing a connection with the SPP of OpenTP1.
  - **TP1/Message Queue - Access**
    Used in establishing a connection with the TP1/Message Queue.
  - **Resource adapter compliant with other Connector 1.0 or Connector 1.5#**
    Used in establishing a connection with any resource.
- Mail configuration
  Used in establishing a connection with the SMTP server.
- JavaBeans resource
  Resource that you can use as an internal resource.

For details about the J2EE resources, see *3. Resource Connections and Transaction Management* in the *uCosminexus Application Server Common Container Functionality Guide*.

#

You can use a resource adapter corresponding to the communication model of `Outbound`. For details, see *3.16.8 Settings to be done when using the connector 1.5 compliant resource adaptor* in the *uCosminexus Application Server Common Container Functionality Guide*.

2.3.7 JPA Provider

A JPA implementation provided by Application Server is called as *JPA provider*. You can use JPA provider for executing JPA applications on Application Server.

For details about JPA providers, see the manual *uCosminexus Application Server Common Container Functionality Guide*.

2.3.8 Container extension library

A common library used in Enterprise Beans, servlets, and JSPs is called as *container extension library*. You can use this library to invoke a user-created common library from the Enterprise Beans, servlets, and JSPs.

For details about the container extension library, see *16. Container Extension Library* in the *uCosminexus Application Server Common Container Functionality Guide*.

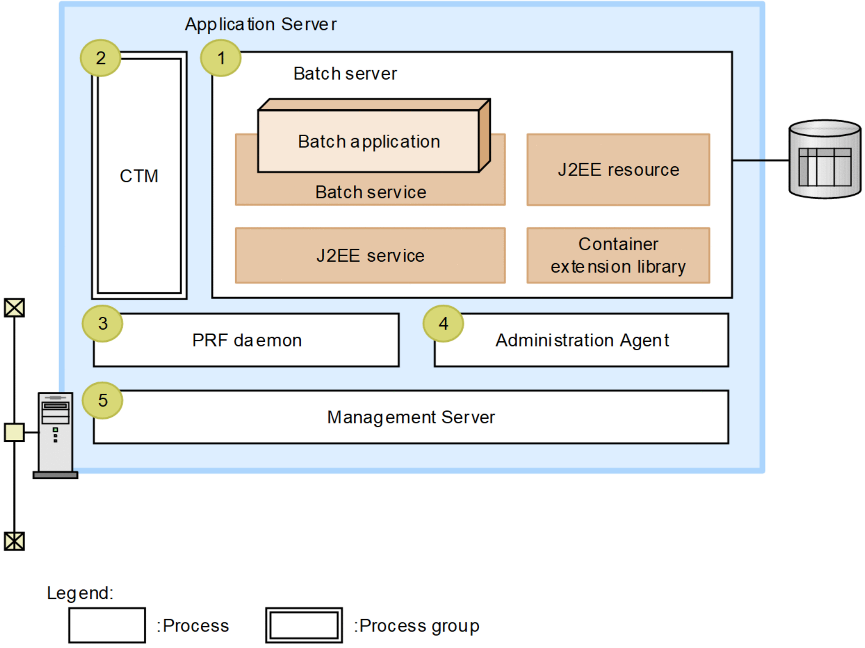
2.4 Determining the functionality to be used (for executing batch processing)

This section describes the process configuration of Application Server for executing batch applications, and the configuration of a batch server.

2.4.1 Process configuration

The following figure shows the process for configuring Application Server for executing batch applications:

Figure 2–6: Process for configuring Application Server for executing batch applications



> **Reference note**
>
> For building a system, match these processes with the requirements of the system, and then deploy one or multiple processes on each machine within the system.

Each process is described as follows. The numbers in the figure correspond to (1) to (5) in the following explanation:

(1) Batch server

A batch server is a process that serves as a batch application execution platform. Multiple program modules such as batch applications, batch services, J2EE services, and J2EE resources configure the batch server. For details about the program modules configuring the batch server, see *2.4.2 Configuration of a batch server*.

(2) CTM

A CTM is a group of processes used for scheduling the execution of batch applications. On using CTM, the execution of a batch application can be distributed and scheduled appropriately. As a result, you can execute multiple batch applications concurrently irrespective of the number of batch servers.

You use multiple processes such as CTM daemons, CTM regulators, and CTM domain managers to implement the functions of CTM. Also, use the CORBA Naming Service as a Naming Service for implementing functions.

For details on the CTM functionality, see *3. Scheduling and Load Balancing of Requests Using CTM* in the *uCosminexus Application Server Expansion Guide*.

Tip

You can use CTM only with the products that contain Cosminexus Component Transaction Monitor in the component software. For details on the available products, see *2.2 Component software* in the manual *uCosminexus Application Server Overview*.

(3) PRF daemon (Performance tracer)

Application Server outputs trace information to a buffer when processing a request. You can also expand the trace target so that applications in addition to Application Server also output trace information to a buffer. A PRF daemon (Cosminexus Performance Tracer) is an I/O process that outputs the trace information (output to the buffer) to a file. The trace information file output by the PRF daemon is useful for verifying bottlenecks of the system and improving the troubleshoot efficiency.

For details about the PRF daemon functionality, see *7. Performance Analysis by Using Trace Based Performance Analysis* in the *uCosminexus Application Server Maintenance and Migration Guide*.

(4) Administration Agent

Administration Agent is an agent function that starts the logical server on each host and updates the setup file, instead of an administrator. Note that the logical server is a server or a cluster that is managed by Management Server.

(5) Management Server

Management Server is a process that issues instructions to Administration Agent deployed on each host of the management domain and executes operation management of the entire management domain.

Reference note

For executing batch processing, other than the above (1) to (5) processes, you can use a process called *user server* according to the purpose of the system. A user server is any service or process defined by user. The user server can be defined as the logical server (logical user server), and thereby specific services and processes can be managed by Management Server. As a result, you can manage the services and processes in a batch in the Management Server alike the other logical servers.

2.4.2 Configuration of a batch server

A *batch server* is a Java application executing the following five program modules:

- Batch applications
- Batch services
- J2EE services

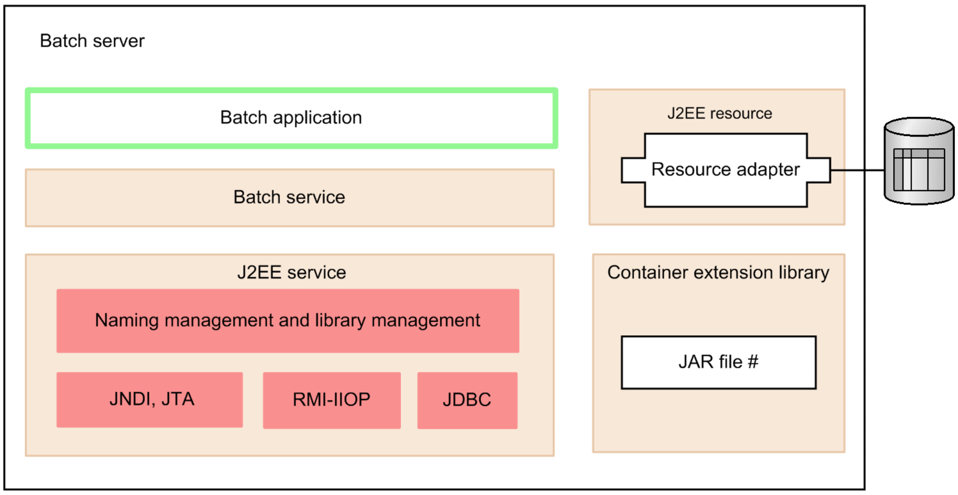
Such as JNDI, JTA, RMI-IIOP, JDBC, Naming management, and Transaction Management

- J2EE resources
- Container extension library

A batch application is a Java application in which batch processing is implemented. The user develops the batch application based on the business contents. Note that the program modules other than the batch applications are the modules provided by Application Server.

The following figure shows the structure of a batch server:

Figure 2–7: Structure of batch server



Legend:

□ :Scope of program module provided with Application Server.

User prepares for the functionality that uses JAR file.

In a batch server, you can use the following Java EE and J2EE functions:

- JDBC 2.0 core / JDBC 2.0 option package
- JDBC 3.0[#1]
- JDBC 4.0[#1, #2]
- Connector 1.0 (DB Connector)[#3]
- JTA 1.0.1 (however, only local)[#4]

#1

A JDBC driver used for establishing a connection must support the functions defined in the specifications for the relevant version.

#2

Only Oracle JDBC Thin Driver can be used to establish a connection.

#3

You can use a DB Connector without transaction or with local transaction.

#4

When `LocalTransaction` is specified in `transaction-support` of the resource adapter DD (`ra.xml`), and a JavaVM is not invoked in the business logic remotely, you can use a local transaction.

Also, the following EJBs can be invoked from the batch server. However, you can invoke EJB remotely and not locally:

- EJB 2.0
- EJB 2.1
- EJB 3.0

The following sections give an overview of each module of the batch server:

2.4.3 Batch application

A batch application is a Java application in which batch processing is implemented. You can execute one batch application for one batch server.

You use the batch execution commands provided with Application Server to start a batch application. In the batch server, after receiving a request for executing a batch application using the batch execution command, start the batch application.

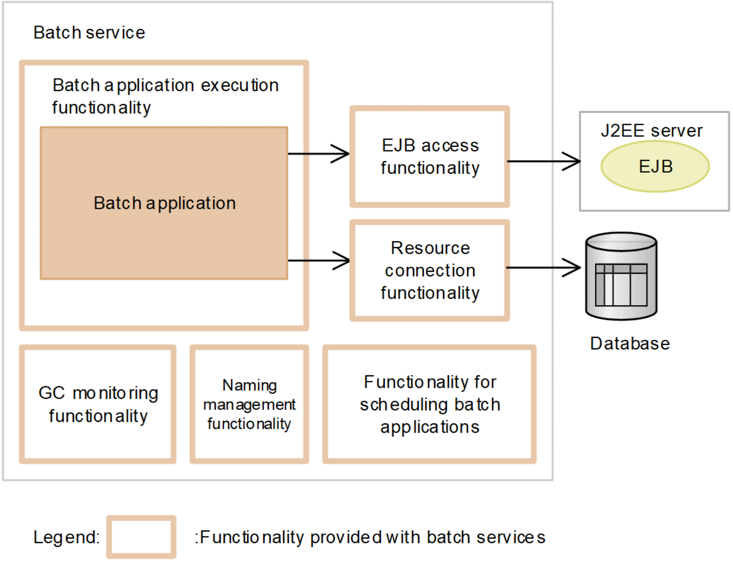
Also, if you use the scheduling function of the batch application, the execution request for batch application is controlled using the schedule queue and automatically distributed to the batch server. When you want to start multiple applications concurrently using the scheduling function of the batch application, you need not be aware of the number of batch servers or the batch server on which the application is to be executed. Furthermore, prepare the batch server for each batch application when you do not use the scheduling function of the batch application.

For details about the batch application, see *2. Executing Applications by Using Batch Servers* in the *uCosminexus Application Server Expansion Guide*.

2.4.4 Batch service

Application Server for executing batch applications provides a batch service. A batch service is a function used for executing batch applications. The following figure shows the functions provided in the batch service:

Figure 2–8: Functions provided in the batch service


Batch service
Batch application execution functionality
Batch application
EJB access functionality
Resource connection functionality
J2EE server
EJB
Database
GC monitoring functionality
Naming management functionality
Functionality for scheduling batch applications
Legend:
:Functionality provided with batch services

- Batch application execution function

This function is provided for starting a batch application and for forced termination of a batch application.

- EJB access function

  This function is provided for accessing EJB of J2EE server from the batch application. An EJB access function uses a J2EE service.

- Resource connection function

  This function is provided for establishing a connection from a batch application to the database. The resource connection function uses the J2EE service and J2EE resources.

- GC control function

  This function is provided for controlling the execution of GC when resources are excluded by a batch application.

- Naming management function

  This function is provided for setting a name when referencing EJB or resources. The naming management function uses a J2EE service.

- Batch application scheduling function

  This function is provided for scheduling the execution of a batch application using CTM.

For details about the respective functionality provided by a batch service, see *2.3 Batch application execution functionality* in the *uCosminexus Application Server Expansion Guide*.

2.4.5 J2EE service

A **J2EE service** provides following functions and APIs:

1. Transaction Management and naming management functionality
2. APIs of JNDI, JDBC, JTA, RMI-IIOP

The J2EE service is used for establishing a connection from a batch application to the database and for invoking an EJB.

In a J2EE service, apart from the functions of the component software of Application Server, you can also use the functions provided by products other than Application Server. The following table describes the software products or the component software of Application Server used for implementing the J2EE service:

Table 2–4: Products or component software used for implementing the J2EE service

Classification		Product or component software
Service	Naming management	Cosminexus Component Container# Cosminexus TPBroker#
	Transaction Management	
API	JNDI	Cosminexus Component Container#
	JTA	
	RMI-IIOP	Cosminexus TPBroker#
	JDBC Standard Extension	HiRDB Type4 JDBC Driver Oracle JDBC Thin Driver SQL Server Driver for JDBC
	JDBC	

#

Component software of Application Server.

2.4.6 J2EE resource

A J2EE resource is used to establish a connection with resources. In a batch server, you can use a database as a resource. For establishing a connection with the database, you use resource adapters among the J2EE resources handled by Application Server.

For details about the J2EE resources, see *3. Resource Connections and Transaction Management* in the *uCosminexus Application Server Common Container Functionality Guide*.

2.4.7 Container extension library

A common library used by an application is called a *container extension library*. You can use this library to invoke a user-created common library from the batch application.

For details about the container extension library, see *16. Container Extension Library* in the *uCosminexus Application Server Common Container Functionality Guide*.

2.5 Determining the application configuration that suits the system purpose (for operations that execute online processing)

This section explains the application configuration to be determined as per the system objectives. The configuration for J2EE applications and the required software are also described in this section.

For details about the application functions that can be implemented on Application Server, see the description about classification of functions in the following manuals:

- *1.1 Functionality Classification* in the *uCosminexus Application Server Web Container Functionality Guide*
- *1.1 Functionality Classification* in the *uCosminexus Application Server EJB Container Functionality Guide*
- *1.1 Functionality Classification* in the *uCosminexus Application Server Common Container Functionality Guide*
- *1.1 Classification of functionality* in the *uCosminexus Application Server Expansion Guide*
- *1.1 Classification of functionality* in the *uCosminexus Application Server Security Management Guide*
- *1.1 Classification of functionality* in the *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*
- *1.1 Classification of functionality* in the *uCosminexus Application Server Maintenance and Migration Guide*
- *1.1 Classification of functionality* in the *uCosminexus Application Server Compatibility Guide*

2.5.1 Determining the J2EE applications to be executed

In this step, you determine the J2EE applications to be executed on a system.

You determine the basic components of a system configuration by identifying the application configuration suitable for the purpose of the system. For example, if you use a Web browser for the client, the application that you would use would be a Web application consisting of servlets and JSPs so that the application receives requests from the Web browser. You can also consider an application that will invoke the Enterprise Beans from servlets and JSPs, as and when required. In this case, the system would be a Web client system and you would need to determine the deployment of the Web server and the integration method for invoking Application Server from the Web server.

Also, if you are building a system that forms the basic component of a business system, you could also consider a configuration that uses EJB client applications for the client. You can also determine the usage of CTM depending on the types of Enterprise Bean of the calling destination.

For more details about the relation between the types of components included in an application and the system configuration, see *3.3 Determining the configuration of an application*. For details about how to determine the system configuration when executing J2EE applications, see *3. Determining the System Configuration (J2EE Application Execution Platform)*.

For details about the functions that you can use according to the purpose of the system, see the description about purpose of the system and their corresponding functions in the following manuals:

- *1.2 Correspondence between the objectives and functions of a system* in the *uCosminexus Application Server Web Container Functionality Guide*
- *1.2 Correspondence between the objectives and functions of a system* in the *uCosminexus Application Server EJB Container Functionality Guide*
- *1.2 Correspondence between the objectives and functions of a system* in the *uCosminexus Application Server Common Container Functionality Guide*

- *1.2 Functionality corresponding to the purpose of the system* in the *uCosminexus Application Server Expansion Guide*
- *1.2 Functionality corresponding to the purpose of the system* in the *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*
- *1.2 Functionality corresponding to the purpose of the system* in the *uCosminexus Application Server Maintenance and Migration Guide*

For details about the application development procedures, see the *uCosminexus Application Server Application Development Guide*.

2.5.2 Determining the processes to be used and preparing the required software

In Application Server systems, the system configuration is determined on the basis of the types of processes to be used and their deployment.

A Web front-end system and a back-end system configure Application Server systems. The Web front-end system uses a Web browser as the client. The back-end system uses an EJB client as the client. For details about the system classification, see *3.1.1 Purpose and configuration of the system*.

The following subsection first explains the processes and software that would be required based on the classification of the system. Next, it describes the processes, modules, and software that would be required depending on the functions to be used. For details about how to deploy the processes, modules, and software on the system, see *3. Determining the System Configuration (J2EE Application Execution Platform)*.

(1) Processes required depending on the system classification

The processes required depending on the system classification are described as follows. These are processes that are required regardless of the functionality used. Application Server provides these processes.

Processes required for a Web front-end system

The processes required for a Web front-end system are as follows:

- Web server#
- J2EE server
- PRF daemon

The Web server included in Application Server is Cosminexus HTTP Server. A Web browser is used for the client.

#

If you access the NIO HTTP server of the J2EE server directly without going through a Web server, you do not need to implement the Web server process.

> **Tip**
>
> **Guidelines for selecting the Web server**
>
> In a Web client system, you can process requests from a Web client using any one of the following Web servers:
>
> - Web server with reverse proxy function enabled

The Web client system processes requests by linking with a Web server that incorporates a proxy module. The requests the Web server receives are sent to the NIO HTTP server of the J2EE server via the proxy module.

- NIO HTTP server

  Processing requests on the HTTP server that functions within the processes of the J2EE server provided as a part of Web container function. You can receive requests from Web clients directly on J2EE servers.

We recommend that you use a Web server with the reverse proxy function enabled on the Application Server. Under the default settings, the Application Server uses a Web server with the reverse proxy function enabled. If you want to build a system with a particular emphasis on performance, consider a configuration that accesses the NIO HTTP server directly.

The following table describes the characteristics of respective Web servers. Use this information as the guideline for selecting Web servers.

Table 2–5: Guidelines for selecting the Web server

Points for Comparison	Web server with reverse proxy function enabled	NIO HTTP server
Functionality available as a Web server	Good You can use various functionality provided by Cosminexus HTTP Server (Web server based on Apache functionality) or Microsoft IIS.	Basic Only minimum functions are provided with an objective of accessing Web applications configured with servlets, JSPs, or HTML.[#1]
Simplified build and operation	Good Web server environment settings are necessary for building a system. You need to start and stop the Web server for operations. However, you can build or run the Web server using the commands of the Smart Composer functionality, and hence, the complex operations are not required.	Good Environment settings of Web servers for the setup, startup, and termination of the Web server for operations are not required.
Access performance for static contents such as HTML and JPEG	Good You can enable optimal performance by deploying the static contents on the Web server. Static contents deployed on a Web container are accessed via reverse proxy, and access processing will take longer.	Good You can ensure optimal performance because access takes place without going through a reverse proxy.
Access performance for dynamic contents such as servlets and JSPs	Basic Access processing takes longer because it goes through a reverse proxy.	Good You can ensure optimal performance because access takes place without going through a reverse proxy.
Notes	If you connect to the internet, from the security point of view, Hitachi recommends that you use a configuration securing DMZ and deploy a reverse proxy on the front-end. If you do not deploy a reverse proxy, you can achieve similar results by deploying a Web server that incorporates a reverse proxy in the DMZ[#2].	While connecting to the Internet, select the configuration that ensures DMZ with the objective of security, and make sure to set up the deployment of reverse proxy at the front-end.[#2]

#1

For details about the functions you can use with the NIO HTTP server, see the manual *uCosminexus Application Server Web Container Functionality Guide*.

#2

For the deployment of a Web server on DMZ, see *3.3 Deployment of reverse proxies in a DMZ* in the *uCosminexus Application Server Security Management Guide*.

Processes required for a back-end system

The processes required for a back-end system are as follows:

- J2EE server
- PRF daemon

The system uses an EJB client for the back-end system client. *EJB client* refers to Servlets, JSPs, other Enterprise Beans, EJB client applications, or other business systems that invoke the Enterprise Beans.

If you are using an EJB client application as an EJB client, you can build the client machine for Windows by using uCosminexus Client. You can invoke the PRF daemon as and when required, whether you use Application Server or uCosminexus Client software.

Reference note

For the system using CTM, you can use any client such as clients of TPBroker or TPBroker Object Transaction Monitor other than EJB client.

(2) Processes and modules required depending on the functions to be used

This subsection describes the processes and modules that are required depending on the functions to be used. Some of these processes and modules are provided by Application Server and some are provided by software other than Application Server.

Among the processes required for each function to be used, the following table describes the processes those are provided by Application Server. You can invoke these processes on the machine where Application Server is installed.

Table 2–6: Processes and modules required for each function (provided by Application Server)

Functions	Necessary processes
Use CTM in the integration between the servers/Distribute the load using CTM	CTM daemon
	CTM regulator
	CTM domain manager
	Global CORBA Naming Service
	Smart Agent
Management using the Management Server	Management Server
	Administration Agent
Invoking the CORBA Naming Service in the out-process	CORBA Naming Service

Among the processes and modules required for each function to be used, Table 2-7 and Table 2-8 describe the processes and modules provided by products other than Application Server.

Table 2–7: Modules required for each function (provided by the products other than Application Server) and software to be provided

Functions	Modules	Provided by	Remarks
Connecting to the database (HiRDB)	HiRDB Type4 JDBC Driver	• HiRDB Server Version 10 • HiRDB/Run Time Version 10 • HiRDB/Developer's Kit Version 10 • HiRDB Developer's Suite Version 10 • HiRDB Server Version 9 • HiRDB Server with Additional Function Version 9 • HiRDB/Run Time Version 9 • HiRDB/Developer's Kit Version 9 • HiRDB Developer's Suite Version 9	This module is required for using the HiRDB Type4 JDBC Driver as a JDBC driver.
Connecting to the database (Oracle)	Oracle JDBC Thin Driver	• Oracle JDBC Thin Driver	This module is required for using the Oracle JDBC Thin Driver as a JDBC driver.
Connecting to the database (SQL Server)	SQL Server JDBC Driver	• SQL Server JDBC Driver	This module is required for using the SQL Server JDBC Driver as a JDBC driver.
Connecting to the database (XDM/RD E2)	HiRDB Type4 JDBC Driver	• HiRDB Server Version 10 • HiRDB/Run Time Version 10 • HiRDB/Developer's Kit Version 10 • HiRDB Developer's Suite Version 10	This module is required for using the HiRDB Type4 JDBC Driver as a JDBC driver.
Connecting to the Message Queue server	TP1/Message Queue - Access	• TP1/Message Queue - Access	--
Connecting to the SPP of OpenTP1	uCosminexus TP1 Connector	• uCosminexus TP1 Connector	--
	TP1/Client/J	• TP1/Client/J	--

Legend:

--: Not applicable.

Note:

The modules are included in the J2EE server processes for the operation.

Table 2–8: Processes required for each function (provided by products other than Application Server) and software to be provided

Function	Necessary processes	Software to be provided
Switch the node using cluster software when an error occurs	Windows Server Failover Cluster	Windows Server Failover Cluster
	HA monitor	HA monitor

2.6 Determining the application configuration that suits the system purpose (for businesses that execute batch processing)

This section describes how to determine an application configuration that suits the system purpose. The configuration for batch application and the required software are also described in this section.

For details about the application functions that can be implemented on Application Server, see the description about classification of functions in the following manuals:

- *1.1 Classification of functions* in the *uCosminexus Application Server Web Container Functionality Guide*
- *1.1 Classification of functions* in the *uCosminexus Application Server EJB Container Functionality Guide*
- *1.1 Classification of functions* in the *uCosminexus Application Server Common Container Functionality Guide*
- *1.1 Classification of functionality* in the *uCosminexus Application Server Expansion Guide*
- *1.1 Classification of functionality* in the *uCosminexus Application Server Security Management Guide*
- *1.1 Classification of functionality* in the *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*
- *1.1 Classification of functionality* in the *uCosminexus Application Server Maintenance and Migration Guide*

2.6.1 Determining batch applications to be executed

Determine batch applications to be run on the system. *Batch applications* are Java applications that are executed as batch processes and in which routine and periodic processes are implemented.

The basic part of the system configuration is determined depending on the process contents implemented in batch applications. For example, when you implement a process that references and updates data in a database, you must determine a method for managing the transactions and method for connecting the resources. When you implement a process that invokes business process programs (Enterprise Beans) on another J2EE server, you must also determine an integration method between servers.

For details about how to determine the system configuration when executing batch applications, see the manual *4. Determining the System Configuration (Batch Application Execution Platform)*.

For details about the functions that you can use according to the purpose of the system, see the description about the purpose of the system and their corresponding functions in the following manuals:

- *1.2 Correspondence between the objectives and functions of a system* in the *uCosminexus Application Server Web Container Functionality Guide*
- *1.2 Correspondence between the objectives and functions of a system* in the *uCosminexus Application Server EJB Container Functionality Guide*
- *1.2 Correspondence between the objectives and functions of a system* in the *uCosminexus Application Server Common Container Functionality Guide*
- *1.2 Functionality corresponding to the purpose of the system* in the *uCosminexus Application Server Expansion Guide*
- *1.2 Functionality corresponding to the purpose of the system* in the *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*
- *1.2 Functionality corresponding to the purpose of the system* in the *uCosminexus Application Server Maintenance and Migration Guide*

2.6.2 Determining the processes to be used and preparing the required software

In Application Server systems, the system configuration is determined according to the types of the processes to be used and their deployment.

In this section, first of all, the necessary processes and software are described. After that the processes, modules, and software required for the functions that you want to use are described. Note that the methods of deploying these processes, modules, and software are described in *4. Determining the System Configuration (Batch Application Execution Platform)*.

(1) Required processes

The required processes are as follows. These processes are required commonly and have no relation to the functions to be used. The processes are provided by Application Server.

- Batch server
- PRF daemon

(2) Processes and modules required for the functions to be used

The processes and modules required for the functions to be used are described here. Some of these processes and modules are provided by Application Server and some are provided by software other than Application Server.

Among the processes required for each function to be used, the following table describes the processes that are provided by Application Server. You can invoke these processes on the machine where Application Server is installed.

Table 2–9: Processes and modules required for each function (provided by Application Server)

Functions	Necessary processes
Management using the Management Server	Management Server
	Administration Agent
Scheduling the execution of the batch application using CTM	CTM daemon
	CTM regulator
	CTM domain manager
	Global CORBA Naming Service
	Smart Agent

Among the processes and modules required for each function to be used, Table 2-10 and Table 2-11 describe the processes and modules provided by products other than Application Server.

Table 2–10: Modules required for each function (provided by products other than Application Server) and software to be provided

Functions	Modules	Provided by	Remarks
Connecting to the database (HiRDB)	HiRDB Type4 JDBC Driver	• HiRDB Server Version 10 • HiRDB/Run Time Version 10 • HiRDB/Developer's Kit Version 10 • HiRDB Developer's Suite Version 10 • HiRDB Server Version 9	This module is required for using the HiRDB Type4 JDBC Driver as a JDBC driver.

Functions	Modules	Provided by	Remarks
		• HiRDB Server with Additional Function Version 9 • HiRDB/Run Time Version 9 • HiRDB/Developer's Kit Version 9 • HiRDB Developer's Suite Version 9	
Connecting to the database (Oracle)	Oracle JDBC Thin Driver	• Oracle JDBC Thin Driver	This module is required for using Oracle JDBC Thin Driver as a JDBC driver.
Connecting to the database (SQL Server)	SQL Server JDBC Driver	• SQL Server JDBC Driver	This module is required for using the SQL Server JDBC Driver as a JDBC driver.
Connecting to the database (XDM/RD E2)	HiRDB Type4 JDBC Driver	• HiRDB Server Version 10 • HiRDB/Run Time Version 10 • HiRDB/Developer's Kit Version 10 • HiRDB Developer's Suite Version 10	This module is required for using the HiRDB Type4 JDBC Driver as a JDBC driver.

Note:

The module is included and executed in the processing of batch servers.

Table 2–11: Processes required for each function (provided by products other than Application Server) and software to be provided

Function	Necessary processes	Software to be provided
Switch the node using cluster software when failure occurs	Windows Server Failover Cluster	Windows Server Failover Cluster
	HA monitor	HA monitor

2.7 Determining the operation method

This section describes the items to be considered when you determine operation methods of Application Server systems.

In the operations of Application Server systems, you can use a process called *Management Server*. If you are using Management Server, you can handle multiple processes configuring Application Server systems as a *logical server*, and execute the processes in a batch.

A business system is generally built by combining Application Server as well as other systems that are built using other programs. When you determine an operation method, apart from Application Server systems, you must determine how to operate the entire system including these programs.

2.7.1 Operations of a system integrated with JP1

When you operate a system by using Management Server, you can integrate Management Server with JP1 programs (Hitachi middleware programs for integrated system management) and realize the following system operations:

- Centralized monitoring of an entire system that is integrated with JP1/IM
- Automatic operations of an entire system that is integrated with JP1/AJS

For details about the functionality that can be implemented by integrating with each JP1 program, see the *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*.

Note that even if you operate a system without using Management Server, you can still automatically operate a system that is using JP1/AJS. However, in this case, you must define items such as the server management commands from scratch as JP1/AJS jobs.

2.7.2 Operations of system integrated with cluster software

If you integrate a system with cluster software, automatic *node switching* can be performed when a failure occurs.

In Application Server systems, you can use the following cluster software for each OS:

- **In Windows**
  Windows Server Failover Cluster
- **In AIX or Linux**
  HA monitor

For integrating with cluster software, you must operate systems using the Management Server.

For details about configurations of the operating system integrated with the cluster software, see *3.11 Determining node switching when cluster software is used and an error occurs*.

3 Determining the System Configuration (J2EE Application Execution Platform)

This chapter describes the system configuration when building a J2EE application execution platform. Along with the flow of system design, this chapter describes the standard pattern of the system configuration of respective design items. This chapter also describes the components, processes, and the processing flow that need to be identified at the respective points.

For determining the system configuration of a batch application execution platform, see *4. Determining the System Configuration (Batch Application Execution Platform)*.

3.1 Points to be considered when determining the system configuration

This section describes the points that must be considered when determining the configuration of a system that uses Application Server.

When determining the system configuration, depending on the functionality that would be used in the J2EE application, you need to identify the processes required for achieving that functionality, and appropriately deploy the processes on each machine. At this point, you also need to give sufficient consideration to the requirements of an Application Server system including reliability and availability.

3.1.1 Purpose and configuration of the system

The systems built using Application Server can be classified into the following two types, depending on the requirements and features of the intended business and the J2EE application that will run on the system:

- Web front-end system
- Back-end system

A *Web front-end system* is a system (in the case of web-based system) that receives requests sent from a Web browser that acts as the front-end and processes such requests. In this system, servlets, JSPs, and Enterprise Beans run in a J2EE server that runs on the Application Server system.

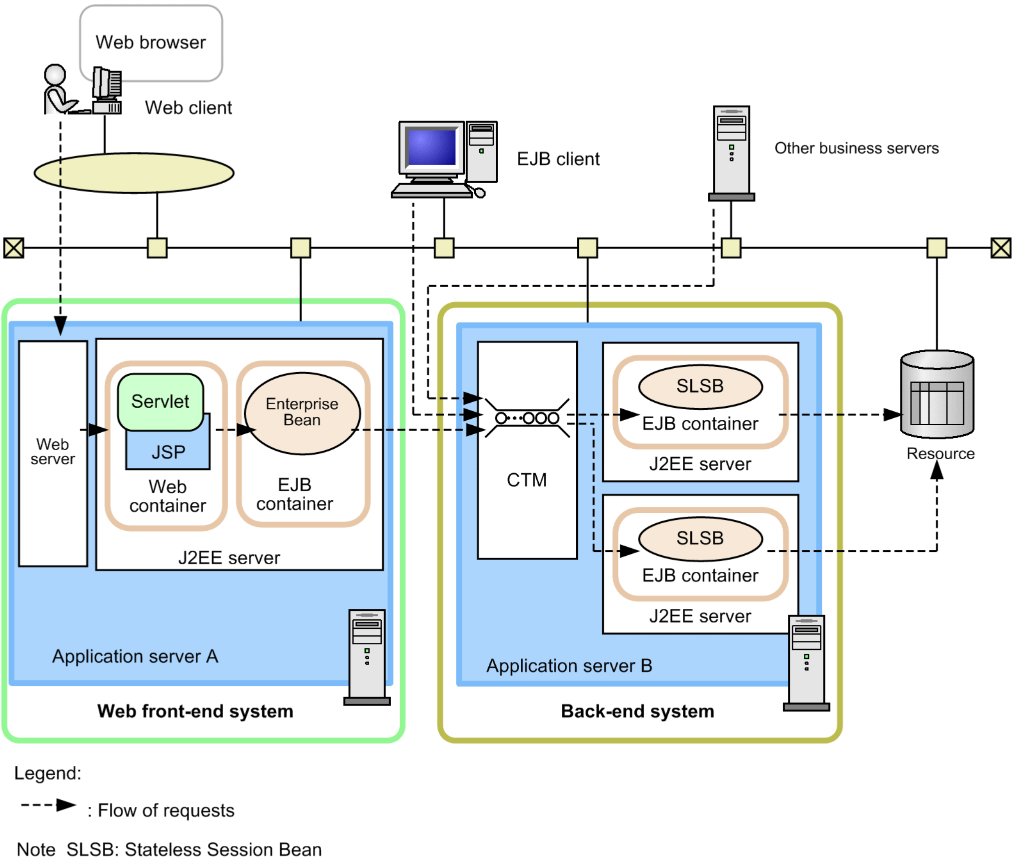
A *Back-end system* runs behind the Web front-end system and executes business services that are common to multiple business systems. The following components, applications, or systems send requests to the back-end system:

- A servlet, JSP, or Enterprise Bean running on the Web front-end system
- EJB client applications running on the EJB client machine
- Other business systems

The Application Server system consists of one or more Web front-end systems and back-end systems, depending on the purpose and scale.

The following figure shows an example of the configuration of a Web front-end system and back-end system:

Figure 3–1: Example configuration of a Web front-end system and a back-end system

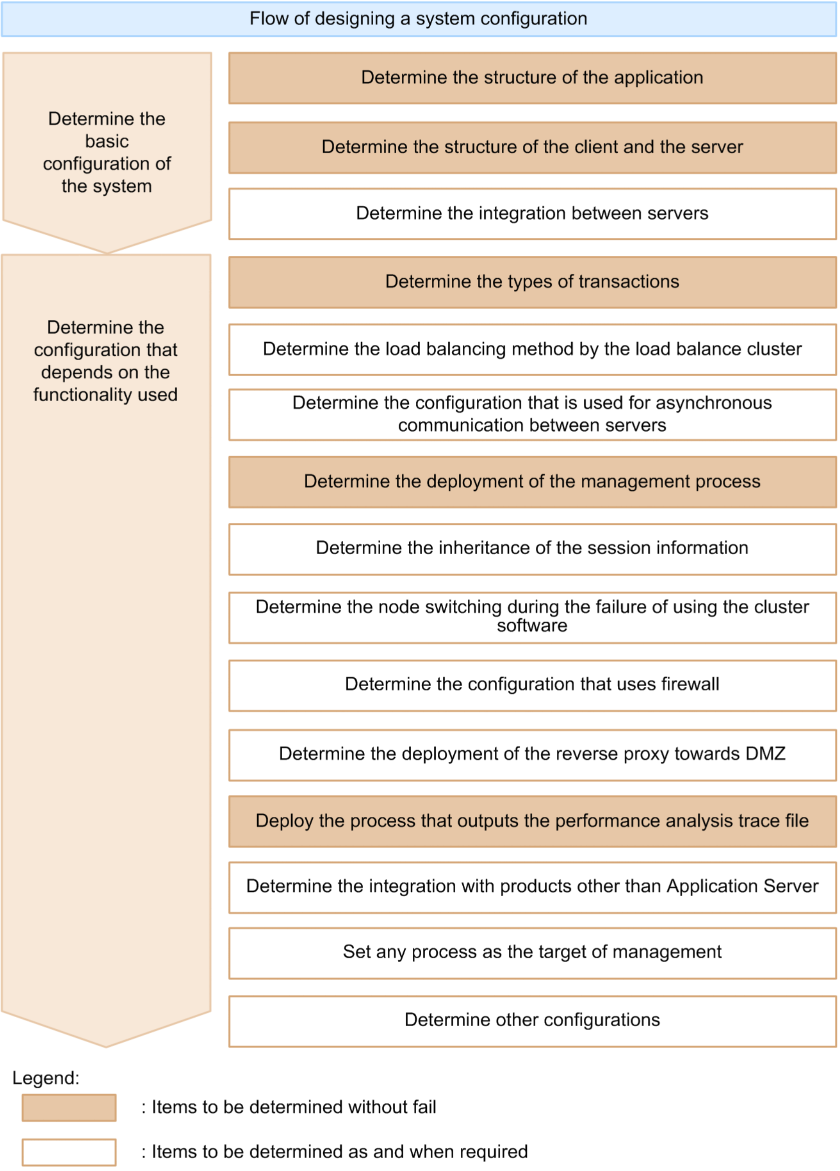


When determining the system configuration of the Application Server system, first, determine the basic configuration. This includes determining the combination of systems required to form the entire system. Next, identify the purpose of each constituent system and the points that will be accessed from the client. Further, determine how the software and the processes should be deployed in order to fulfill the common requirements of reliability, performance and extensibility that are expected of any Application Server, as well as the system specific requirements such as connectivity with EIS and load balancing, and thus, design the optimum system configuration.

3.1.2 Procedure for designing the system configuration

Design the system configuration in the order given below:

Figure 3–2: Flow for designing the system configuration (for a J2EE application execution platform)


Flow of designing a system configuration
Determine the basic configuration of the system
Determine the configuration that depends on the functionality used
Determine the structure of the application
Determine the structure of the client and the server
Determine the integration between servers
Determine the types of transactions
Determine the load balancing method by the load balance cluster
Determine the configuration that is used for asynchronous communication between servers
Determine the deployment of the management process
Determine the inheritance of the session information
Determine the node switching during the failure of using the cluster software
Determine the configuration that uses firewall
Determine the deployment of the reverse proxy towards DMZ
Deploy the process that outputs the performance analysis trace file
Determine the integration with products other than Application Server
Set any process as the target of management
Determine other configurations
Legend:
: Items to be determined without fail
: Items to be determined as and when required

Note: The order to determine the configuration that depends on the functionality used is optional.

(1) Determining the application configuration

Specify the configuration of components used in an application running on each system and decide which components of the application will serve as the access points. For details, see *3.3 Determining the configuration of an application*.

Access points are the components that operate as the request-receiving window of the application, in the case of remote access from the client. The system configuration that can be implemented is determined, based on the type of the access point components.

Determine whether an application is to be connected to a database or other resources. The resource adapter to be used is determined, based on the resource to be connected.

The items that need to be determined in designing the rest of the system configuration differ according to the type of access point components determined here. The following table describes the items that need to be determined for various access point components:

Table 3–1: Items to be determined for various access point components

Design items	Types of access point components				Reference
	Web front-end system	Back-end system			
	Servlet or JSP	Session Bean / Entity Bean	Stateless Session Beam when using CTM	Message-driven Bean	
Configuration of the client and server	Y	Y	Y	--	*Section 3.4*
Method of integration between servers	--	R	R	--	*Section 3.5*
Transaction type	Y	Y	Y	Y	*Section 3.6*
Load balancing method as per the load-balancing cluster	R	R	R	--	*Section 3.7*
Asynchronous communication between the servers	--	--	--	Y	*Section 3.8*
Deploying the operation management process	Y	Y	Y	Y	*Section 3.9*
Configuration for inheriting the session information	R	R	--	--	*Section 3.10*
Configuration for node switching using cluster software	R	R	R	R	*Section 3.11*
Deploying a firewall	R	R	R	R	#2
Deploying a reverse proxy on DMZ	R#1	--	--	--	#3
Deploying a process that outputs the performance analysis trace file	Y	Y	Y	Y	*Section 3.12*
Integrating with products other than Application Server	R	R	R	R	*Section 3.13*
Operation management of the optional processes	R	R	R	R	*Section 3.14*

Legend:

Y: Item that needs to be determined.

R: Item that needs to be determined, if required.

--: Item that need not be determined.

#1

You must determine this item when using the NIO HTTP server directly without going through a Web server.

#2

See *3.2 System configurations using a firewall* in the *uCosminexus Application Server Security Management Guide*.

#3

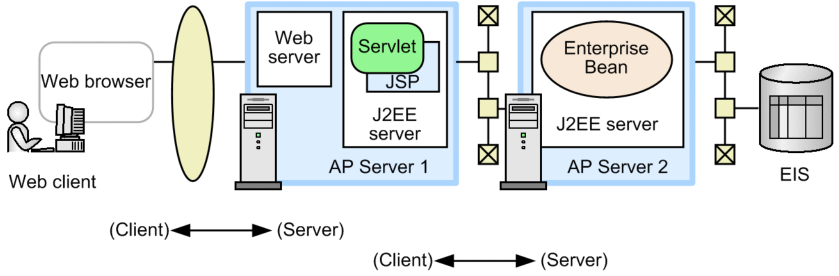
See *3.3 Deployment of reverse proxies in a DMZ* in the *uCosminexus Application Server Security Management Guide*.

(2) Determining the client and server configuration

Specify the correspondence between the client and the server, depending on the access point components. The Application Server instances that are deployed in the system function as a client and as a server, depending on role of each instance.

The following figure shows the concept of the client and the server configuration:

Figure 3–3: Concept of the client and the server configuration



Note: AP Server: Application Server

In the case of this configuration, the AP server 1 is the server for the web client and the access point is the servlet or JSP. AP server 1 is also the client for AP server 2. AP server 2 is the server for AP server 1 and the access point is the Enterprise Bean.

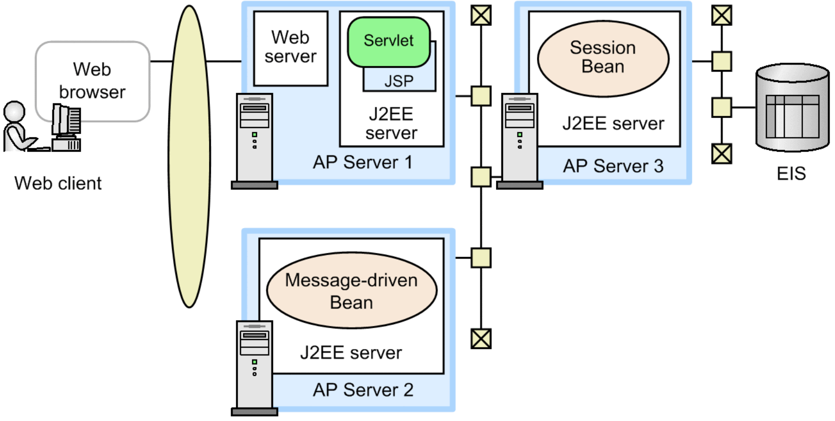
For details, see *3.4 Determining the configuration of the client and the server*.

(3) Determining the method of server integration

In the case of multiple servers, determine whether to perform server integration and if integration is performed, the way in which the integration would take place. Server integration refers to invoking and processing the access point components of other servers from the multiple servers that are arranged vertically as seen from the client.

The following figure shows the concept of server integration:

Figure 3–4: Concept of server integration



Note AP server: Application Server

In the case of this configuration, the AP server 1 and the AP server 2 become the clients for the AP server 3 and invoke a Session Bean of AP server 3 from their respective servlets, JSPs, or Message-driven Beans.

If required, along with server integration, revise the form of the application such as division of application. Even when performing integration between multiple systems, you need to determine the method of integration between servers present in the respective systems.

For details, see *3.5 Determining integration between servers*.

(4) Determining the transaction type

In the case of a system that uses a database or other resources, determine the transaction type to be used, depending on the number of resources for which the transaction management has to be performed. For details, see *3.6 Determining the transaction type*.

(5) Determining the load balancing method based on the load-balancing cluster

To increase the availability of the system, determine whether the load is to be balanced by the load-balancing cluster configuration. When balancing the load, determine the implementation method depending on the type of the access point components. For details, see *3.7 Determining the load balancing method by the load-balancing cluster*.

(6) Determining the configuration for asynchronous communication between servers

When asynchronous communication takes place between Application Server instances by using a Message-driven Bean, determine the system configuration corresponding to the product that is used. At the same time, determine the load balancing by the Message-driven Bean. For details, see *3.8 Determining the configuration for asynchronous communication between servers*.

(7) Determining the deployment of the operation management process

If you are using the operation management process (Management Server), determine on which server machine the Management Server is to be deployed, depending on the scope of management. For details, see *3.9 Determining the deployment of the operation management process*.

(8) Determining the inheritance of the session information

If an error occurs in the J2EE application or the J2EE server running on the Web front-end system, determine the configuration for inheriting the session information in another J2EE server. For details, see *3.10 Determining the inheritance of session information*.

(9) Determining the configuration for node switching when cluster software is used and an error occurs

Determine a configuration in which cluster software executes node switching to continue system operations when an error occurs in an Application Server or when you want to perform system maintenance. You can determine a configuration (mutual standby configuration) in which the executing node and the standby node are mutually switched over. For details, see *3.11 Determining node switching when cluster software is used and an error occurs*.

(10) Determining the deployment of a firewall

Determine the deployment of a firewall for ensuring the security of the Application Server instances and the resources. Deploy a firewall in front of the access point components to prevent invalid access to these access points. You decide the points for deploying a firewall based on the access points.

For details on allocating a basic firewall, see *3.2 System configurations using a firewall* in the *uCosminexus Application Server Security Management Guide*.

For details about the security configuration containing an intrusion detection system, see *4.11.2 Deploying a firewall and intrusion detection system* in the *uCosminexus Application Server Security Management Guide*.

(11) Determining the deployment of a reverse proxy on DMZ

In the case of a system that is connected to the Internet, determine the deployment of a reverse proxy on DMZ to ensure the security of the Application Server instances and the resources. For details, see *3.3 Deployment of reverse proxies in a DMZ* in the *uCosminexus Application Server Security Management Guide*.

(12) Deploying a process for the output of the performance analysis trace file

Determine the deployment of a PRF daemon (performance tracer) that is a process used for performance analysis. For details, see *3.12 Deploying a process for the output of the performance analysis trace file*.

(13) Determining the integration with products other than Application Server

Determine the integration with products other than Application Server, such as JP1 for centralized monitoring, configuration management, or automatic execution of the system, as and when required. For details, see *3.13 Determining integration with products other than Application Server*.

(14) Managing optional processes with operation management

Determine the deployment of a user server when you want to manage user-defined optional processes by using the Management Server as a user server. For details, see *3.14 Managing optional processes with operation management*.

(15) Determining other configurations

Consider the configurations other than those determined up to (14). Also consider system configurations that are compatible with systems created using older versions of Application Server as needed.

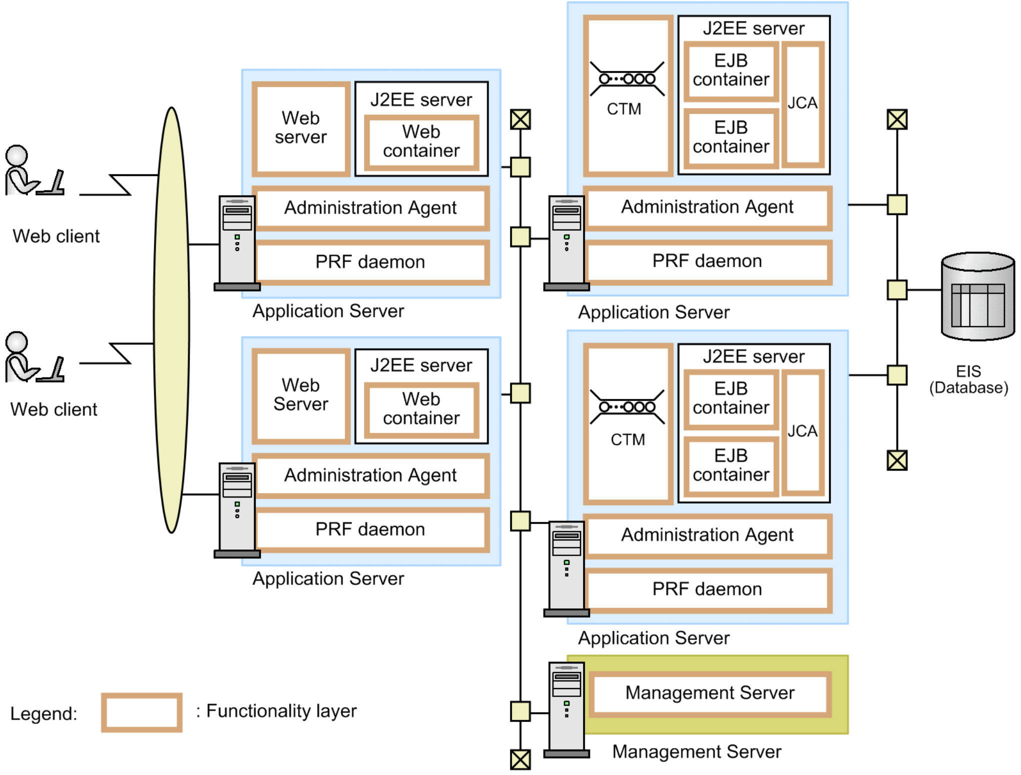
3.1.3 Concept of system configuration

This subsection describes the basic concept when you determine a system configuration. An execution environment that is built in the Application Server is configured of multiple processes. Each process has multiple layers for each provided functionality. These layers are called the *functionality layers*.

In the execution environment, you can deploy each functionality layer on different servers. You need to determine the deployment, depending on the scale and purpose of the system.

The following figure shows an example of deploying the functionality layers:

Figure 3–5: Example of deploying the functionality layers



The concept of a configuration with the following functionality layers deployed in the execution environment is explained below:

- Web server
- Web container
- CTM
- EJB container
- JCA
- EIS
- Naming Service
- Management Server

(1) Deploying a Web server and a Web container (J2EE server)

In addition to sending the HTTP requests to the Web container, a Web server processes the static contents included in the business process. The web container operates as a part of the J2EE server and functions as a base for executing the servlet and JSP. The Web server and the Web container can be deployed either on the same host or on different hosts. Note that a Web server is not required when accessing the NIO HTTP server of the J2EE server directly without going through a Web server.

Hitachi recommends that you deploy one Web server for each J2EE server on which the Web container runs. In the case of load balancing, Hitachi recommends that you deploy the load balancer in front of the Web server.

After doing this, determine the system configuration by referring to the following explanation:

- *3.4.1 Configuration with servlets and JSPs as access points (for Web server integration)*

- *3.4.2 Configuration where servlets and JSPs are used as access points (when accessing the NIO HTTP server directly)*

(2) Deployment of CTM and EJB containers (J2EE server)

The CTM is the functionality that uses the OLTP technology for scheduling the requests from a client. The EJB container is a functionality that operates as a part of J2EE server, and provides the system level services such as execution and communication of Enterprise Beans and transaction management.

The CTM executes the scheduling process of IIOP requests sent to the EJB container and load balancing process. The CTM is a special functionality layer for processing IIOP requests, and therefore, the CTM is not required to be deployed on the EJB container and other hosts. The data transfer between the CTM executes the load balancing of IIOP requests. However, only the load balancing of Stateless Session Beans is executed by the CTM.

Hitachi recommends that you configure the respective load-balancing cluster configuration by setting up the ratio of CTM and EJB containers as one to many. This setup increases the throughput. You can build a system having higher expandability and performance. With cluster configurations, you can perform the partial degeneration and partial restoration, if an error occurs. The reliability and also the availability increase with cluster configurations.

Based on this, reference the following subsections and decide the system configuration:

- *3.4.4 Configuration where Stateless Session Bean is used as access point when using CTM*
- *3.7.4 Load balancing when using CTM (for Stateless Session Bean)*

(3) Deploying JCA and EIS

JCA provides a supporting function for connecting to the EIS of the existing system and the database.

The connection pooling process for EIS is executed in JCA. When EIS is built on one server, the relationship between JCA and EIS is many-to-one. If EIS is built on multiple servers, the relationship between JCA and EIS becomes many-to-many. When failure occurs in the EIS connection, the reliability and availability of the system is ensured, because JCA supports the function for automatic recovery.

(4) Deploying the Naming Service

The Naming Service provides the naming management function to enable the use of an object by its name. The Cosminexus TPBroker provides the Naming Service. Hitachi recommends that you invoke the Naming Service by the in-process of the J2EE server.

You can use invoking by the in-process to reduce the number of processes in the Application Server. Furthermore, the individual start and stop processes are no longer required and operability will improve.

(5) Deploying the operation management process

Management Server contains the functionality for operation management of the entire Application Server system. Deploy one Management Server instance on the domain where the operation management and monitoring of the system are to be performed.

Although, this manual describes the configuration where you deploy Management Server on a host other than the one on which the other J2EE server processes are deployed, you may also deploy Management Server on the same host.

Invoke Administration Agent on the respective hosts where the processes, for which operation management and monitoring are to be performed, are deployed. Administration Agent executes the operation in each host on receiving the instructions from Management Server.

The operation administrator uses the Smart Composer functionality or Management Server commands for management. These commands are executed on the same host where Management Server is deployed.

For details about the system configuration where the operation management process is deployed, see *3.9 Determining the deployment of the operation management process*.

3.2 Description of the system configuration

Chapter 3 describes various system configuration patterns used for determining the system configuration as well as the features of the respective patterns. This section describes the common points to be considered for each system configuration pattern that you need to check before reading the explanation about the system configurations. The legend items that are used in the system configuration figures of this chapter are also explained. Read the following sections after checking these legend items.

> **Reference note**
>
> The legend items described here are also applicable to, *4. Determining the System Configuration (Batch Application Execution Platform)*, and *4.11.2 Deploying a firewall and intrusion detection system* in the *uCosminexus Application Server Security Management Guide*.

3.2.1 Common points to be considered for the system configurations described in this chapter

Note the following common points for each system configuration:

- This chapter describes the system configuration using Application Server. You can read the terminology used in this chapter for the products that you use, as and when required.

  The products that you use and products used in this chapter are described in the following table:

  Table 3–2: Products to be used and products used in this chapter

Products to be used	Products described in this chapter
uCosminexus Developer#	Application Server
uCosminexus Service Architect#	
uCosminexus Service Platform	

  #

  You must read this as is when you use this product in a test environment.

- Most of the examples of system configurations show the deployment of one Application Server for each role. However, in the case of actual configuration, Hitachi recommends that you implement a cluster configuration by using the load balancer for scalability and availability. This will help to avoid complete suspension of the service whenever problems occur and server maintenance is required. For details about the configuration where load balancing is implemented, see *3.7 Determining the load balancing method by the load-balancing cluster*.

- This chapter describes examples where Management Server is used for performing operations. When you perform an operation without using Management Server, the following processes are unnecessary so you need not read the description:

  - Management Server
  - Administration Agent

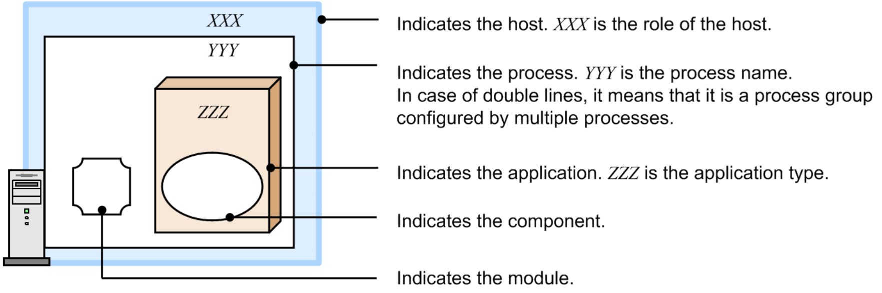
  The machine that starts only those processes that are required for operation management is called the *Management Server machine*. Note that the management sever machine is only required when you use Management Server for performing operations.

- This chapter mainly cites examples of configurations that involve Web server integration when the access point is either a servlet or JSP. Because the following process is unnecessary when accessing the NIO HTTP server of the J2EE server directly without going through a Web server, you can ignore mentions of this process:
  - Web server
- The PRF daemon is a process that outputs a performance analysis trace file. Always deploy the PRF daemon on the Application Server and if required, also deploy the PRF daemon on the EJB client. For details about the deployment of a PRF daemon, see *3.12 Deploying a process for the output of the performance analysis trace file*.

3.2.2 Legend items used for system configuration figures

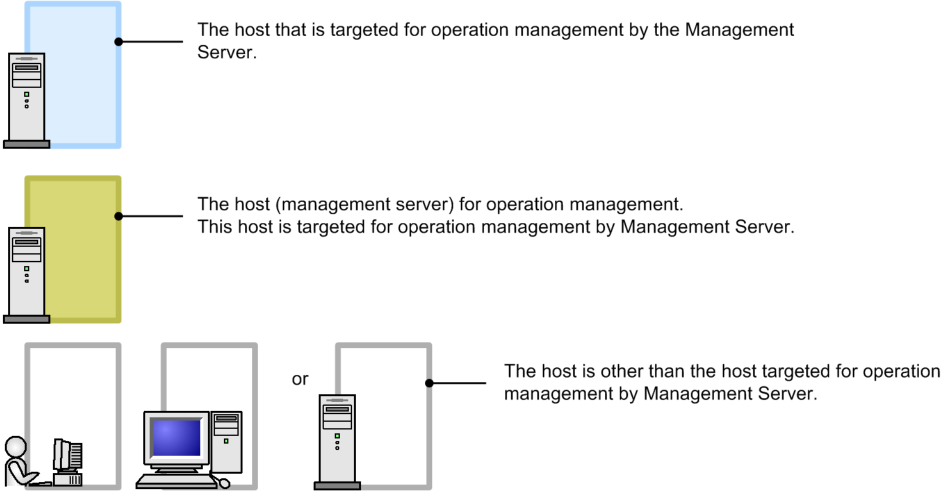
This subsection shows the legend items used in the system configuration figures of chapter 3 and chapter 4.

Figure 3–6: Legend items used for system configuration figures


XXX
YYY
ZZZ
Indicates the host. XXX is the role of the host.
Indicates the process. YYY is the process name.
In case of double lines, it means that it is a process group configured by multiple processes.
Indicates the application. ZZZ is the application type.
Indicates the component.
Indicates the module.

The color that indicates the host also indicates the host type for operation management by Management Server.

Figure 3–7: Legend items used for figures in system configuration (classification of the host)


The host that is targeted for operation management by the Management Server.
The host (management server) for operation management.
This host is targeted for operation management by Management Server.
or
The host is other than the host targeted for operation management by Management Server.

The legend items that are not explained here are given in the respective figures.

3.3 Determining the configuration of an application

This section describes how to determine the configuration of applications.

You can classify the applications based on the type of components to be configured. Each classification has its access points as seen from the client side. This section describes the type of components constituting the application, the access point components in each application, and the resource adapter type corresponding to the type of connected resources.

3.3.1 Configuration of an application and the access points

This subsection describes the type of components that constitute an application and the access points of the respective configurations.

An application consists of the following three types of components:

- **Servlet and JSP**
- **Session Bean and Entity Bean**
- **Message-driven Bean**

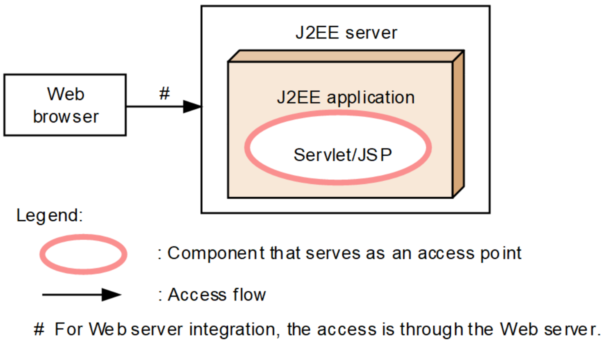
(1) Application consisting of servlets and JSPs

The servlets and JSPs are components for dynamic generation of the presentation displayed on the Web browser of a client machine. They are accessed from the Web browser (that acts as a client) through the Web server using HTTP or HTTPS protocols.

In an application consisting of servlets and JSPs, servlets or JSPs deployed in the front form the access point components as seen from the client side.

The following figure shows an application consisting of servlets and JSPs:

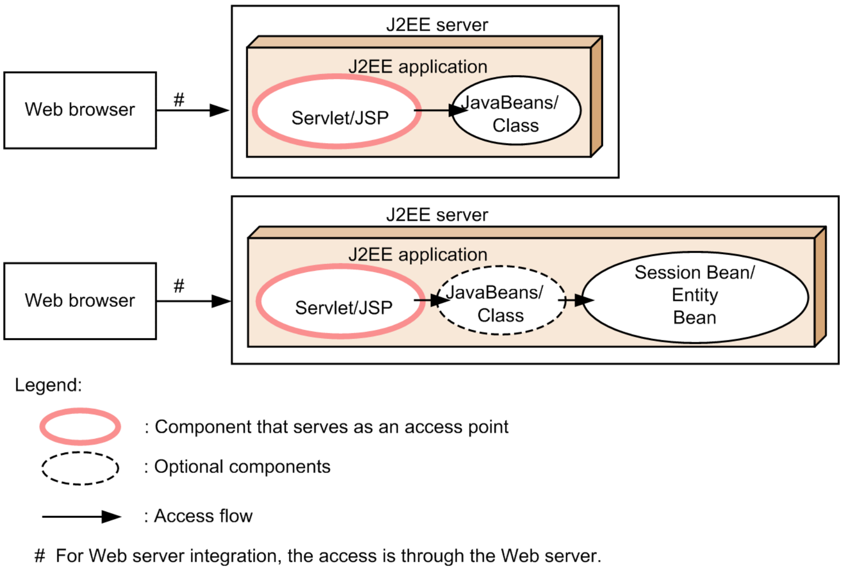
Figure 3–8: Application consisting of servlets and JSPs


J2EE server
Web browser
#
J2EE application
Servlet/JSP
Legend:
: Component that serves as an access point
: Access flow
# For Web server integration, the access is through the Web server.

Note: For other legend items, see *3.2 Description of the system configuration*.

Other components, such as JavaBeans and Java class, can be invoked from servlets or JSPs. Session Beans and Entity Beans can also be invoked. In such cases also, servlets or JSPs deployed in the front form the access point components as seen from the client side.

Figure 3–9: Access points when other components are invoked from servlets and JSPs



Note: For other legend items, see *3.2 Description of the system configuration*.

This application mainly runs on the Web front-end system.

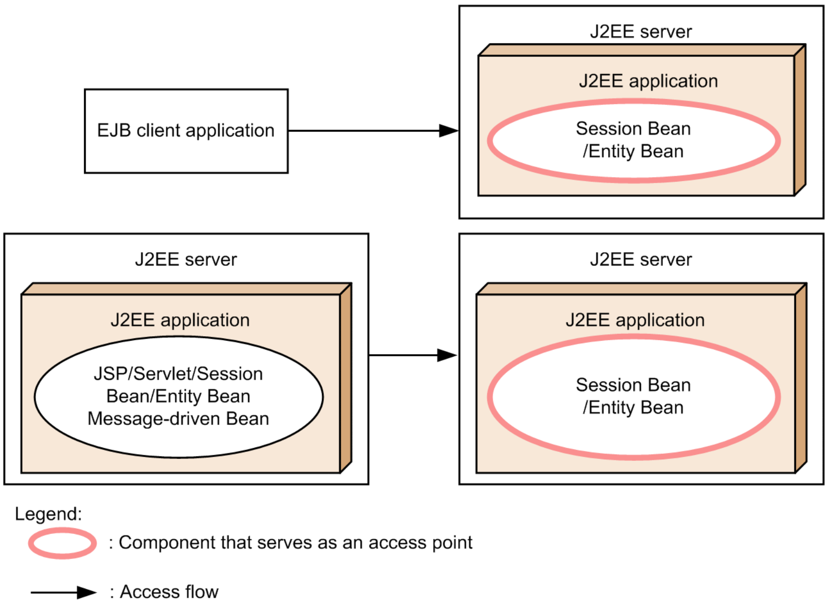
(2) Application consisting of Session Beans and Entity Beans

Session Beans and Entity Beans are components for implementing the business logic. They are accessed from the EJB client by RMI-IIOP. *EJB client* is a general name for the components used for invoking the Enterprise Bean. These components include the EJB client applications running on the client machine and the servlets, JSPs, Session Beans, Entity Beans, or Message-driven Beans running on another J2EE server.

In an application consisting of Session Beans and Entity Beans, the Session Beans or Entity Beans deployed in the front form the access point components.

The following figure shows an application consisting of Session Beans and Entity Beans:

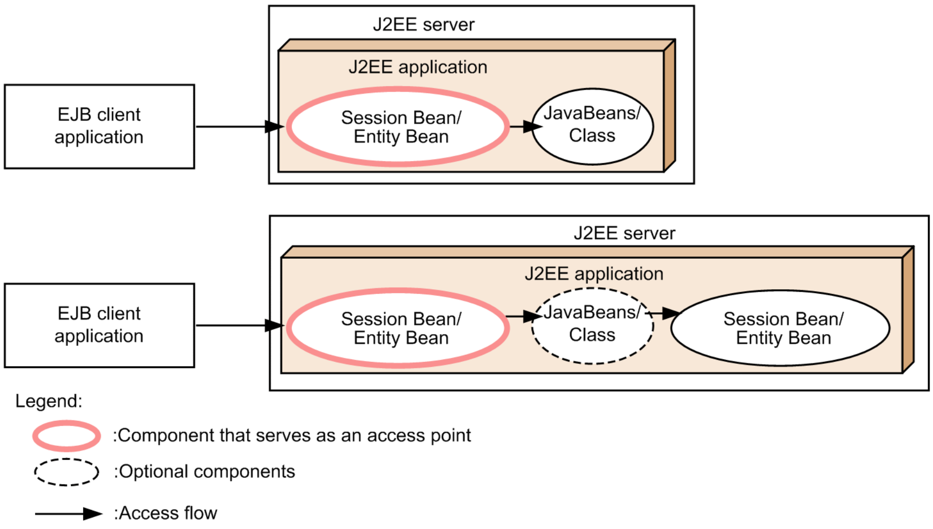
Figure 3–10: Application consisting of Session Beans and Entity Beans



Note: For other legend items, see *3.2 Description of the system configuration*.

Other components, such as JavaBeans and Java class, can be invoked from the Session Beans or Entity Beans. Other Session Beans and Entity Beans can also be invoked. In such cases also, the Session Beans or Entity Beans deployed in the front form the access point components as seen from the client.

Figure 3–11: Access points when other components are invoked from the Session Beans or Entity Beans



Note: For other legend items, see *3.2 Description of the system configuration*.

This application mainly runs on the back-end system.

(3) Application configured with Message-driven Beans

The Message-driven Beans are components for implementing the business logic in a message-driven system. The access can be in any of the following ways:

- Access through Cosminexus JMS provider
- Access through TP1/Message Queue and TP1/Message Queue - Access
- Access through the database (HiRDB or Oracle) and Cosminexus RM
- Access through TP1 inbound integration

In a system configuration with the Message-driven Bean as the access point, you require any one of the following as a resource adapter:

- CJMSP resource adapter#
- TP1/Message Queue - Access
- Cosminexus RM#

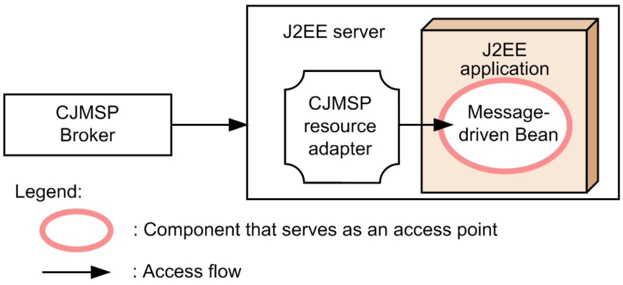
  Use Cosminexus RM in combination with the DB Connector for Cosminexus RM#
- TP1 inbound adapter#

# Resource adapter provided by Application Server.

In the case of an application consisting of Message-driven Beans, the Message-driven Beans form the access point components.

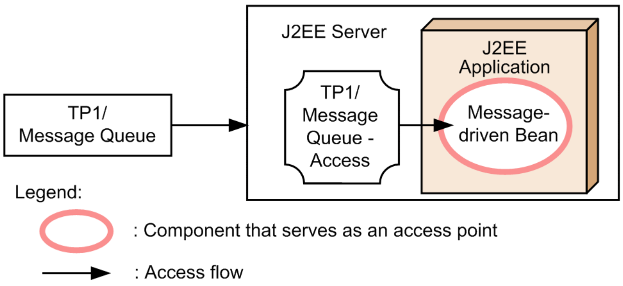
The following figures show an application consisting of Message-driven Beans:

Figure 3–12: Application configured with Message-driven Bean (through Cosminexus JMS provider)



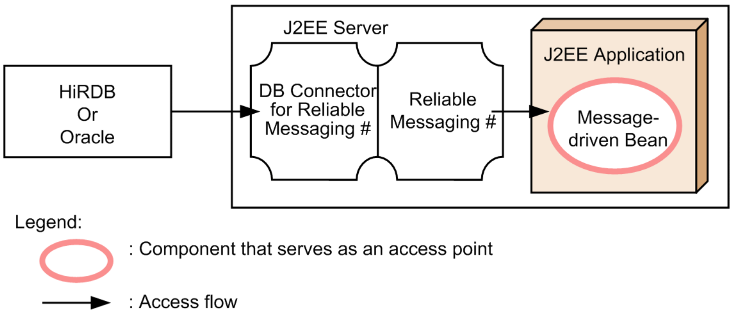
Note: For other legend items, see *3.2 Description of the system configuration*.

Figure 3–13: Application configured with Message-driven Bean (through TP1/Message Queue - Access)



Note: For other legend items, see *3.2 Description of the system configuration*.

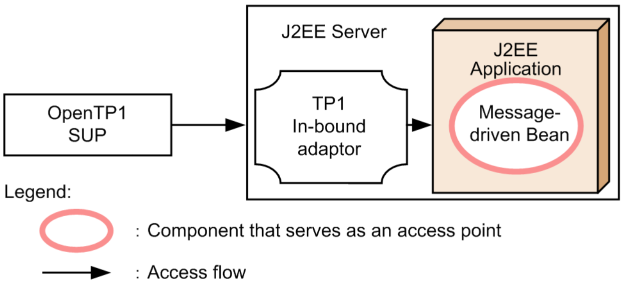
Figure 3–14: Application configured with Message-driven Bean (through Cosminexus RM)



Resource adapter provided with the Application Server.

Note: For other legend items, see *3.2 Description of the system configuration*.

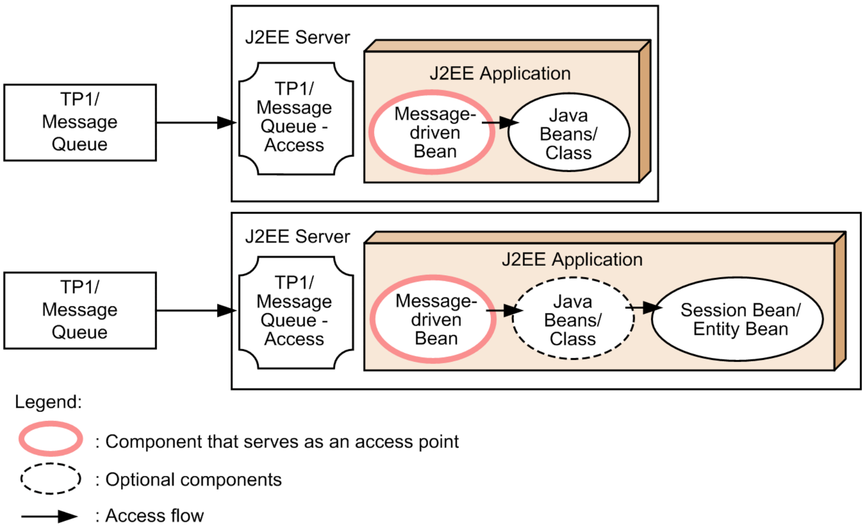
Figure 3–15: Application configured with Message-driven Bean (through TP1 inbound integration)



Note: For other legend items, see *3.2 Description of the system configuration*.

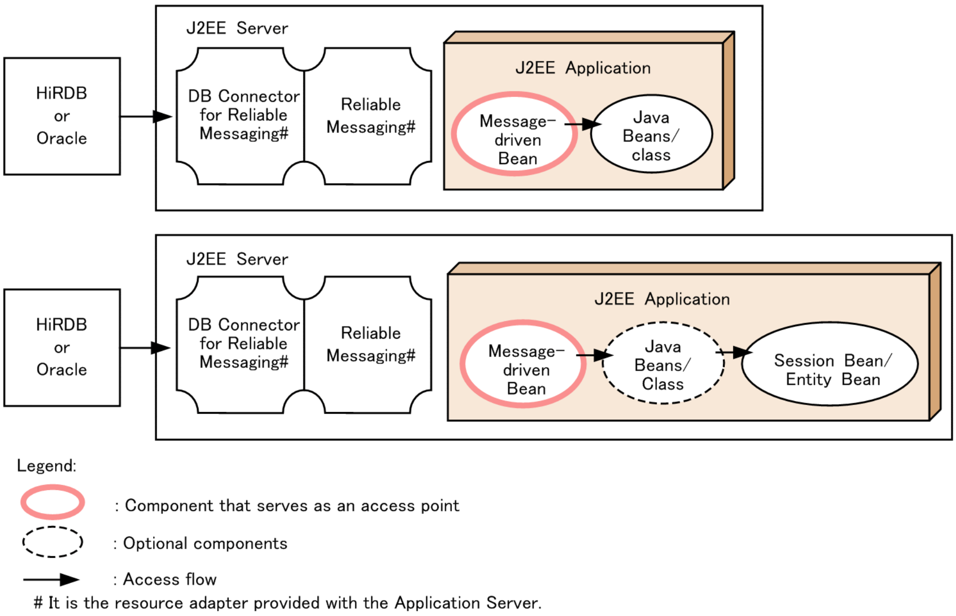
Other components, such as JavaBeans and Java class, can be invoked from the Message-driven Beans. Session Beans and Entity Beans can also be invoked. In such cases, component that is the access point as seen from the client, is the Message-driven Bean. The following figure shows an example:

Figure 3–16: Access point when invoking other components from Message-driven Bean (through TP1/Message Queue - Access)



Note: For other legend items, see *3.2 Description of the system configuration*.

Figure 3–17: Access points when other components are called from the Message-driven Beans (through Cosminexus RM)



Note: For other legend items, see *3.2 Description of the system configuration*.

This application mainly runs on the back-end system.

3.3.2 Resource types and resource adapters

You can connect with the following resources in the Application Server system:

- Database (HiRDB, Oracle, SQL Server or XDM/RD E2)
- TP1/Message Queue of OpenTP1
- SPP of OpenTP1
- SUP of OpenTP1
- CJMSP Broker of Cosminexus JMS provider
- Mail configuration
- JavaBeans resources

SUP of OpenTP1 is connected as Inbound. Note that the resource adapter is not required when you use a mail configuration or JavaBean Resources.

The resource adapters used for various types of resources connected by the application are explained as follows:

> **Tip**
>
> In Application Server, you can use resource adapters conforming to Connector 1.0 or Connector 1.5 specifications.
>
> For details about using resource adapters other than those described here, check the description for the resource adapter being used.

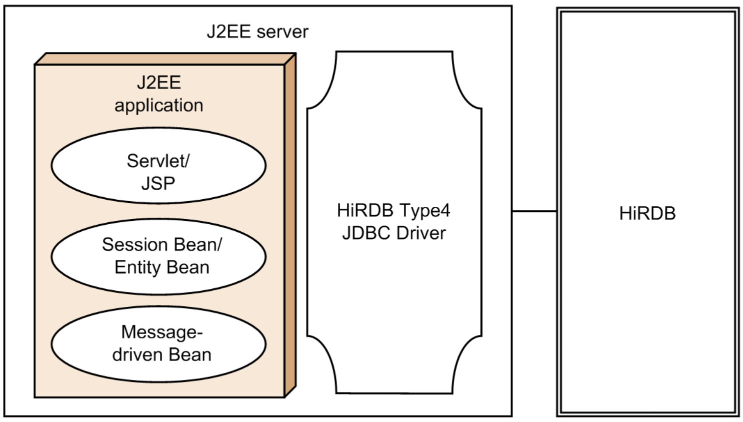
(1) Resource adapter for connecting to the database (when the JDBC interface is used)

Use *DB Connector* as the resource adapter when connecting to the database using the JDBC interface. When you use the DB Connector, you can use the JDBC interface for accessing the database from servlets, JSPs, Session Beans, Entity Beans, or Message-driven Beans.

Application Server can access HiRDB, Oracle, SQL Server, or XDM/RD E2 databases using resource adapters. The following figures show the respective configurations when you access each type of database with a resource adapter:

- **Configuration for accessing HiRDB**
  When connecting to HiRDB, the HiRDB Type4 JDBC Driver is required on the same machine where a J2EE server is installed.
  The following figure shows the configuration for accessing HiRDB:

Figure 3–18: Configuration for accessing HiRDB with a resource adapter (for using the HiRDB Type4 JDBC Driver)



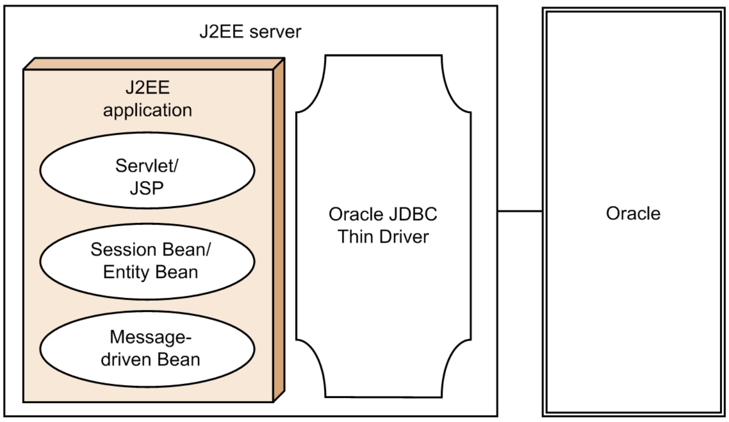
Note: For other legend items, see *3.2 Description of the system configuration*.

- **Configuration when accessing Oracle**

  When connecting to Oracle, Oracle JDBC Thin Driver is required on the same machine where the J2EE server is installed.

  The following figure shows the configuration for accessing Oracle:

Figure 3–19: Configuration for accessing Oracle with a resource adapter (for using the Oracle JDBC Thin Driver)



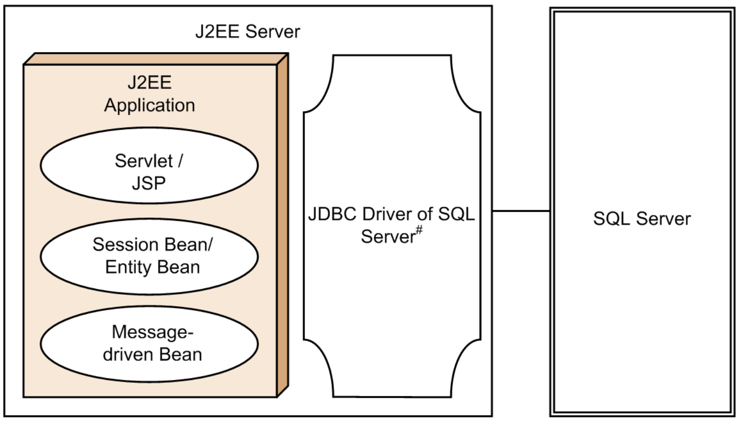
Note: For other legend items, see *3.2 Description of the system configuration*.

- **Configuration for accessing the SQL Server**

  When connecting to SQL Server, the JDBC driver for SQL Server is required on the same machine as the J2EE server.

  The following figure shows the configuration when accessing the SQL Server:

Figure 3–20: Configuration for accessing the SQL Server with a resource adapter



It serves as SQL Server 2005 JDBC Driver while connecting to SQL Server 2005.

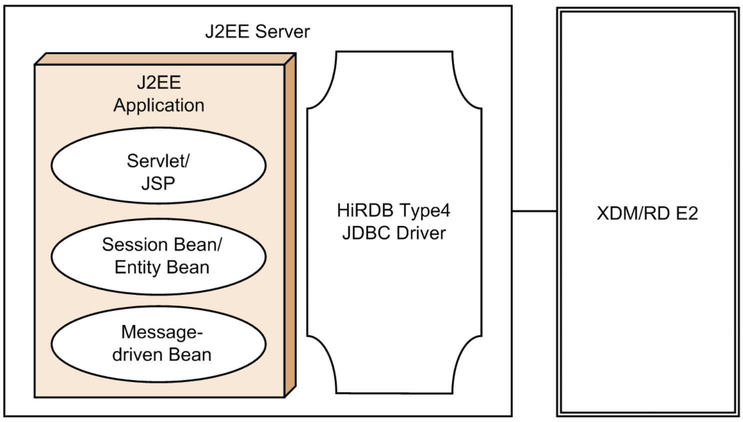
Note: For other legend items, see *3.2 Description of the system configuration*.

- **Configuration for accessing XDM/RD E2**

  When connecting to XDM/RD E2, the HiRDB Type4 JDBC Driver is required on the same machine where the J2EE server is installed.

  The following figure shows the configuration required when accessing XDM/RD E2:

Figure 3–21: Configuration when accessing XDM/RD E2 with a resource adapter (for using the HiRDB Type4 JDBC Driver)



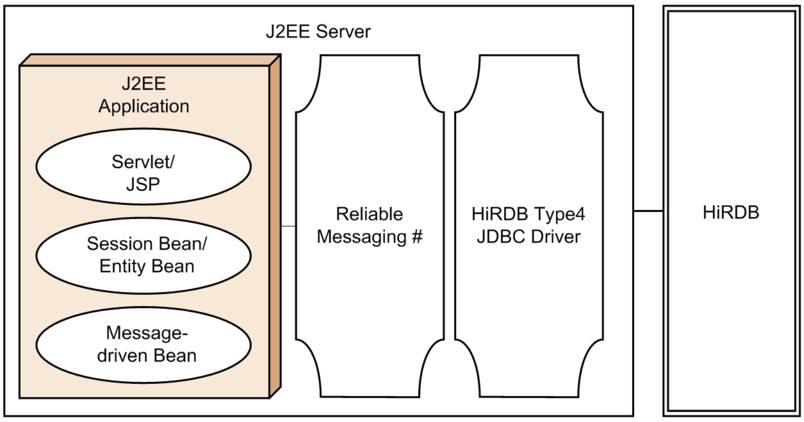
Note: For other legend items, see *3.2 Description of the system configuration*.

(2) Resource adapter for connecting to the database (when the JMS interface is used)

If you are using the JMS interface for connecting to the database, use *DB Connector for Cosminexus RM* as the resource adapter. DB Connector for Cosminexus RM is a resource adapter for integrating with Cosminexus RM. If you use Cosminexus RM, you can access the queues implemented on the database from a servlet, JSP, Session Bean, Entity Bean, or Message-driven Bean, by using the JMS interface. In such a case, the Cosminexus RM libraries run on the J2EE server and the queue data is saved in the database. Cosminexus RM is component software of Application Server.

The following figures separately show the configuration for accessing the HiRDB and Oracle databases with Cosminexus RM:

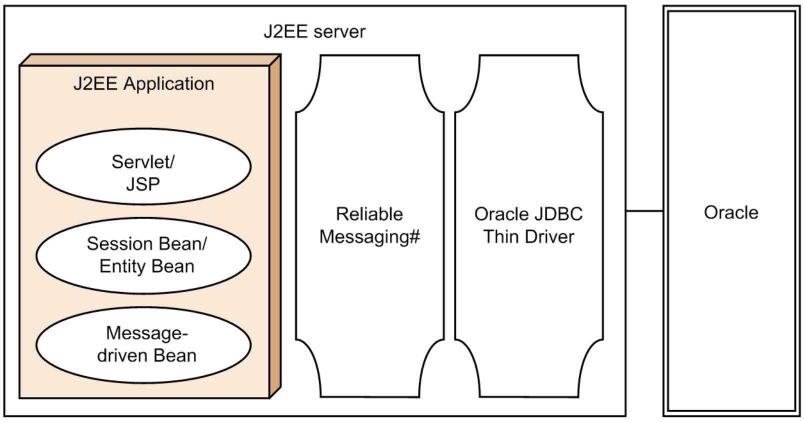
Figure 3–22: Configuration for accessing HiRDB with Cosminexus RM



Resource adapter provided with the Application Server.

Note: For other legend items, see *3.2 Description of the system configuration*.

Figure 3–23: Configuration for accessing Oracle with Cosminexus RM



This is a resource adapter provided with the Application Server.

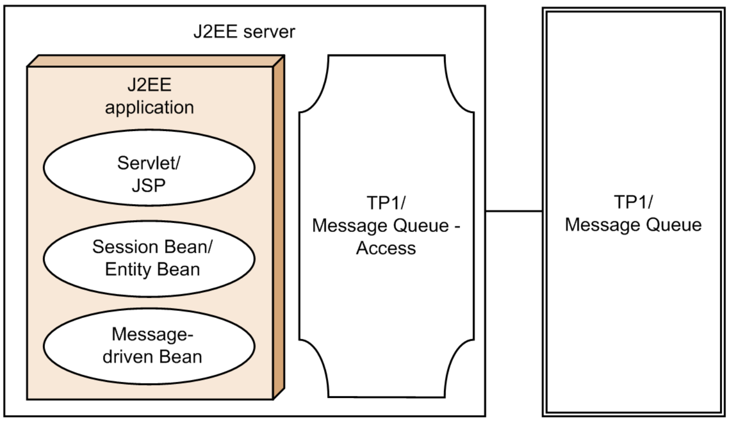
Note: For other legend items, see *3.2 Description of the system configuration*.

(3) Resource adapter for connecting to TP1/Message Queue of OpenTP1

When connecting to TP1/Message Queue, which is the message queuing function of OpenTP1, use *TP1/Message Queue - Access* as the resource adapter. If you use TP1/Message Queue - Access, TP1/Message Queue can be accessed using a JMS interface from a servlet, JSP, Session Bean, Entity Bean, or Message-driven Bean. In such cases, TP1/Message Queue - Access is required on the same machine as the J2EE server.

The following figure shows the configuration for accessing TP1/Message Queue using resource adapter:

Figure 3–24: Configuration when accessing TP1/MessageQueue using resource adapter



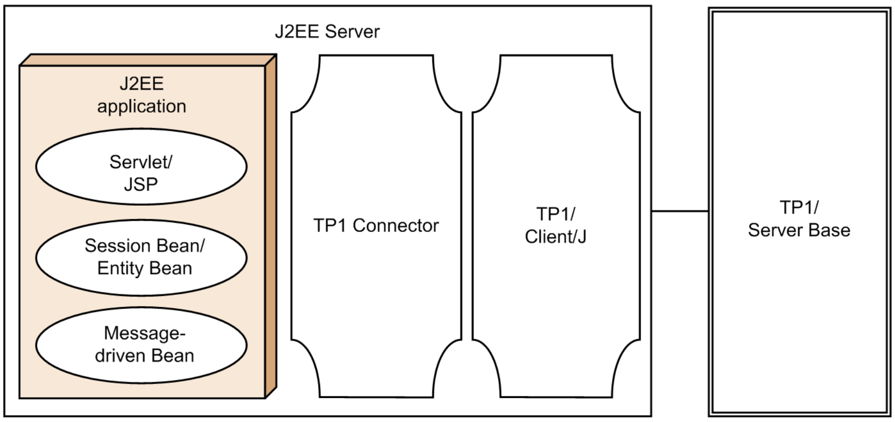
Note: For other legend items, see *3.2 Description of the system configuration*.

(4) Resource adapter for connecting to SPP of OpenTP1

When connecting to SPP (Service Providing Program) of OpenTP1, use *uCosminexus TP1 Connector* and *TP1/Client/J* as resource adapters. If you are using uCosminexus TP1 Connector and TP1/Client/J, access to SPP of OpenTP1 is possible from a servlet, JSP, Session Bean, Entity Bean, or Message-driven Bean. In such cases, uCosminexus TP1 Connector and TP1/Client/J must be on the same machine as the J2EE server.

The following figure shows the configuration for accessing SPP of OpenTP1 using resource adapter:

Figure 3–25: Configuration when accessing SPP of OpenTP1 using resource adapter



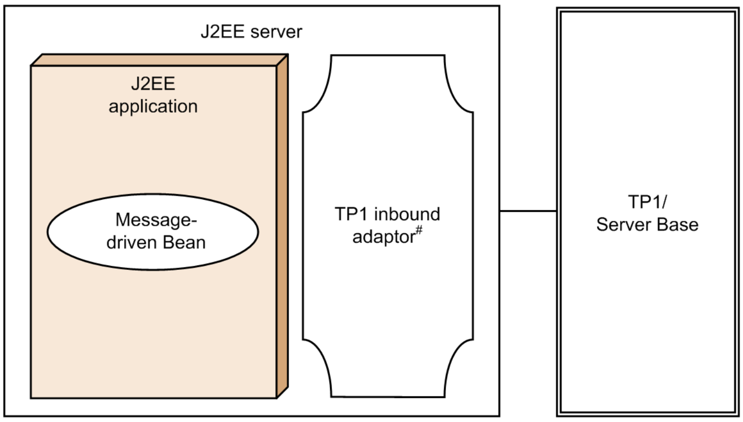
Note: For other legend items, see *3.2 Description of the system configuration*.

(5) Resource adapter for connecting to SUP of OpenTP1

When connecting to SUP (Service Using Program) of Open TP1 by the TP1 inbound integrated function, use *TP1 inbound adapter* as the resource adapter. If you use the TP1 inbound integrated function, you can access a Message-driven Bean running on a J2EE server as inbound from SUP of OpenTP1. In such cases, a TP1 inbound adapter must be on the same machine as the J2EE server.

The following figure shows the configuration when accessing as inbound from SUP of OpenTP1 using resource adapter:

Figure 3–26: Configuration when accessing as inbound from SUP of OpenTP1 using resource adapter



#1 Resource adapter provided with the Application Server.

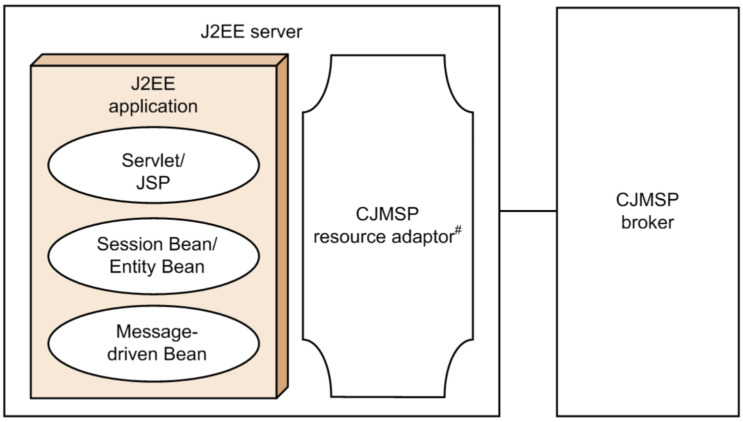
Note: For other legend items, see *3.2 Description of the system configuration*.

(6) Resource adapter for connecting to CJMSP Broker of Cosminexus JMS provider

When connecting to CJMSP Broker of Cosminexus JMS provider, use *CJMSP resource adapter* as the resource adapter. If you use the CJMSP resource adapter, you can access CJMSP Broker using the JMS interface from servlets, JSPs, Session Beans, Entity Beans, or Message-driven Beans. In such cases, a CJMSP resource adapter must be on the same machine as the J2EE server.

The following figure shows the configuration for accessing CJMSP Broker using resource adapter:

Figure 3–27: Configuration when accessing CJMSP Broker using resource adapter



Resource adapter provided with the Application Server.

Note: For other legend items, see *3.2 Description of the system configuration*.

3.4 Determining the configuration of the client and the server

This section describes the configuration types for the client and server, the processes deployed on each machine in the respective cases and the features of the respective configurations.

To determine the configuration of the client and server, you need to decide the relationship between the client side that calls the procedure and the server side that receives the call. You then need to identify the access points of the application running on the server side.

The client and server configurations where the following components act as access points are explained below. For configurations that use servlets and JSPs as access points, separate explanations are provided for systems that use Web server integration and systems that access the NIO HTTP server of the J2EE server directly without going through a Web server.

- Servlet and JSP
- Session Bean and Entity Bean
- Stateless Session Bean when using CTM

Hereafter, the machine with Application Server deployed is called the *Application Server machine* and the machine that is used as the client is called the *client machine*.

3.4.1 Configuration with servlets and JSPs as access points (for Web server integration)

This subsection describes a Web-based system configuration with servlets and JSPs as the access points. This configuration is called the *Web client configuration*. You can use this configuration in a Web front-end system. The configuration for Web server integration is explained here.

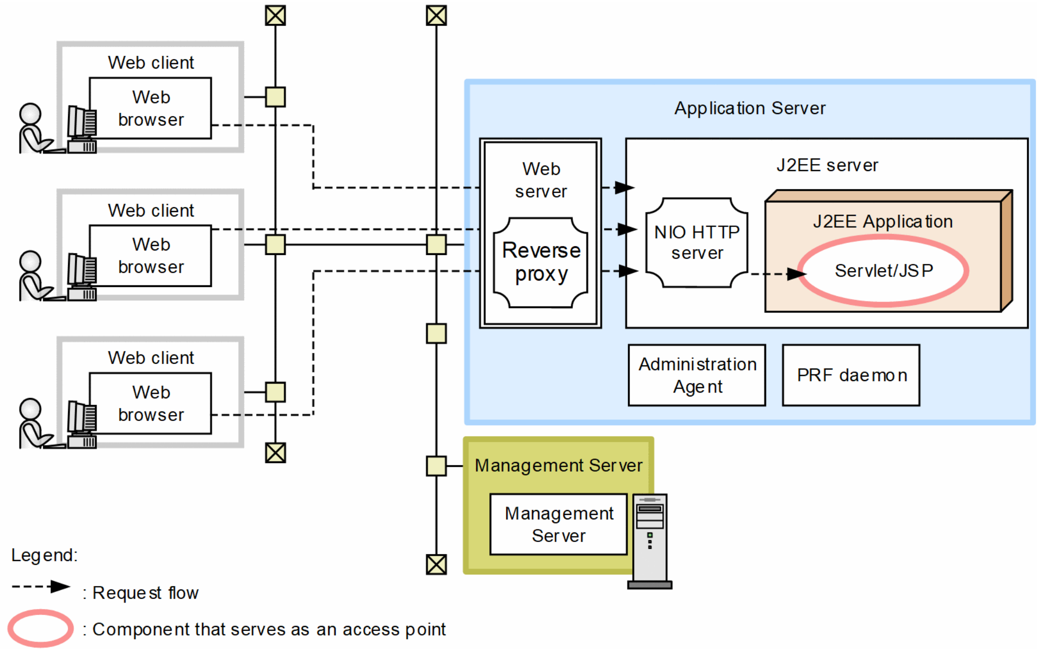
When connecting to the Internet with this configuration, from the security point of view Hitachi recommends that you deploy a Web server with a reverse proxy function on the DMZ, apart from deploying the Web server on the same machine as the Application Server. For details, see *3.3 Deployment of reverse proxies in a DMZ* in the *uCosminexus Application Server Security Management Guide*.

(1) Features of the system configuration

This system configuration is applicable in a Web front-end system, when the requests sent from the Web browser are processed in Application Server.

You can build the most basic web client configuration by using client machines and one Application Server machine. The following figure shows an example of a web client configuration built with one Application Server machine:

Figure 3–28: Example of a web client configuration built with one Application Server machine (in the case of Web server integration)



Note: For other legend items, see *3.2 Description of the system configuration*.

Feature

A Web browser is the only software used in the client.

Flow of requests

The servlets and JSPs forming the access points run on the J2EE server. These components are accessed from the Web browser through the Web server.

(2) Processes and required software on the respective machines

This subsection describes the software and the processes required on the respective machines. For details about the processes required for connecting to the resources, see *3.6 Determining the transaction type*.

(a) Application server machine

You need to install Application Server on the Application Server machine.

The processes required are as follows:

- Web server
- J2EE server
- Administration Agent
- PRF daemon

Application Server contains the Web server, Cosminexus HTTP Server. In Windows, you can also use Microsoft IIS as the Web server. In this case, you need Microsoft IIS software.

(b) Management Server machine

You need to install Application Server on the Management Server machine.

Invoke the following process:

- Management Server

(c) Web client machine

The Web client machine requires a Web browser.

3.4.2 Configuration where servlets and JSPs are used as access points (when accessing the NIO HTTP server directly)

This subsection describes the Web-based system configuration that uses servlets and JSPs as access points. It also explains the configuration of a system that accesses the NIO HTTP server directly.

When connecting to the Internet with this configuration, set up the deployment of a Web server having a reverse proxy function on the DMZ from the security point of view. For details, see *3.3 Deployment of reverse proxies in a DMZ* in the *uCosminexus Application Server Security Management Guide*.

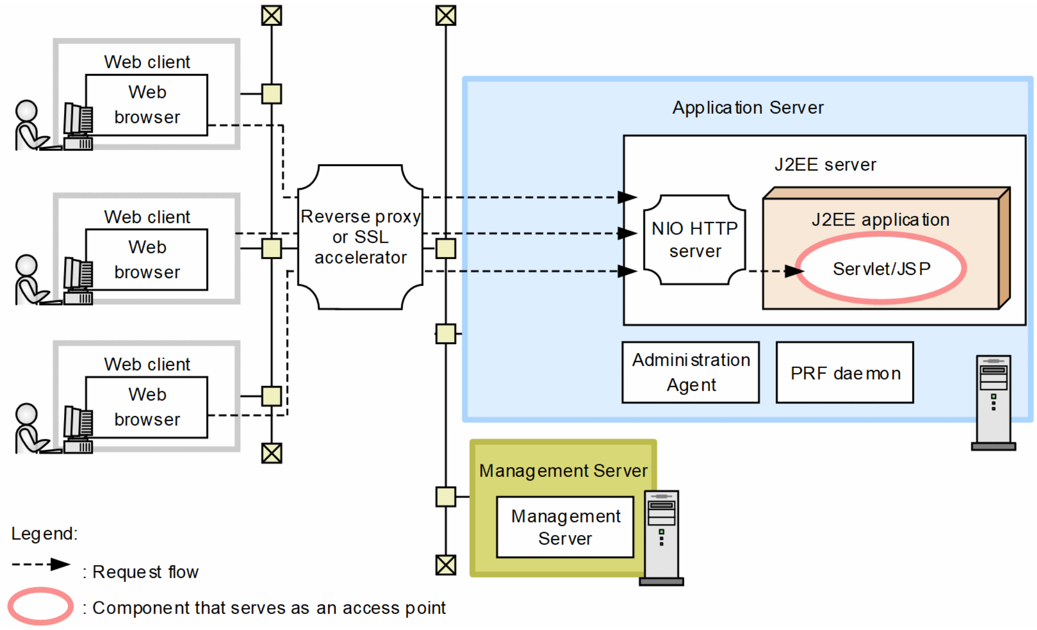
(1) Features of the system configuration

A system configuration applied when processing requests sent from the Web browser on the Application Server, in Web front-end systems.

The most basic Web client configuration can be built with a client machine and one Application Server machine. When accessing the NIO HTTP server directly, there is no need to run a Web server in front of the J2EE server. Use functions of HTTP servers on J2EE servers.

The following figure shows an example of a Web client configuration where the NIO HTTP server is accessed directly.

Figure 3–29: Example of Web client configuration built on one Application Server machine (when accessing the NIO HTTP server directly)



Note: For other legend items, see *3.2 Description of the system configuration*.

Features

A Web browser is the only software used by clients.

From the Web browser, direct access to the J2EE server is possible without using Web servers, and hence, the Web browser has a merit of higher performance. Moreover, operations are simple because you need not start a Web server.

Keep the following in mind when configuring a system that accesses the NIO HTTP server directly:

- The NIO HTTP server only supports the minimum functionality needed to operate as a Web server. For this reason, a configuration that accesses the NIO HTTP server directly is unsuitable for systems that require a wide array of Web server functions. Select the configuration that integrates with the Web server if your system requires these functions. For details about the functions available with the NIO HTTP server, see the manual *uCosminexus Application Server Web Container Functionality Guide*.
- The NIO HTTP server does not support HTTPS. Therefore, for using SSL, use the SSL function of the Web server embedded with the proxy module deploying proxy server at the front-end or deploying SSL accelerator at the front-end.
- From the security point of view, deploy a reverse proxy at front-end, for systems connected to Internet. For details about the configuration deploying a reverse proxy, see *3.3 Deployment of reverse proxies in a DMZ* in the *uCosminexus Application Server Security Management Guide*.

Request flow

Operate servlets and JSPs that are used as access points, on the J2EE server. These components are directly accessed from the Web browser.

(2) Process and software required for each machine

This subsection describes the software and processes required on the respective machines. For details about processes required to connect to resource, see *3.6 Determining the transaction type*.

(a) Application server machine

You must install Application Server on the Application Server machine.

The required processes are as follows:

- J2EE server
- Administration Agent
- PRF daemon

(b) Management Server machine

You must install Application Server on the Management Server machine.

The process to be started is as follows:

- Management Server

(c) Web client machine

A Web client machine must have a Web browser.

3.4.3 Configuration with Session Beans and Entity Beans as access points

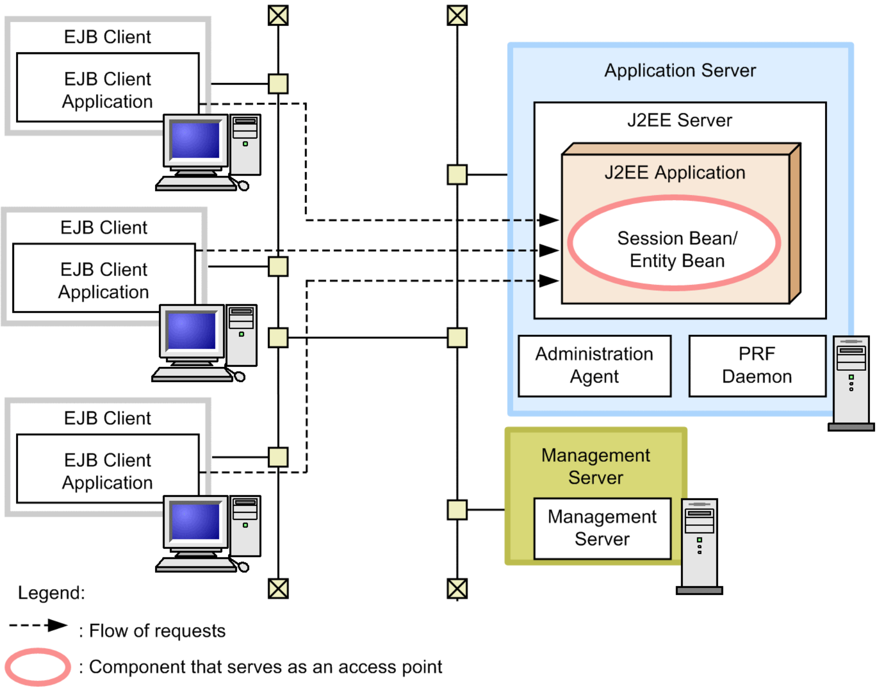
This subsection describes a system configuration where the Session Beans and Entity Beans act as access points and an EJB client application operates on a client. This configuration is called an *EJB client configuration*.

(1) Features of the system configuration

You can build the most basic EJB client configuration by using client machines and one Application Server machine.

The following figure shows an example of an EJB client configuration built with one Application Server machine:

Figure 3–30: Example of EJB client configuration built with one Application Server machine



Note: For other legend items, see *3.2 Description of the system configuration*.

Feature

In Windows, you can use uCosminexus Client to build the environment on the client machine.

Flow of requests

The Session Beans and Entity Beans forming the access points run on the J2EE server. The requests from the EJB client application are sent to the access points by RMI-IIOP to invoke a Session Bean and Entity Bean. At this point, the EJB client application searches (lookup) for names from the Naming Service running on the J2EE server and accesses a Session Bean and Entity Bean.

(2) Required software and processes to be invoked on the respective machines

This subsection describes the required software and the processes to be invoked on the respective machines. For details about the processes used for connecting to the resources, see *3.6 Determining the transaction type*.

(a) Application server machine

You need to install Application Server on the Application Server machine.

Invoke the following processes:

- J2EE server
- Administration Agent
- PRF daemon

(b) Management Server machine

You need to install Application Server on the Management Server machine.

Invoke the following process:

- Management Server

(c) EJB client machine

You need to install Application Server or uCosminexus Client (in the case of Windows) on the EJB client machine.

The process thus invoked is the process of the EJB client application.

3.4.4 Configuration where Stateless Session Bean is used as access point when using CTM

This subsection describes about the system configuration where a Stateless Session Bean is used as the access point when you use CTM.

Reference note

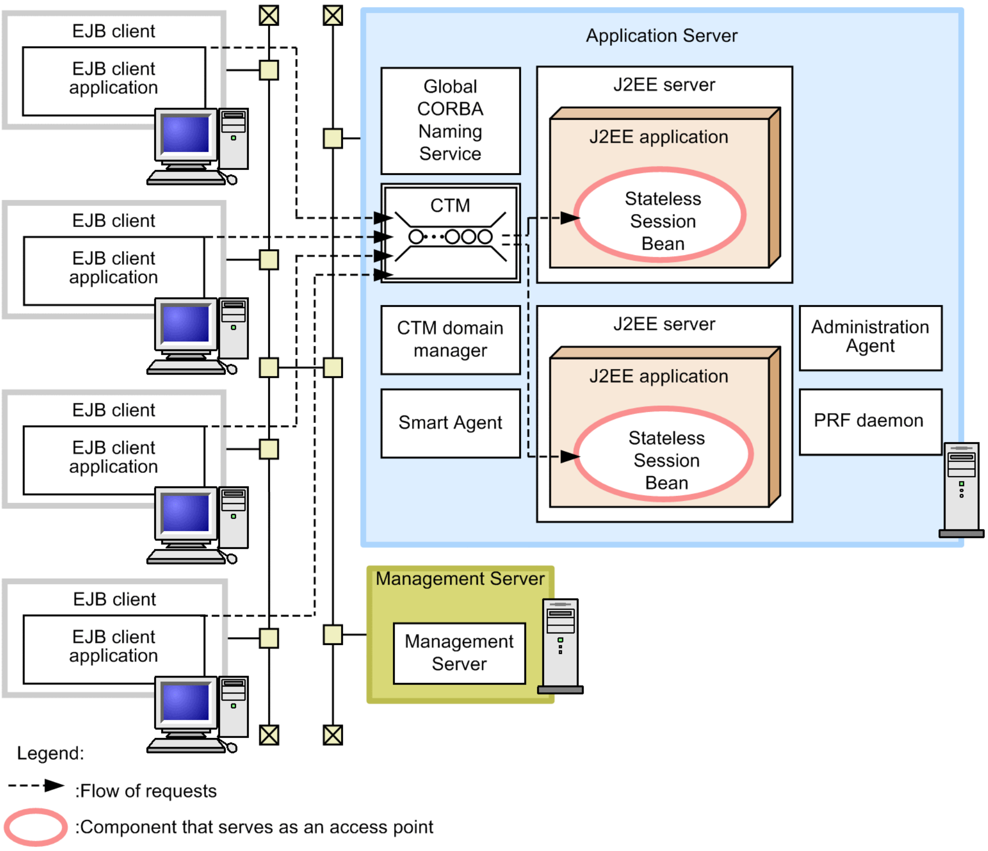
An example of EJB client configuration is explained here where an EJB client application is used as the client. When you use the CTM gateway function, you can implement a configuration where an EJB application on the J2EE server is directly invoked from a client application other than an EJB client. For details about the configurations, see *3.13.3 Configuration in which a Stateless Session Bean is invoked from other than an EJB client using the CTM gateway function*.

(1) Features of the system configuration

This system configuration is one of the EJB client configurations. The scheduled requests are transmitted by CTM to a Stateless Session Bean, which is the access point.

The following figure shows an example of EJB client configuration when using CTM:

Figure 3–31: Example of EJB client configuration when using CTM



Note: For other legend items, see *3.2 Description of the system configuration*.

Features

- Request scheduling is executed by the CTM and the system operations such as executing service lock and controlling the concurrent execution count can be executed.
- Start two or more J2EE servers. If an error occurs on a J2EE server, the system operation can be continued by executing degeneration operation.
- In Windows, the environment of the client machine can be built using uCosminexus Client.

Requests flow

A Stateless Session Bean, which is the access point, operates on the J2EE server.

The requests from the EJB client application are transmitted through the CTM and invoke the Stateless Session Bean. In such cases, the EJB client application looks up the name from the global CORBA Naming Service and accesses Stateless Session Bean.

(2) Required software and processes to be started on each machine

This subsection describes the required software and processes to be started on the respective machines. For details about the processes to connect to a resource, see *3.6 Determining the transaction type*.

(a) Application server machine

You must install Application Server on the Application Server machine.

The processes to be started are as follows:

- J2EE server
- Administration Agent
- PRF daemon
- Global CORBA Naming Service
- CTM processes (CTM daemon and CTM regulator)
- CTM domain manager
- Smart Agent

(b) Management Server machine

You must install Application Server on Management Server machines.

The process to be started is as follows:

- Management Server

(c) EJB client machine

You must install Application Server or uCosminexus Client (in Windows) on EJB clients.

The process to be started is the processing for EJB client application.

3.5 Determining integration between servers

This section describes the configuration types for integrating the applications running on J2EE servers that run on multiple Application Server machines, and also describes the processes deployed on each machine for respective configuration types. This section also describes the features of the respective configurations.

In a system that consists of multiple Application Server machines, the Application Server machine on which the caller application is running is called the *client-side application server* and the Application Server machine on which the invoked application is running is called the *server-side application server*.

This subsection describes the configuration involving integration between the following two servers:

- Integration between servers that invoke Session Beans and Entity Beans
- Integration between the servers that invoke Stateless Session Bean through CTM

3.5.1 Integration between servers invoking Session Bean and Entity Bean

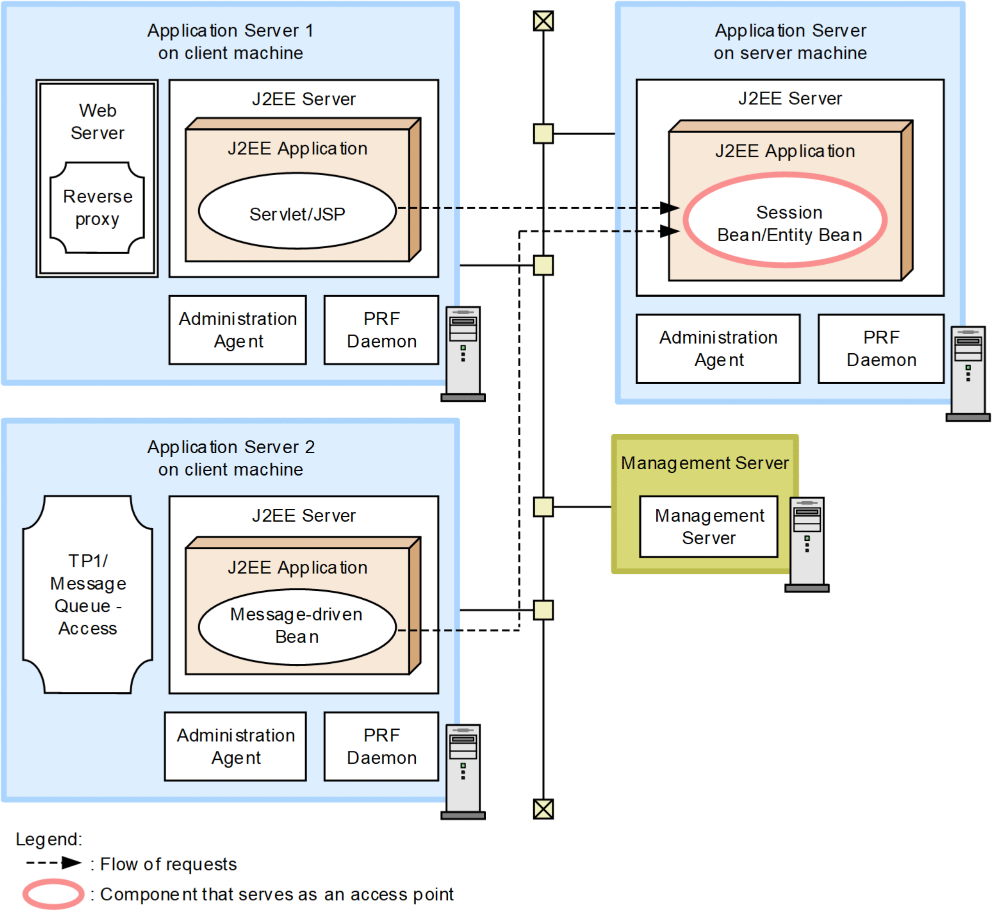
This subsection describes the configuration for invoking Session Beans and Entity Beans from the J2EE server that runs on another Application Server machine.

(1) Features of the system configuration

In this configuration, the Application Server instance on the client side invokes the Application Server instance on the server side. An application consisting of servlets, JSPs, Entity Beans, Session Beans, or Message-driven Beans runs on the client-side application server. The access point component of the server side is either a Session Bean or an Entity Bean. The invocation from the client-side application is executed by RMI-IIOP.

The following figure shows an example configuration of integration between servers that invoke Session Beans and Entity Beans:

Figure 3–32: Example of the configuration for integration between servers that invoke Session Bean and Entity Bean



Note: For other legend items, see *3.2 Description of the system configuration*.

Features

- You can use this configuration for integrating the applications. Use the configuration for invoking the server application from multiple client applications.
- You can also use this configuration for integrating the systems. If an already built client system and server system are changed as per this configuration, you can invoke the applications between the systems.

Flow of requests

The client side J2EE server sends the requests to the Session Beans or Entity Beans forming the server side access points.

At this point, the Session Beans or Entity Beans are accessed in the client side by looking up the names from the CORBA Naming Service that is invoked as an in-process in the J2EE server on the server-side application server.

(2) Required software and the processes to be invoked in the respective machines

This subsection provides information about the required software and the processes to be invoked in the respective machines when integrating the servers that invoke Session Beans or Entity Beans. For details about the processes required for connecting to the resources, see *3.6 Determining the transaction type*.

(a) Client-side application server machine (execution environment of servlets or JSPs)

You need to install Application Server on client-side application server machine that operates the servlets and JSPs.

Invoke the following processes:

- Web server
- J2EE server
- Administration Agent
- PRF daemon

(b) Client-side application server machine (execution environment of Message-driven Beans)

You need to install Application Server on the client-side application server machine that operates the Message-driven Bean. To use TP1/Message Queue - Access as a resource adapter, you must install TP1/Message Queue - Access. If you use Cosminexus RM, you need not install TP1/Message Queue - Access separately because TP1/Message Queue - Access is included in Application Server.

Invoke the following processes:

- J2EE server
- Administration Agent
- PRF daemon

(c) Server-side application server machine

You need to install Application Server on the server-side application server machine.

Invoke the following processes:

- J2EE server
- Administration Agent
- PRF daemon

(d) Management Server machine

You need to install Application Server on the Management Server machine.

Invoke the following process:

- Management Server

3.5.2 Integration between the servers that invokes Stateless Session Bean through CTM

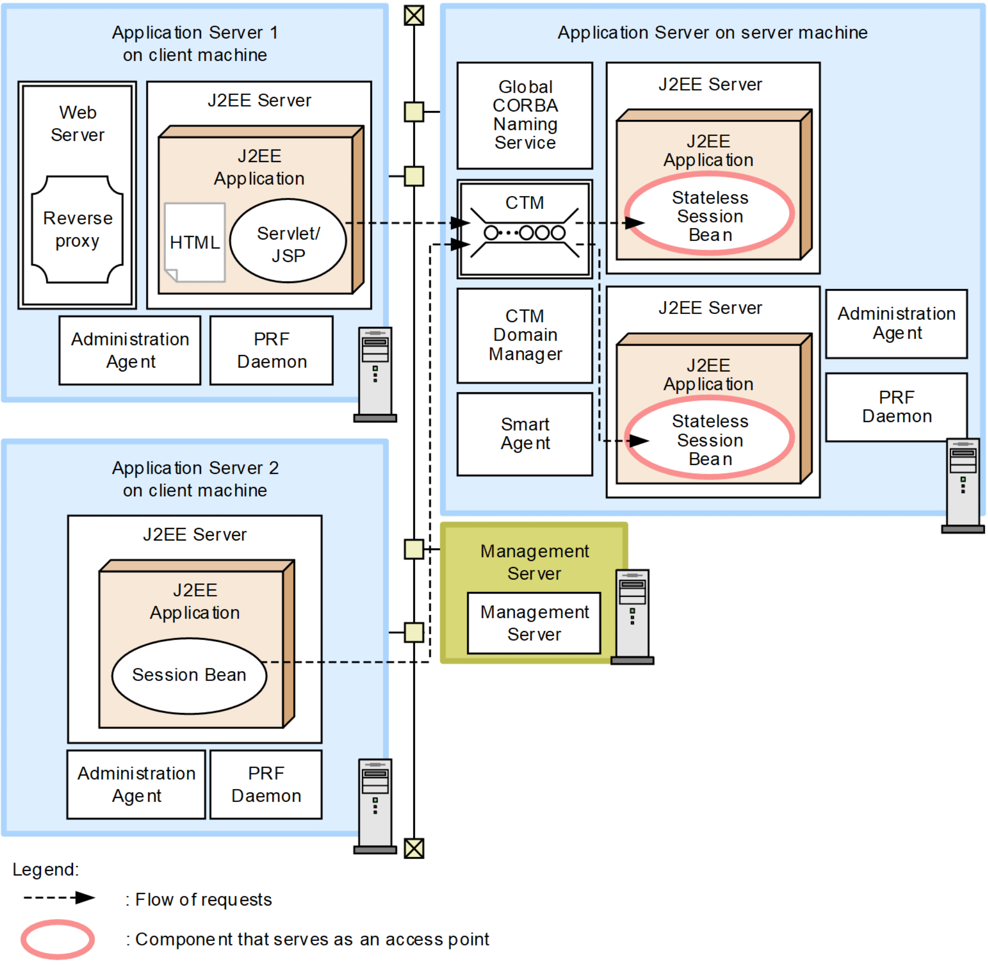
This subsection describes a configuration using CTM, when you invoke Stateless Session Beans from a J2EE server exists on another Application Server through CTM.

(1) Features of the system configuration

This is the configuration in which the Application Server on the server side is invoked from the Application Server on the client side through CTM. The application configured with servlet, JSP, Entity Bean. Session Bean, or Message-driven Bean operates on the Application Server at the client side. Stateless Session Bean is the access point component at server side. Invoking from the application at client side is executed by RMI-IIOP.

The following figure shows an example of the configuration of integration between the servers that invoke Stateless Session Bean through CTM:

Figure 3–33: Example of configuration of integration between servers that invoke Stateless Session Bean



Note: For other legend items, see *3.2 Description of the system configuration*.

Features

- This is a configuration used in the integration between applications. This configuration is suitable when invoking a server application from multiple client applications.
- This can also be used for integration between the systems. Since this system configuration includes the already built client side system and the server side system, the application can be invoked between the systems.

Request flow

The requests of a Stateless Session Bean, which is the access point at server side, are sent from the J2EE server at client side through CTM.

The name from the global CORBA Naming Service of the server-side application is looked up at the client side and the Stateless Session Bean is accessed through CTM.

The request through CTM can be appropriately distributed to a Stateless Session Bean running on the J2EE server by CTM.

(2) Required software and process to be started on each machine

This subsection describes the required software and the processes to be started on the respective machine when integrating between the servers that invoke Stateless Session Beans through CTM. For details about the processes required for connecting to the resources, see *3.6 Determining the transaction type*.

(a) Application server machine at client side (Execution environment of Servlet/JSP)

You must install Application Server on the Application Server machine at client side where the servlet and JSP operate.

The processes to be started are as follows:

- Web server
- J2EE server
- Administration Agent
- PRF daemon

(b) Application server machine at client side (Execution environment of Session Bean)

You must install Application Server on the Application Server machine at client side that operates Session Bean.

The processes to be started are as follows:

- J2EE server
- Administration Agent
- PRF daemon

(c) Application server machine at server side

You must install Application Server on the Application Server machine at server side.

The processes to be started are as follows:

- J2EE server
- Administration Agent
- PRF daemon
- Global CORBA Naming Service
- CTM processes (CTM daemon and CTM regulator)
- CTM domain manager
- Smart Agent

(d) Management Server machine

You must install Application Server on the Management Server machine.

The process to be started is as follows:

- Management Server

3.6 Determining the transaction type

This section describes the system configuration for each transaction type.

Use a local transaction when only one resource participates in the transaction. Use a global transaction when multiple resources participate in the transaction. In an integrated server configuration, you can also use transaction context propagation, when the respective J2EE servers connect to different resources.

For details about the mapping between the connected resources and the resource adapters, see *3.3.2 Resource types and resource adapters*.

Reference note

Although this section describes the configuration of each transaction type that is started by Application Server, you can also start a transaction by using a EJB client application. When starting a transaction by using an EJB client application, you will require Application Server on the EJB client machine.

3.6.1 Configuration when using a local transaction

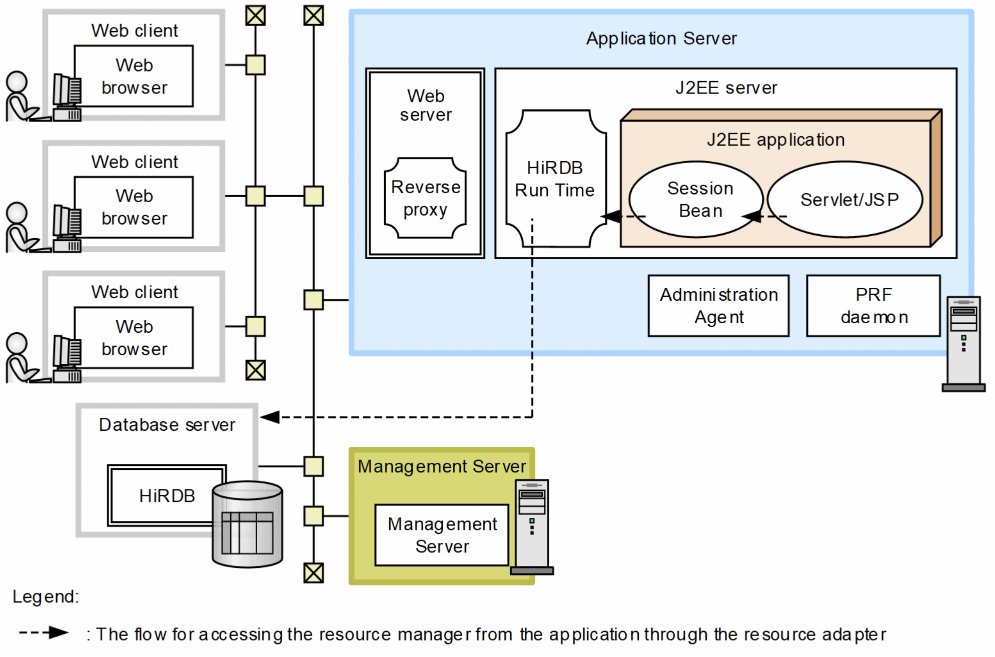
This subsection describes the configuration when only one resource participates in the transaction.

(1) Features of the system configuration

In this configuration, one resource manager is accessed from a single J2EE application consisting of servlets, JSPs, and Session Beans, through a resource adapter. The resource manager manages the transaction used for accessing the resource manager from the application. The transaction type is a local transaction that does not use an XA interface.

The following figure shows an example configuration when using a local transaction:

Figure 3–34: Example configuration when using a local transaction



Note: For other legend items, see *3.2 Description of the system configuration*.

Features

In this configuration, one resource manager will be accessed from the application. A two-phase commit is not required.

Flow of accessing the resource manager from the application through the resource adapter

Servlets and JSPs that are accessed from the Web browser through the Web server locally invoke a Session Bean existing in the same application. A Session Bean accesses the resource manager (HiRDB in the case of the figure) through the resource adapter.

(2) Required software and the processes to be invoked on the respective machines

This subsection describes the required software and the processes to be invoked on the respective machines when you use a local transaction:

(a) Application server machine

You need to install Application Server on the Application Server machine.

The following software is required for connecting to the resource manager:

Resource manager to be connected to	Required software
HiRDB	HiRDB Run Time or HiRDB Type4 JDBC Driver
Oracle	Oracle Client or Oracle JDBC Thin Driver
SQL Server	JDBC driver for SQL Server
XDM/RD E2	HiRDB Run Time or HiRDB Type4 JDBC Driver
TP1/Message Queue	TP1/Message Queue - Access
SPP of OpenTP1	uCosminexus TP1 Connector TP1/Client/J

Invoke the following processes:

- Web server
- J2EE server
- Administration Agent
- PRF daemon

(b) Machine on which the resource manager is running

Install any of the following software on the machine on which the resource manager is running:

- HiRDB (when connecting to HiRDB)
- Oracle (when connecting to Oracle)
- SQL Server (when connecting to SQL Server)
- XDM/RD E2 (when connecting to XDM/RD E2)
- TP1/Message Queue (when connecting to TP1/Message Queue)
- TP1/Server Base (when connecting to SPP of OpenTP1)

Invoke the required processes in the respective resource managers.

(c) Management Server machine

You need to install Application Server on the Management Server machine.

Invoke the following process:

- Management Server

(d) Web client machine

A Web browser is required on the Web client machine.

3.6.2 Configuration when using a global transaction

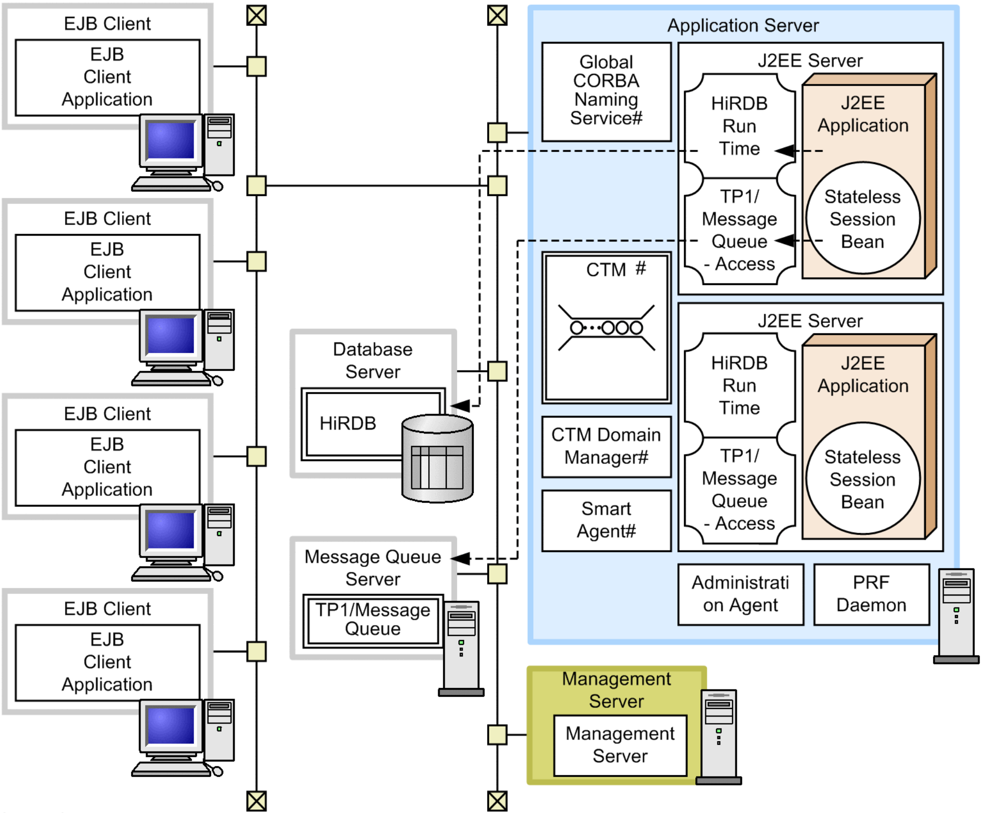
This subsection describes the configuration, when multiple resources participate in a transaction.

(1) Features of the system configuration

In this configuration, multiple resource managers are accessed from a single J2EE application consisting of Session Beans, through the resource adapter. The J2EE server manages the transaction that is used for accessing the resource manager from the application. The transaction type is a global transaction that uses an XA interface.

The following figure shows an example configuration when you use a global transaction. The following figure shows an example of the configuration of a system using CTM:

Figure 3–35: Example configuration when using a global transaction



Legend:

- - -►: Flow for accessing the resource manager from application through resource adapter

# Required when using CTM.

Note: For other legend items, see *3.2 Description of the system configuration*.

Features

- In this configuration, multiple resource managers will be accessed from the application. A two-phase commit is required.
- You need to implement this configuration even in a system where the applications that access the single resource manager and applications that access multiple resource managers are mixed.

Flow of accessing the resource manager from the application through the resource adapter

A Stateless Session Bean that is accessed from the EJB client application accesses the resource manager through the resource adapter.

(2) Required software and the processes to be invoked on the respective machines

This subsection describes the required software and the processes to be invoked on the respective machines when you use a global transaction:

(a) Application server machine

You need to install Application Server on the Application Server machine.

The following software is required for connecting to the resource manager:

Resource manager to be connected to	Required software
HiRDB	HiRDB Run Time or HiRDB Type4 JDBC Driver
Oracle	Oracle Client or Oracle JDBC Thin Driver
TP1/Message Queue	TP1/Message Queue - Access
SPP of OpenTP1	uCosminexus TP1 Connector TP1/Client/J

Note: You cannot use SQL Server and XDM/RD E2 in a global transaction.

Invoke the following processes:

- J2EE server
- Administration Agent
- PRF daemon

In a configuration where CTM is used, you must install Application Server and start the global CORBA Naming Service, CTM processes, CTM domain manager, and Smart Agent apart from the above processes. For details, see *3.5.2 Integration between the servers that invokes Stateless Session Bean through CTM*.

(b) Machine on which the resource manager is running

You install any of the following software on the machine on which the resource manager is running. You cannot use SQL Server and XDM/RD E2 in a global transaction.

- HiRDB (when connecting to HiRDB)
- Oracle (when connecting to Oracle)
- TP1/Message Queue (when connecting to TP1/Message Queue)
- TP1/Server Base (when connecting to SPP of OpenTP1)

Invoke the processes required on the respective resource managers.

(c) Management Server machine

You need to install Application Server on the Management Server machine.

Invoke the following process:

- Management Server

(d) EJB client machine

You need to install Application Server or the uCosminexus Client (in the case of Windows) on the EJB client machine.

Invoke the processes of the EJB client application.

3.6.3 Configuration when using transaction context propagation

This subsection describes the configuration where the respective J2EE servers connect to the different resources in the case of an integrated server configuration. Use transaction context propagation in this configuration. Note that CTM cannot be used for this configuration.

(1) Features of the system configuration

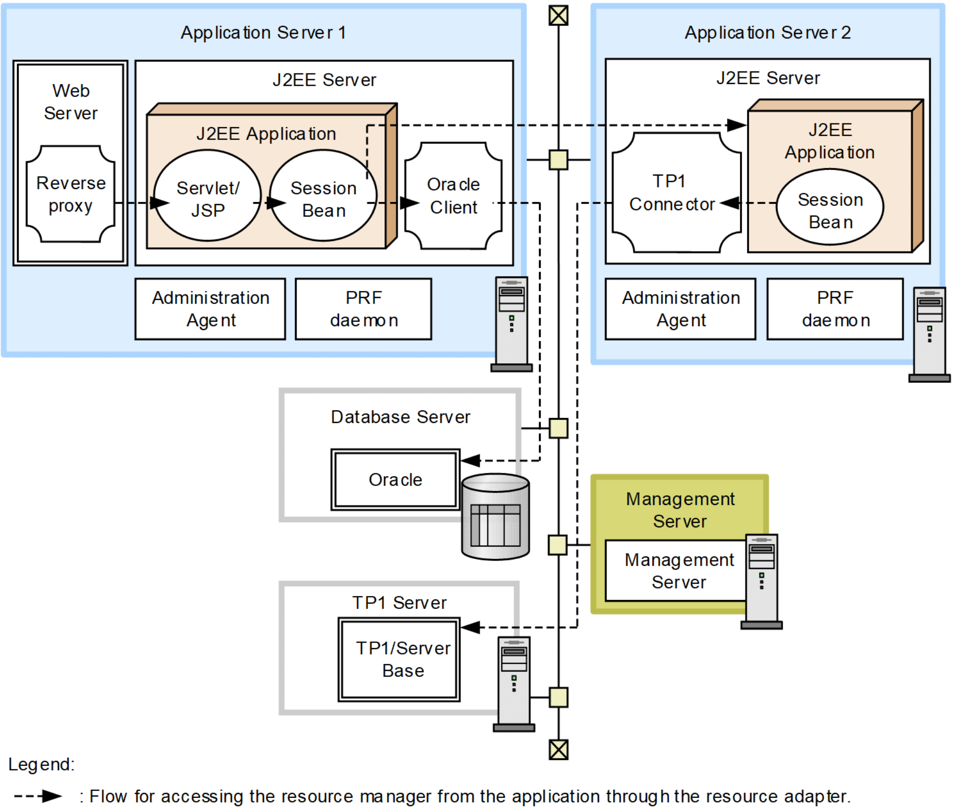
In this configuration, integration between servers is performed with multiple J2EE applications. This subsection describes an example of integrating servers using a J2EE application that consists of servlets, JSPs, and Session Beans and a J2EE application that consists of only Session Beans. The application that consists of servlets, JSPs, and Session Beans runs on the client-side application server and the J2EE application that consists of Session Beans runs on the server-side application server.

The respective J2EE applications access different resource managers through the resource adapter. In such a case, the J2EE server manages the transactions used when accessing the resource managers. The transaction type is global transaction that uses an XA interface.

In configurations where Stateless Session Bean is invoked by CTM, transaction context propagation cannot be used.

The following figure shows an example of the configuration when you use transaction context propagation. In this example, the J2EE applications access the database and SPP of OpenTP1.

Figure 3–36: Example configuration when using transaction context propagation



Note: For other legend items, see *3.2 Description of the system configuration*.

Features

- This configuration is applicable when accessing multiple resource managers from applications where server integration is performed.
- You need to implement this configuration when linking servers in a system where applications that access a single resource manager and applications that access multiple resource managers are coexist.

Flow of accessing the resource manager from the application through a resource adapter

The servlets and JSPs that are accessed from the Web browser through the Web server, locally invoke a Session Bean existing in the same application. This Session Bean starts the transaction, accesses the database, and then invokes a

Session Bean running on the J2EE server that runs on the server-side application server. The invoked Session Bean that runs on the server-side application server accesses SPP of OpenTP1. A Session Bean running on the client-side application server commits the transaction when the control returns from the server-side Session Bean.

(2) Required software and the processes to be invoked on the respective machines

The required software and the processes invoked when you use transaction context propagation are same as *3.6.2 Configuration when using a global transaction*. However, if you are using transaction propagation of context, CTM cannot be used.

3.7 Determining the load balancing method by the load-balancing cluster

This section describes the configuration used for load balancing by the load-balancing cluster.

Load balancing is implemented as explained below. The method that can be implemented depends on the type of access point component.

- Use a load balancer (in the case of servlet or JSP)
- Use a Naming Service (in the case of Session Bean or Entity Bean)
- Use CTM (for Stateless Session Bean)

There are two configurations in which the load balancer is used. One is a system configuration that uses Web server integration, and the other a system configuration that accesses the NIO HTTP server of the J2EE server directly without going through a Web server. Each of these configurations is described separately. For details about load balancing when the access point is a Message-driven Bean, see *3.8.4 Load balancing using the Message-driven Bean instance pool (when using TP1/Message Queue)*.

3.7.1 Load balancing with a load balancer in case of Web server integration (in the case of servlet or JSP)

This subsection describes the configuration where the load balancer (*layer 5 switch*) balances the load. You can use this method when the servlets or JSPs are the access points.

This subsection describes Web server integration.

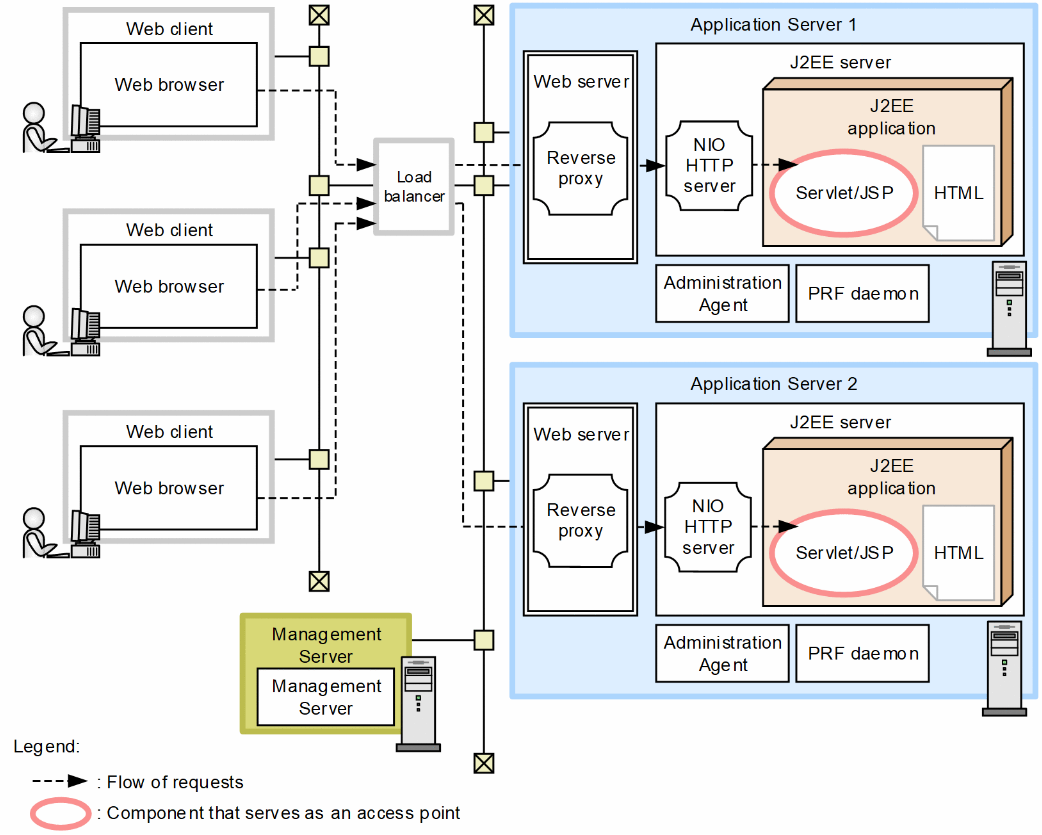
(1) Features of the system configuration

You can use this configuration when the access point of the application running on the J2EE server is either a servlet or JSP.

You can implement load balancing by registering the target Application Server in a load balancer. You can distribute the client access load by registering multiple Application Server instances.

The following figure shows an example configuration of load balancing using a load balancer:

Figure 3–37: Example configuration of load balancing using a load balancer (in the case of Web server integration)



Note: For other legend items, see *3.2 Description of the system configuration*.

Features

- Scalability and availability of servlets and JSPs can be ensured.
- When a problem occurs in a specific Application Server or when maintenance is required, system fall back is possible by stopping the access from the load balancer to the corresponding Application Server.

Flow of requests

Requests are sent from the Web browser to the servlets and JSPs forming the access points on the J2EE server through the load balancer and the Web server. At this point, the load balancer distributes the access from the Web browser to the Web servers running on the respective Application Server machines. The load balancer also controls the relationship between the Sticky and Affinity HTTP sessions.

Reference note

- When using HTTPS, you need to deploy an SSL accelerator in front of the load balancer when using the contents (such as a header) of the request to sort out the requests in the load balancer.
- When using the Smart Composer functionality, you can also deploy the load balancer in a redundant configuration.

(2) Required software and the processes to be invoked in the respective machines

The required software and the processes to be invoked in the respective machines when you use a load balancer are same as in the configuration with servlets and JSPs as the access points. See *3.4.1 Configuration with servlets and JSPs as access points (for Web server integration)*.

3.7.2 Load balancing with a load balancer when using the NIO HTTP server (when not going through a Web server)

This subsection describes the configuration where load balancing is executed by load balancer (*layer 5 switch*). This method is used when servlets or JSPs are the access points.

The following explains a system that accesses the NIO HTTP server of the J2EE server directly without going through a Web server.

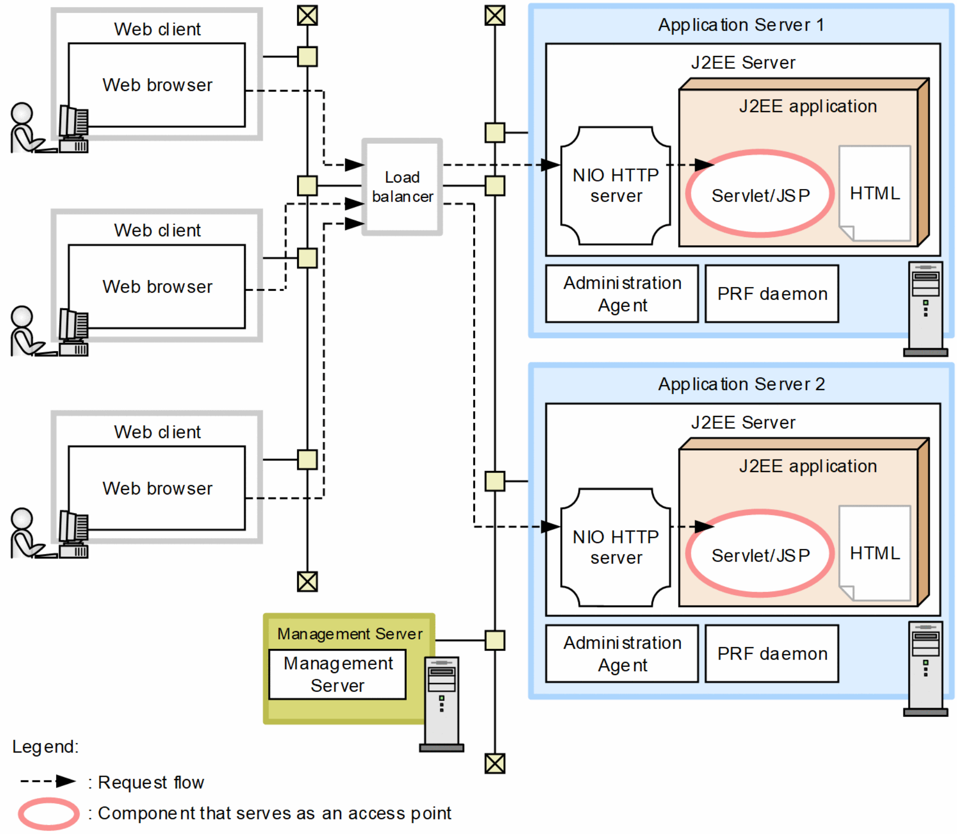
(1) Features of system configuration

This configuration can be used when servlets or JSPs are the access points of the applications running on J2EE servers.

The load balancing can be implemented by registering the Application Server instance that is the target of the load balancer. You can register multiple Application Server instances to reduce the distribute the load created by access from the clients.

The following figure shows the configuration of load balancing using the load balancer:

Figure 3–38: Example of the configuration of load balancing using the load balancer (when accessing the NIO HTTP server directly)



Note: For other legend items, see *3.2 Description of the system configuration*.

Features

- The scalability and the availability of servlets and JSPs can be secured.
- If an error occurs on a particular Application Server, or if maintenance is required, you can stop accessing the relevant Application Server with the load balancer and can execute the degeneration operation of the system.
- As J2EE servers can be directly accessed from a Web browser without using Web servers, the performance is high in this configuration. Moreover, the operation is simple because you need not start Web servers. However, you must take care of the functions and the configuration that you can use. When connecting to the Internet, set up the deployment of the web server embedded with the reverse proxy at the front-end. For details, see *3.4.2 Configuration where servlets and JSPs are used as access points (when accessing the NIO HTTP server directly)*.

Request flow

The request is sent from the Web browser to the access point servlets and JSPs on the J2EE server through the load balancer. In such cases, the load balancer distributes the access from the Web browser to J2EE servers running on the respective Application Server machines. Note that the load balancer also controls the correlation of Sticky and Afinity of HTTP.

Reference note

When you use HTTPS, you must set up the deployment of the SSL accelerator at the front-end of load balancer.

(2) Required software and process to be started on each machine

If you are using load balancer, the required software and processes to be started on the respective machines is same as in configurations where servlets and JSPs are the access points. For details, see *3.4.2 Configuration where servlets and JSPs are used as access points (when accessing the NIO HTTP server directly)*.

3.7.3 Load balancing with the CORBA Naming Service (in the case of Session Beans and Entity Beans)

This subsection describes the configuration where load is balanced by using the round robin search functionality of the CORBA Naming Service present on the J2EE server (in-process), when the access point component is either a Session Bean or an Entity Bean.

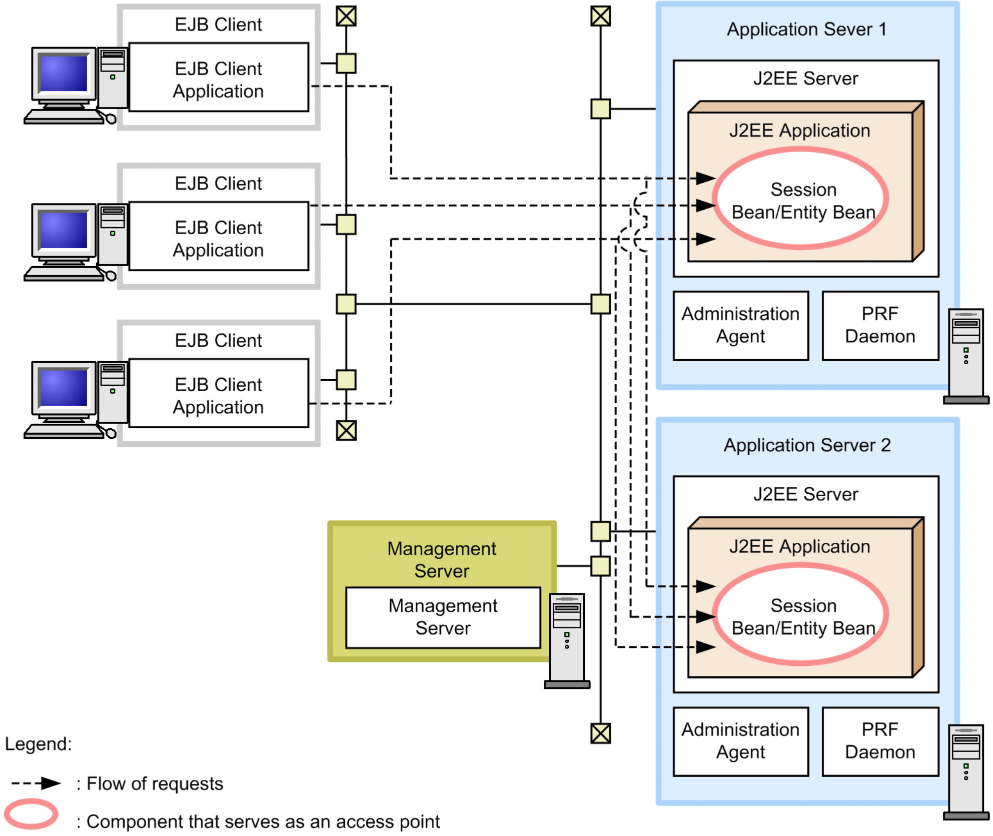
(1) Features of the system configuration

You can use this configuration when the access point of the application running on the J2EE server is either a Session Bean or an Entity Bean. The EJB client application is the client. The EJB client application distributes the request distribution destinations by looking up the object reference in the round robin format from the logical Naming Service registered in the system properties. The Session Bean and Entity Bean, however, need to be started with the same name (same optional name) in the respective J2EE servers.

The EJB client application accesses the same J2EE server from the time the Naming Service of the J2EE server is looked up in the round robin format until the Naming Service is looked up again.

The following figure shows an example configuration of load balancing with a Session Bean and an Entity Bean as the access points:

Figure 3–39: Example configuration of load balancing with a Session Bean and an Entity Bean as the access points



Note: For other legend items, see *3.2 Description of the system configuration*.

Features

- Scalability and availability of a Session Bean and an Entity Bean can be ensured.
- You can prepare multiple Application Server instances and then select and access the CORBA Naming Service in the round robin format from the EJB client, and thus distribute the load.
- When an error occurs in a specific Application Server or when maintenance is required, the corresponding Application Server is not accessed after the EJB client application detects the termination of the J2EE server. As a result, reduced system operation is possible.

Flow of requests

Requests are sent from the EJB client application to a Session Bean and an Entity Bean forming the access points on the J2EE server. At this point, the EJB client application selects the Naming Service of the J2EE server in the round robin format and looks up the object reference.

(2) Required software and the processes to be invoked on the respective machines

The required software and the processes to be invoked on the respective machines during load balancing with the CORBA Naming Service are the same as in the configuration that uses Session Beans and Entity Beans as the access points. See *3.4.3 Configuration with Session Beans and Entity Beans as access points*.

3.7.4 Load balancing when using CTM (for Stateless Session Bean)

This is a configuration for load balancing using CTM when the access point component is Stateless Session Bean.

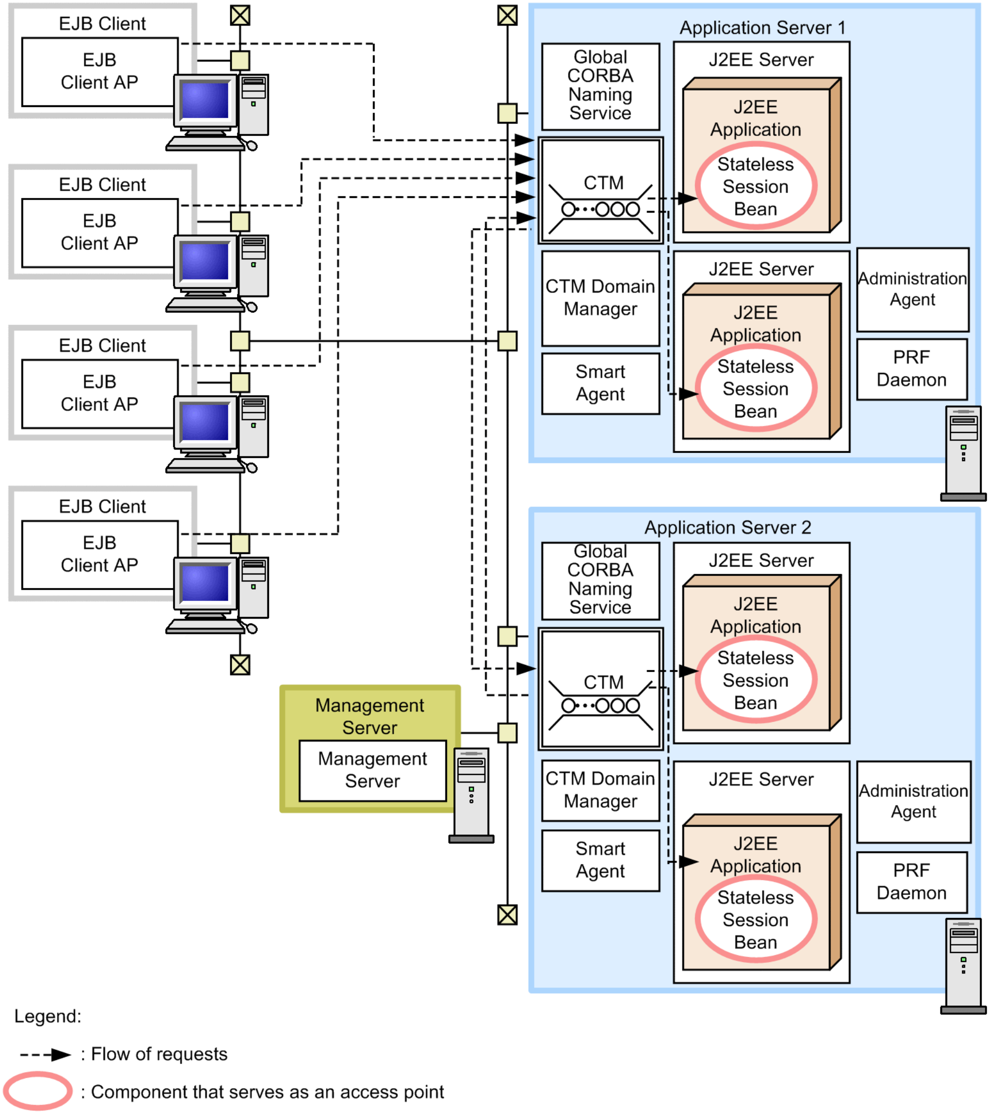
(1) Features of the system configuration

This configuration is implemented using CTM when the access point of the application running on the J2EE server is Stateless Session Bean. This subsection describes the cases when clients are the EJB client applications.

An EJB client application looks up the object reference in round robin format from global the CORBA Naming Service registered in the system properties, and deploys the distribution destination of requests. However, you must start the Stateless Session Bean with the same name (same alias) on each J2EE server.

The following figure shows an example of the configuration of load balancing using CTM for Stateless Session Beans:

Figure 3–40: Example of configuration of load balancing by CTM for Stateless Session Beans



Note: For other legend items, see *3.2 Description of the system configuration*.

Features

- High availability can be secured by implementing load balancing of Stateless Session Beans.

- Load balancing between the J2EE servers can be implemented by looking up the global CORBA Naming Service in round robin format from an EJB client and by distributing the requests by CTM.
- Stateless Session Beans can be easily scaled out, and therefore, a higher operation usage rate of Application Server can be achieved.
- If an error occurs on a particular J2EE server, degeneration operation of the system is possible by scheduling the requests on the other J2EE server by CTM. The J2EE server, where the error is detected, cannot be accessed from the EJB client application.

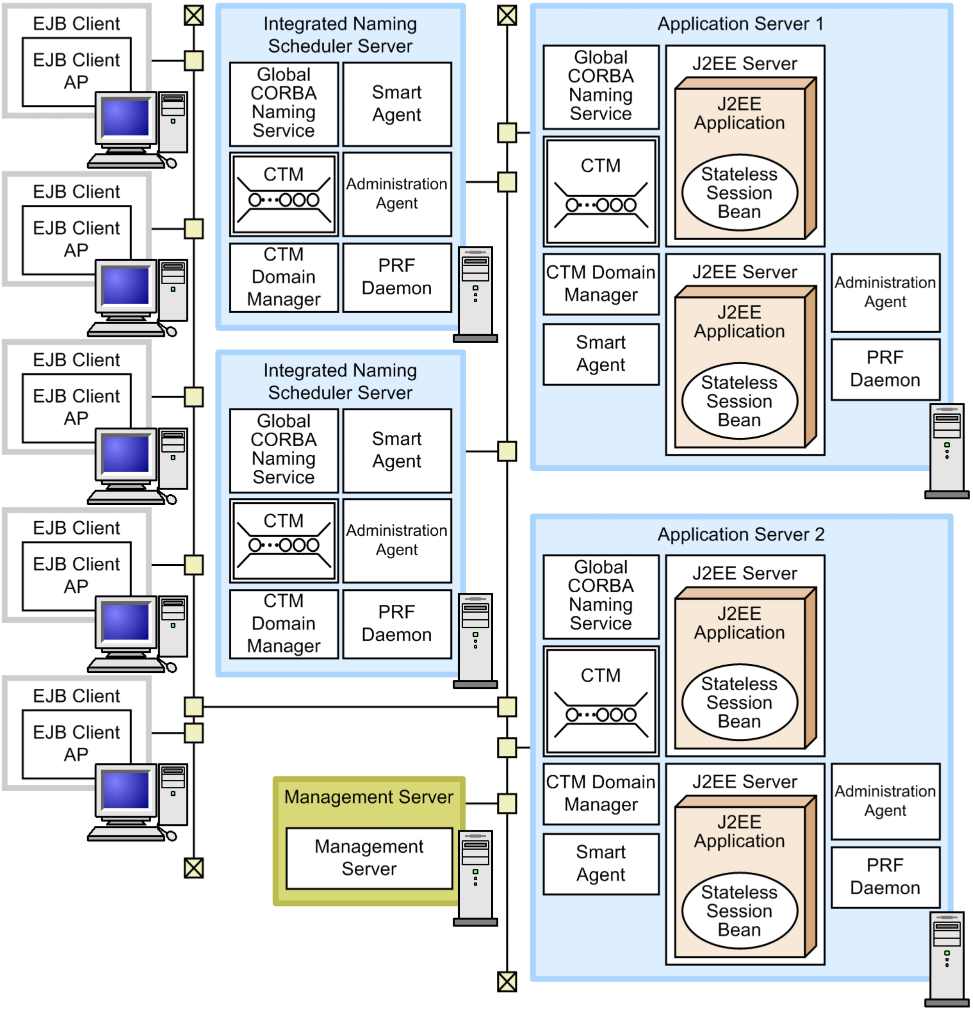
Requests flow

The request is sent through CTM from the EJB client to the Stateless Session Bean on the J2EE server, which is the access point. In such cases, the EJB client looks up the name of the EJBHome object reference of the Stateless Session Bean from the global CORBA Naming Service. After that, the request is distributed to the J2EE server on the appropriate Application Server machine by CTM.

When balancing the load using CTM, the global CORBA Naming Service can be deployed on an independent machine. The machine where the global CORBA Naming Service is deployed is called the *integrated naming scheduler server*.

The following figure shows the configuration of a system where an integrated naming scheduler server is deployed:

Figure 3–41: Example of configuration of load balancing by CTM for Stateless Session Bean (when deploying integrated naming scheduler server)



EJB Client AP: EJB Client Application

Note: For other legend items, see *3.2 Description of the system configuration*.

Features

- High availability can be secured by implementing the load balancing of Stateless Session Beans.
- The availability of the Naming Service can be secured by creating a replica of the integrated naming scheduler server and multiple deployments.
- Load balancing between J2EE servers is implemented by looking up the global CORBA Naming Service in round robin format from the EJB client and by distributing the requests using CTM.
- Since Stateless Session Beans can be easily scaled out, the operation rate of the application also improves. It is not required to change the list of the global CORBA Naming Service defined by EJB client during scale out.
- If an error occurs on a particular J2EE server, degeneration operation of the system is possible by scheduling the requests on the other J2EE server using CTM. The J2EE server, where the error is detected, cannot be accessed from the EJB client application.
- The EJB client looks up the name of the EJBHome object reference of the Stateless Session Bean from the global CORBA Naming Service on the integrated naming scheduler server. After that, the process is distributed to the CTM daemon of Application Server.

(2) Required software and processes to be started on each machine

This subsection describes the required software and processes to be started on the respective machines, for load balancing using CTM.

(a) Application server machine

You must install Application Server on the Application Server machine.

The processes to be started are as follows:

- J2EE server
- Administration Agent
- PRF daemon
- Global CORBA Naming Service
- CTM processes (CTM daemon and CTM regulator)
- CTM domain manager
- Smart Agent

(b) Integrated naming scheduler server machine

For the system configuration where integrated naming scheduler server is deployed, you must install Application Server on the integrated naming scheduler server machine.

The processes to be started are as follows. It is not required to start the J2EE server.

- Global CORBA Naming Service
- CTM processes (CTM daemon)
- CTM domain manager
- Smart Agent
- Administration Agent
- PRF daemon

(c) Management Server machine

You must install Application Server on the Management Server machine.

The process to be started is as follows:

- Management Server

(d) EJB client machine

You must install Application Server or uCosminexus Client (In Windows) on the EJB client machine.

The process of EJB client application is to be started.

3.8 Determining the configuration for asynchronous communication between servers

This section describes the system configuration when there is an asynchronous communication between servers using a Message-driven Bean. The Message-driven Bean is the access point.

System configurations where a Message-driven Bean is used for asynchronous communication include the system configurations using Cosminexus JMS provider, system configurations using TP1/Message Queue, and system configurations using Cosminexus RM. JMS provider and Cosminexus RM are component software of Application Server. In a system using Cosminexus RM, you cannot distribute loads using the instance pool of the Message-driven Bean.

3.8.1 Configuration in which a Message-driven Bean is used as the access point (when using Cosminexus JMS provider)

This subsection describes the configuration where a Message-driven Bean of the J2EE server is invoked using Cosminexus JMS provider.

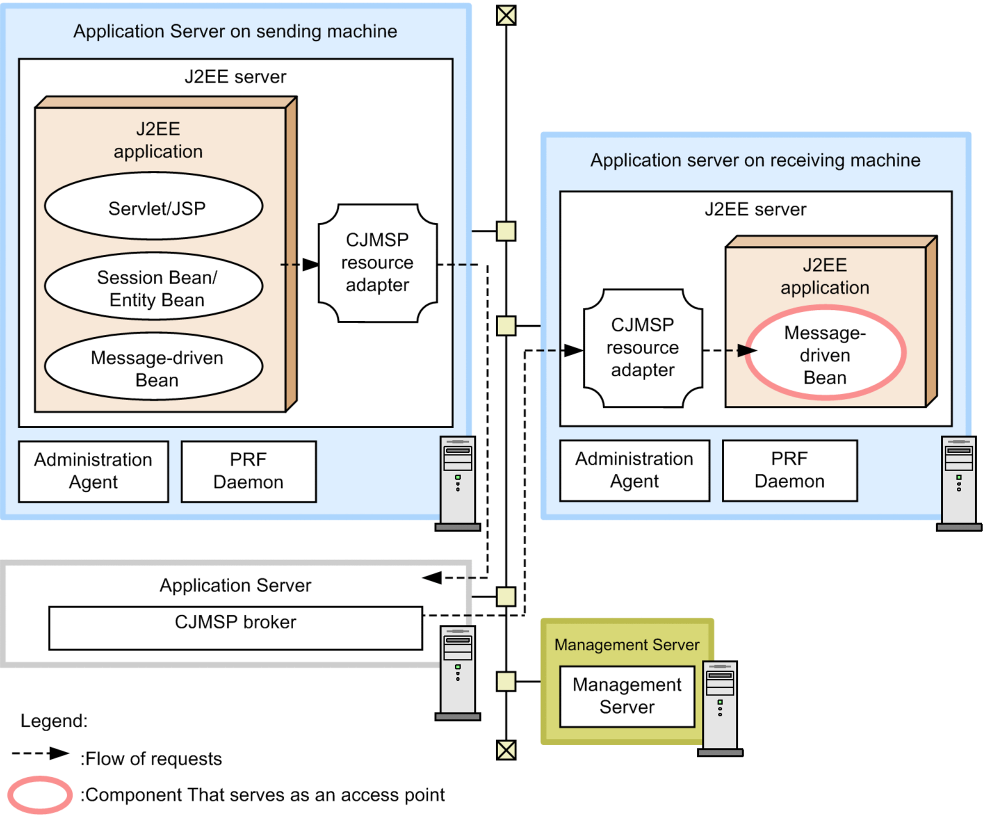
In this configuration, Application Server on receiving machine is invoked via CJMSP Broker from Application Server on the sending machine that sends the message. An application configured with servlets, JSPs, Entity Beans, Session Beans, or Message-driven Beans operates in Application Server on the sending machine. The component that is the access point on the receiving machine is a Message-driven Bean.

(1) Features of the system configuration

This configuration is one of the most basic message-driven systems.

The following figure shows an example of the message-driven system configuration for using Cosminexus JMS provider:

Figure 3–42: Example of the message driven system configuration (for using Cosminexus JMS provider)



Note: For other legend items, see *3.2 Description of the system configuration*.

Features

In Application Server on the sending machine, J2EE client applications that use a JMS interface and CJMSP resource adapter are used for sending messages.

Request flow

The Message-driven Bean, which is the access point, operates on the J2EE server of Application Server on the receiving machine. The CJMSP resource adapter library in the resource adapter operates on the J2EE server of Application Server on the sending machine and the receiving machine.

The request message from the J2EE application of Application Server on the sending machine is sent through CJMSP Broker, and the Message-driven Bean on Application Server on the receiving machine is invoked.

Note that you can allocate Application Server for the sending machine, Application Server for the receiving machine, and CJMSP Broker on the same machine.

(2) Required software and processes to be started on each machine

This subsection describes the required software and processes to be started on the respective machines.

(a) Application Server machine (Application Server machine on the sending machine)

You must install Application Server on the Application Server machine (Application Server machine on the server machine).

The following are the processes to be started:

- J2EE server

- Administration Agent
- PRF daemon

(b) Application Server machine (Machine for allocating CJMSP Broker)

You must install Application Server on the machine where CJMSP Broker is allocated.

CJMSP Broker is the process to be started. Furthermore, Management Server does not manage the CJMSP Broker.

(c) Application Server machine (Application Server machine on the receiving machine)

You must install Application Server on the client machine (Application Server machine on the client machine).

The processes to be started are as follows:

- J2EE server
- Administration Agent
- PRF daemon

(d) Management Server machine

You must install Application Server on the Management Server machine.

The process to be started is as follows:

- Management Server

3.8.2 Configuration in which a Message-driven Bean is used as the access point (when using TP1/Message Queue)

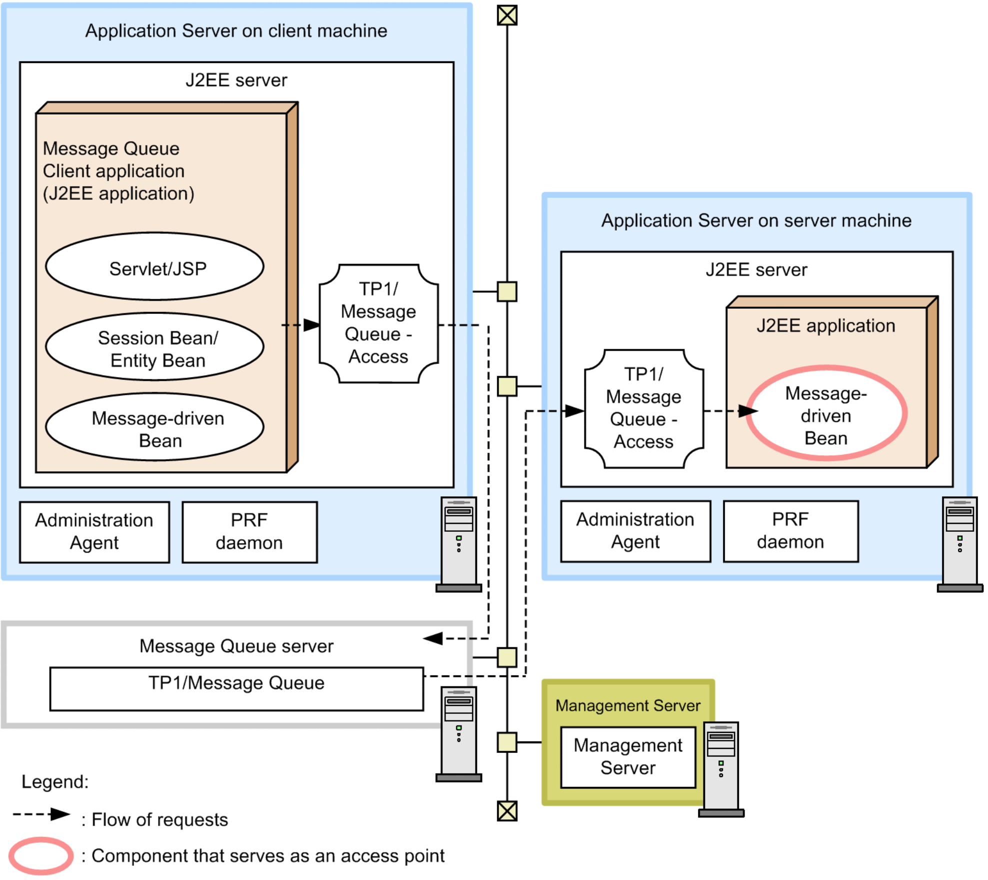
This subsection describes configurations in which a Message Queue server operated by TP1/Message Queue is used as the access point.

(1) Features of the system configuration

This configuration is one of the most basic message-driven systems.

The following figure shows an example of the message-driven system configuration for using TP1/Message Queue:

Figure 3–43: Example of the message driven system configuration (for using TP1/Message Queue)


Application Server on client machine
J2EE server
Message Queue Client application (J2EE application)
Servlet/JSP
Session Bean/ Entity Bean
Message-driven Bean
TP1/ Message Queue - Access
Administration Agent
PRF daemon
Application Server on server machine
J2EE server
J2EE application
TP1/ Message Queue - Access
Message-driven Bean
Administration Agent
PRF daemon
Message Queue server
TP1/Message Queue
Management Server
Management Server
Legend:
: Flow of requests
: Component that serves as an access point

Note: For other legend items, see *3.2 Description of the system configuration*.

Features

A Message Queue client application and TP1/Message Queue that send TP1/Message Queue messages are used at the client. Note that the Message Queue client application is a J2EE application that uses a JMS interface.

Request flow

The Message-driven Bean, which is the access point, operates on the J2EE server of the Application Server at the server side. TP1/Message Queue - Access library, which exists in the resource adapter, operates on the J2EE server of the Application Server at the server side, and on the J2EE server of the Application Server at the client side.

The request message (message) is sent from the J2EE application of the Application Server at the client side through TP1/Message Queue, and invokes the Message-driven Bean on the Application Server at the server side.

(2) Required software and processes to be started on each machine

This subsection describes the required software and processes to be started on the respective machines. For details about the processes to connect to resource, see *3.6 Determining the transaction type*.

(a) Application server machine (Application Server machine at server side)

You must install Application Server and TP1/Message Queue - Access on the Application Server machine (Application Server machine at the server side).

The processes to be started are as follows:

- J2EE server

- Administration Agent
- PRF daemon

(b) Message Queue server machine

You must install TP1/Message Queue on the Message Queue server machine.

The process of TP1/Message Queue must be started.

(c) Application server machine (Application Server machine at client side)

You must install Application Server and TP1/Message Queue - Access on the client machine (Application Server machine at the client side).

The processes to be started are as follows:

- J2EE server
- Administration Agent
- PRF daemon

(d) Management Server machine

You must install Application Server on the Management Server machine.

The process to be started is as follows:

- Management Server

3.8.3 Configuration when using a Message-driven Bean as an access point (when using Cosminexus RM)

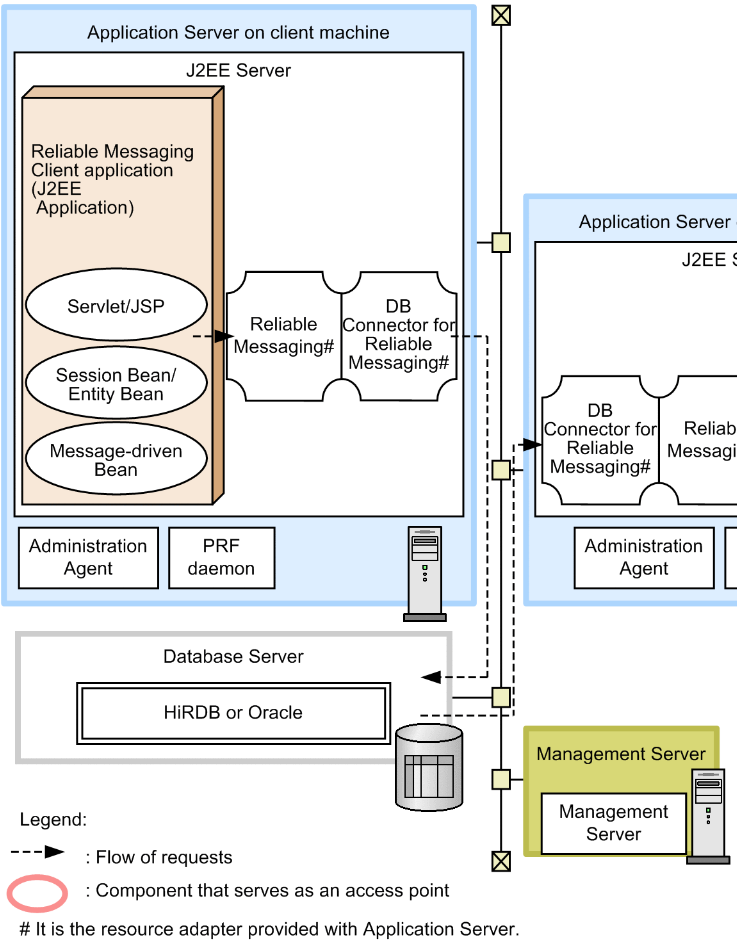
This subsection describes the configuration when using a database (HiRDB or Oracle) that is integrated with Cosminexus RM, as the client.

(1) Features of the system configuration

This is one of the most basic message-driven systems.

The following figure shows an example configuration of a message-driven system when using Cosminexus RM:

Figure 3–44: Example configuration of a message-driven system (when using Cosminexus RM)


Application Server on client machine
J2EE Server
Reliable Messaging Client application (J2EE Application)
Servlet/JSP
Session Bean/ Entity Bean
Message-driven Bean
Reliable Messaging#
DB Connector for Reliable Messaging#
Administration Agent
PRF daemon
Application Server on server machine
J2EE Server
J2EE Application
DB Connector for Reliable Messaging#
Reliable Messaging#
Message-driven Bean
Administration Agent
PRF daemon
Database Server
HiRDB or Oracle
Management Server
Management Server
Legend:
: Flow of requests
: Component that serves as an access point
# It is the resource adapter provided with Application Server.

Note: For other legend items, see *3.2 Description of the system configuration*.

Features

This configuration uses the client application of Cosminexus RM that sends messages to Cosminexus RM and the database server as a client. The Cosminexus RM client application refers to a J2EE application that uses a JMS interface.

Flow of requests

The Message-driven Bean forming the access point runs on the J2EE server of the server-side application server. The library of Cosminexus RM that acts as the resource adapter runs on the J2EE server of the server-side application server as well as the J2EE server of the client-side application server.

The requests (messages) from the J2EE application of the client-side application server are sent through the queues implemented on the database to invoke the Message-driven Bean that runs on the server-side application server.

(2) Required software and the processes to be invoked on the respective machines

This paragraph describes the required software and the processes to be invoked on the respective machines. For details about the processes used for connecting to the resources, see *3.6 Determining the transaction type*.

(a) Server-side application server machine

You need to install Application Server on the server-side application server machine.

Invoke the following processes:

- J2EE server
- Administration Agent
- PRF daemon

(b) Database server machine

You need to install either HiRDB or Oracle on the database server machine.

Invoke the HiRDB or Oracle processes.

(c) Client machine (client-side application server machine)

You need to install Application Server on the client machine (client-side application server machine).

Invoke the following processes:

- J2EE server
- Administration Agent
- PRF daemon

(d) Management Server machine

You need to install Application Server on the Management Server machine.

Invoke the following process:

- Management Server

3.8.4 Load balancing using the Message-driven Bean instance pool (when using TP1/Message Queue)

This subsection describes the load balancing according to the instance pool count of Message-driven Beans for each J2EE server, when the access point component is a Message-driven Bean.

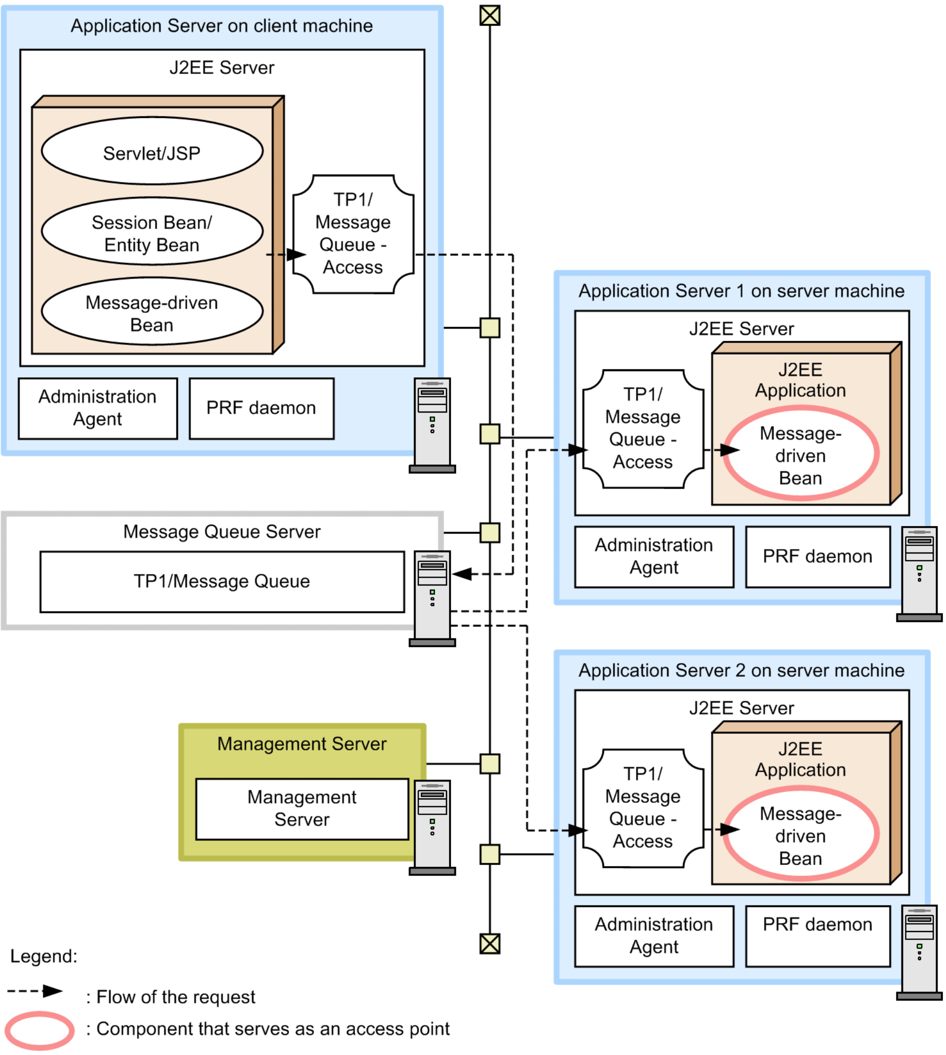
Load balancing with this configuration is possible only when you access Message-driven Beans using TP1/Message Queue. When accessing Message-driven Beans using Cosminexus RM, the load balancing with the instance pool of Message-driven Beans is not possible.

(1) Features of the system configuration

In this configuration, the access point of the application running on the J2EE server is a Message-driven Bean, and this configuration can be used for message-driven systems. When the message from the Message Queue client arrives in TP1/Message Queue on the Message Queue server, the Message-driven Bean is invoked depending on the number of instances of Message-driven Beans pooled on the J2EE server.

The following figure shows an example of the configuration of load balancing for Message-driven Beans:

Figure 3–45: Example of configuration of load balancing for Message-driven Bean



Note: For other legend items, see *3.2 Description of the system configuration*.

Features

- Load balancing of Message-driven Beans leads to high availability.
- Loading of the Application Server can be balanced by preparing multiple Application Server instances.
- If an error occurs in the Application Server at server side or if the maintenance of the Application Server is required, the J2EE server on the Application Server of the relevant TP1/Message Queue cannot be accessed enabling the degeneration operation.

Request flow

The request (message) is sent to the access point Message-driven Bean on the J2EE server of the Application Server at the server side from the J2EE application running on the J2EE server of the Application Server at the client side, through TP1/Message Queue. When a message arrives in TP1/Message Queue, the Message-driven Bean on the J2EE server of the Application Server that exists at the server side is invoked. The requests are distributed according to the instance pool count of Message-driven Beans.

(2) Required software and processes to be started on each machine

The required software and processes to be started when balancing the load using the instance pool of Message-driven Beans is the same as for configurations in which a Message-driven Bean is used as the access point. For details, see *3.8.2 Configuration in which a Message-driven Bean is used as the access point (when using TP1/Message Queue)*.

3.9 Determining the deployment of the operation management process

This section describes the deployment of Management Server, a process used for operation management. *Management Server* is a process used to manage and operate the entire Application Server system configured on various hosts in a batch. You can use Management Server to perform the environment settings of servers on each host and run the servers in a batch. Also, you can understand the state of the entire system using Management Server.

The configurations explained here are as follows:

- Configuration wherein Management Server is deployed on a Management Server machine (Management Server Model)
- Configuration wherein the Management Server is deployed on each machine (Model for managing each host)
- Configuration when operating with commands

Note that the machine on which Management Server is deployed in the Management Server model is called the *management server*.

Reference note

- You use the agent program called *Management Agent* to monitor or collect the statistics of J2EE servers, or batch servers using the Management Server functionality. Include one agent program each in a J2EE server, or batch server on the host to be monitored. Also, include one agent program each in a J2EE server included in a cluster, when using cluster configuration.
- When a LAN where a server for business is deployed and a LAN where a server for management are deployed separately, you can also place the Management Server in the LAN where the server for management is deployed. You must take care of the environment settings when the LANs are separated and when one machine connects to multiple network segments. For details, see, *Appendix D Notes for Setting the Environment for Connecting to the Multiple Network Segments of a Single Machine* in the *uCosminexus Application Server Management Portal User Guide*.

3.9.1 Configuration wherein the Management Server is deployed on each Management Server

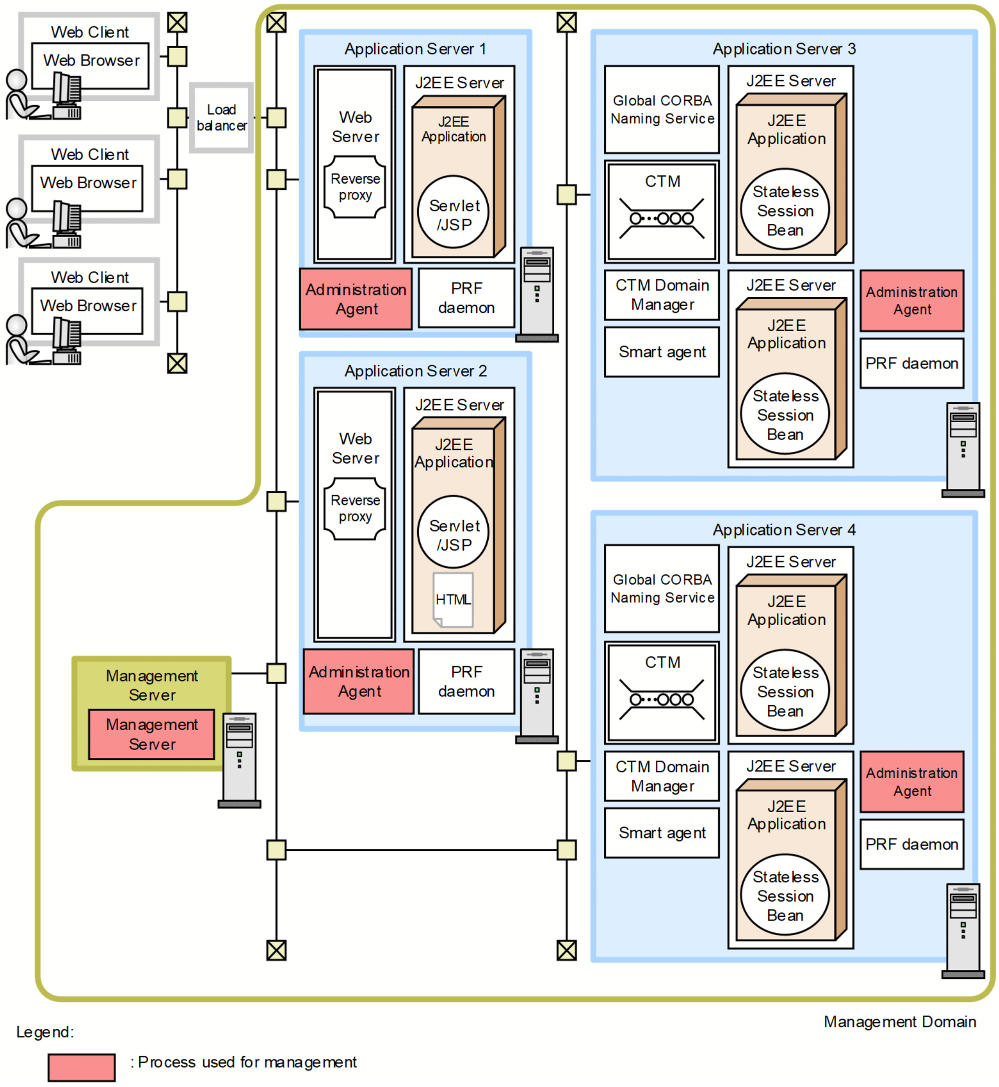
This subsection describes the configuration where you deploy a single Management Server on a domain, when performing operations using Management Server.

(1) Features of the system configuration

In this configuration, the management server machine is deployed for each domain on a system that consists of multiple Application Server instances. You can perform operation management and monitoring of each domain (*Management domain*) that is already defined in Management Server. Management Server on the management server machine executes the management operations based on Administration Agent deployed on each Application Server machine.

The following figure shows an example of a configuration wherein Management Server is deployed on the Management Server machine. In this example, the resource manager to be connected to is omitted.

Figure 3‒46: Example of configuration for deploying the Management Server on a management server



Note: For other legend items, see *3.2 Description of the system configuration*.

Features

- Suitable for batch operation of the systems.
- Hitachi recommends that you prepare the machine for an isolated management server. You can, however, deploy Management Server on a single Application Server machine in the system.

(2) Processes invoked on the respective machines

This subsection describes the required software and processes to be invoked on the respective machines, in the case of a configuration wherein Management Server is deployed on a Management Server machine.

(a) Management Server machine

You need to install Application Server on the management server machine.

Invoke the following process:

- Management Server

Tip

If the management server machine cannot be prepared, you can configure a system where an Application Server instance present in the domain also performs the dual function of Management Server.

In such a case, deploy the processes of Management Server on the Application Server machine that functions as the Management Server machine.

(b) Application server machine

For performing the operation management on each Management Server, invoke the following process on the respective Application Server machine:

- Administration Agent

The required software and the processes to be invoked on the Application Server machine are different for each system configuration, depending on the used functions. Deploy the required software and the processes, depending on the used functions.

3.9.2 Configuration for deploying Management Server on each machine

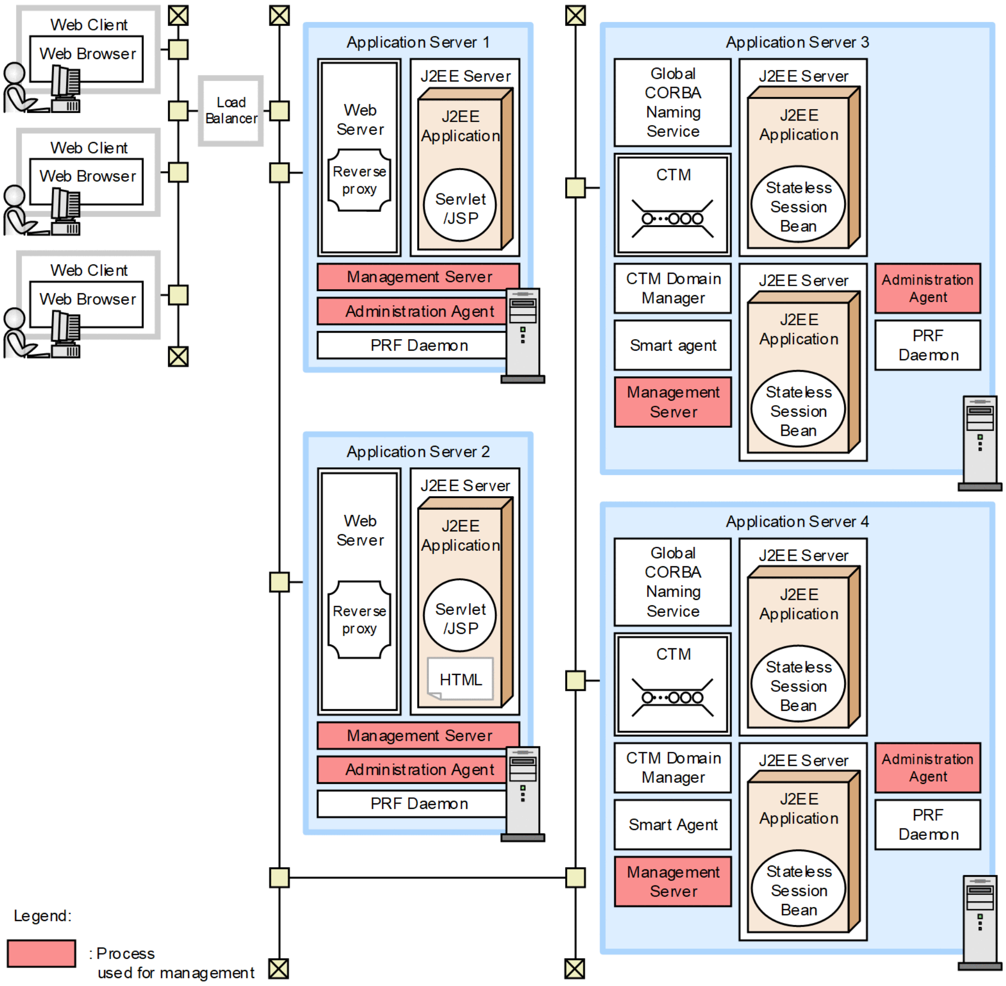
This subsection describes the configuration wherein you deploy Management Server on each Application Server machine, when performing operations using Management Server.

(1) Features of the system configuration

In this configuration, Management Server is deployed on each Application Server machine in a system that has multiple Application Server machines. You can perform operation management and can monitor each Application Server system. The Management Server executes the management operations based on Administration Agent.

The following figure shows an example of the configuration for deploying Management Server on each machine. In this example, the resource managers to be connected are omitted.

Figure 3–47: Example of configuration for deploying the Management Server on each machine



Note: For other legend items, see *3.2 Description of the system configuration*.

Features

- Suitable for operations on each Application Server machine.

(2) Required software and processes to be invoked on the respective machines

This subsection describes the required software and processes to be invoked on the respective machines in the case of a configuration wherein Management Server is deployed on each machine.

(a) Application server machine

For performing the operation management on each machine, invoke the following processes on the respective Application Server machines:

- Management Server
- Administration Agent

The required software and processes to be invoked on the Application Server machine are different for each system configuration, depending on the functions used. You deploy the required software and processes, depending on the used functions.

3.9.3 Configuration when operating with commands

This subsection describes the configuration when operating without using the Management Server.

When Management Server is not used, operate Application Server by editing the definition file and by executing the server management commands. In such a case, you do not require Administration Agent and Management Server on the respective Application Server instances. You consider this configuration, when managing a system using operation support software other than Application Server.

3.10 Determining the inheritance of session information

This section describes the system configuration used for inheriting session information. As a prerequisite for this configuration, the J2EE servers have to be made redundant by the load balancer.

The following configuration inherits session information:

- Configuration using a database (When using the *database session failover functionality*)

3.10.1 Configuration using a database (database session failover functionality)

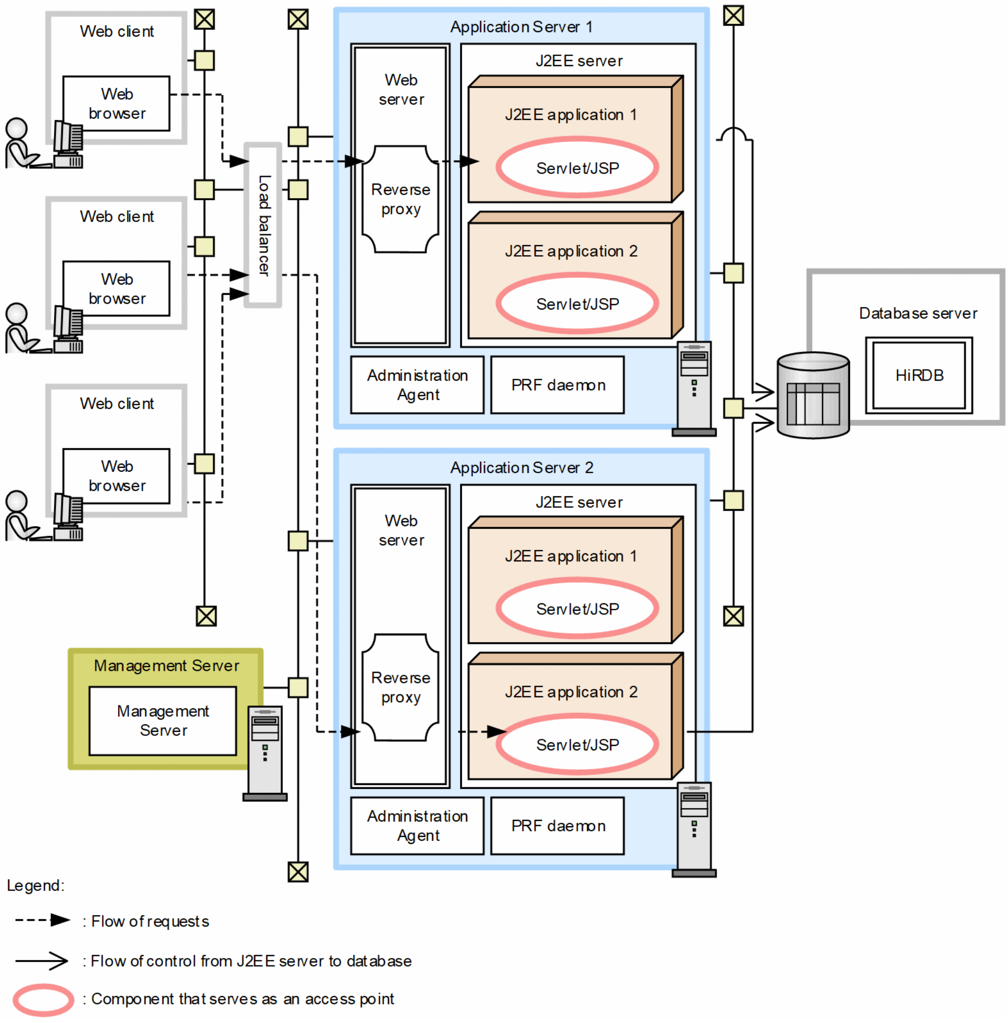
This subsection describes the configuration where session information is inherited using a database.

(1) Features of system configuration

You use a database for saving the session information. If a database is used for saving the business information, you can use the same database for saving the session information.

The following figure shows an example of a configuration where session information is inherited using a database:

Figure 3–48: Example of a configuration where session information is inherited using a database



Note: For other legend items, see *3.2 Description of the system configuration*.

Features

- When an error occurs in a specific J2EE server, the session information is inherited in a separate J2EE server.
- When an error occurs in a database server, the session information saved in the database is left as is. You can use that information after restarting the database server.

Flow of requests

The global session information in the J2EE server is duplicated and managed in the database.

A J2EE server receives the request and when the session is established, the global session information saved in the database is locked. After completing the processing of the J2EE application, the global session information of the database is updated in accordance with the global session information contents of the J2EE server. The lock is then released.

When there is an error in the J2EE server, the global session information saved in the database is acquired from another J2EE server, and the session information is inherited.

(2) Processes invoked on the respective machines

This point describes the software and processes required on the respective machines:

(a) Application Server machine

You must install the software required for establishing a connection with the database when using the database session failover functionality. The following table describes the required software:

Database used	Required software
HiRDB	HiRDB Type4 JDBC Driver
Oracle	Oracle JDBC Thin Driver

Software other than those mentioned in the above table and any other processes to be invoked differ in each system configuration, depending on the functions to be used. Deploy the required software and processes as per the functions to be used.

(b) Database server machine

Install any of the following software on a machine where the database will run:

- HiRDB (When connecting to HiRDB)
- Oracle (When connecting to Oracle)

Start the processes required in respective databases.

(c) Management Server machine

You must install Application Server on the Management Server machine.

The process to be invoked is as follows:

- Management Server

(d) Web client machine

A Web browser is required on the Web client machine.

3.11 Determining node switching when cluster software is used and an error occurs

This subsection describes the system configuration for switching nodes when cluster software is used and an error occurs.

In Application Server systems, the respective OS uses the following cluster software:

- **In Windows**
  Windows Server Failover Cluster
- **In AIX or Linux**
  HA monitor

In systems configured with Application Server, the following configurations can be realized using cluster software. Note that these configurations must be operated using Management Server.

- **Configuration where executing node and standby node of Application Server are in 1-to-1 ratio**
  This configuration consists of one Application Server machine in the standby node for one Application Server machine in the executing node. If an error occurs on the Application Server machine in the executing node and the machine stops or if Administration Agent ends, the cluster software starts the Application Server machine in the standby node, and the processes are switched. Whether a shared disk device is necessary depends on whether you use a transaction service. Note that when you use Windows Server Failover Cluster, you need a shared disk device even if a transaction service is not used.
- **Configuration where executing node and standby node of management server are in a 1-to-1 ratio**
  This configuration consists of one management server machine in the standby node corresponding to one management server machine in the executing node. If an error occurs on the management server machine in the executing node, and the machine stops or if the Management Server process ends, the cluster software starts the management server machine in the standby node, and the processes are switched.
- **Configuration in which the executing node and standby node of Application Server are at mutual standby**
  This is one of the configurations where the executing node and the standby node of Application Server are in 1-to-1 ratio. Deploy a J2EE server of the same type on each Application Server machine, and start a different J2EE server. Each Application Server instance will function as the standby node for each other while operating as the executing node. If an error occurs in any of the nodes, the nodes are switched. As a result, operations with very little wastage are possible on a few Application Server machines.

  With this configuration, Management Server is started on each Application Server machine, and their respective different management domains are controlled.
- **Configuration where one standby node is used for N executing nodes (configuration in which a recovery server is used)**
  If a global transaction is used in a configuration where J2EE servers are redundant, this system configuration is used to conclude transactions when an error occurs on a specific J2EE server.

  Use a load balancer to make N J2EE servers redundant.
- **Configuration where the executing node and standby node of the host unit management model have an n-to-1 ratio**
  This system is used for allocating many executing nodes (1-to-N machines) of Application Server (Host) instances and one machine in the standby node, and allocating the respective Management Server and Administration Agent instances. When there is a failure on an Application Server machine in the executing node, you can continue the business by performing node switching to the Application Server machine in the standby node.

3.11.1 Configuration in which executing node and standby node of Application Server are in 1-to-1 ratio (when transaction service is not used)

This subsection describes the configuration where the executing node and standby node of Application Server are in a 1-to-1 ratio, and the transaction service is not used.

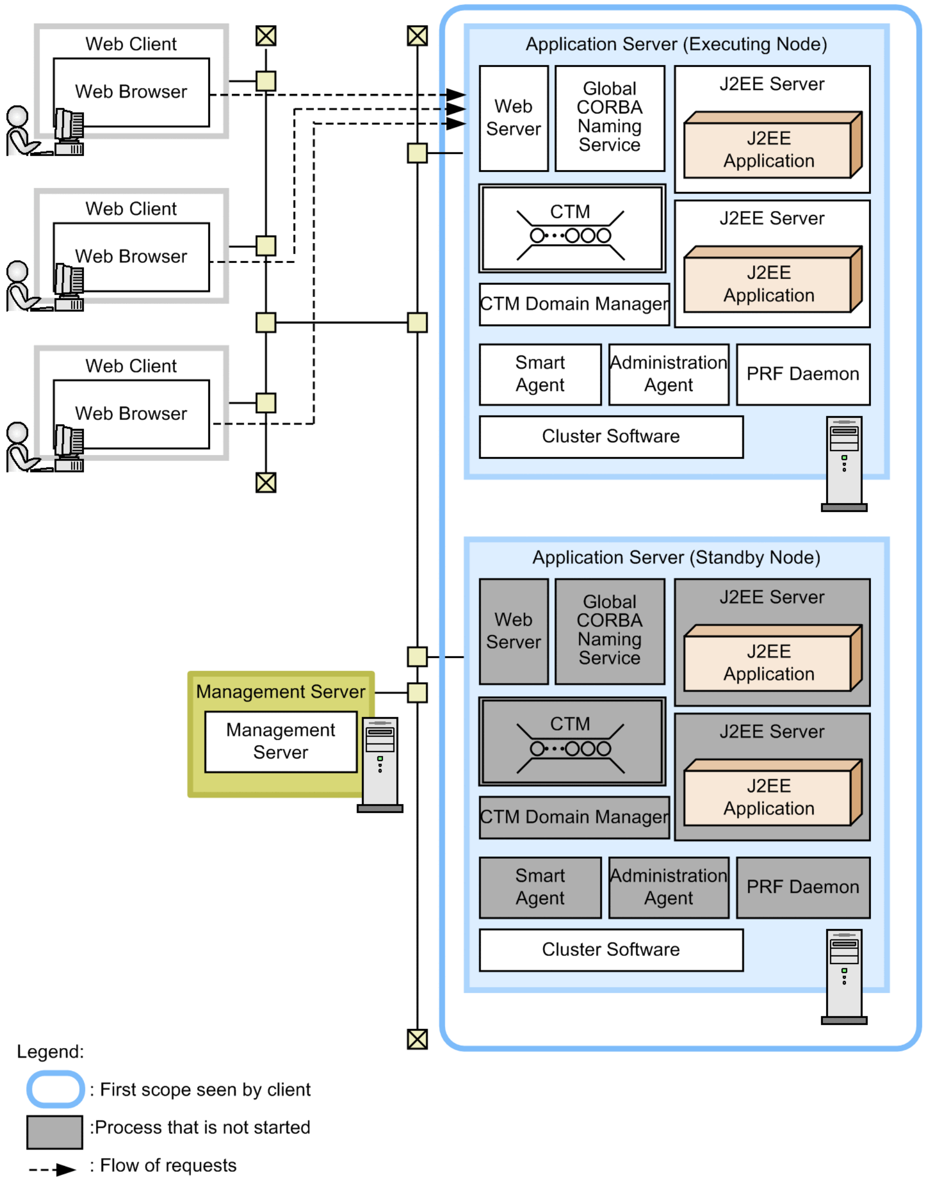
(1) Features of the system configuration

This is a system configuration that does not use transaction service when using local transaction.

This subsection also describes the configuration in which Application Server performs the node switching. For details about the configuration in which the node switching is performed by Management Server, see *3.11.3 Configuration in which executing node and standby node of Management Server are in 1-to-1 ratio*.

The following figure shows an example of configuration where transaction service is not used:

Figure 3–49: Example of system configuration where executing node and standby node are in 1-to-1 ratio and cluster software is used (when transaction service is not used)



Note: For other legend items, see *3.2 Description of the system configuration*.

Features

- The executing node and standby node are configured in a 1-to-1 ratio.
- A shared disk device is not required when you use HA monitor. However, a shared disk device is required when you use Windows Server Failover Cluster. The above figure shows the configuration with HA monitor.
- When the database has a cluster configuration and if Application Server identifies only the virtual address (Logical address), you can still establish the connection.
- For details about allocating Management Server and Application Server on the same machine, see *3.11.6 Configuration in which the executing node and standby node of the host unit management model are in an N-to-1 ratio*.

Flow of requests

The client recognizes one machine for both the Application Server instance in the executing node and instance in the standby node. All the requests are sent to the Application Server in the executing node.

Node switching flow

Node switching is executed when there is a failure in the Application Server machine in executing node. Node switching is executed when the OS or Administration Agent is down. Also, node switching is performed when a command is executed explicitly for node switching, or when there is an event where node switching is to be performed for cluster software.

On executing node switching, Administration Agent of Application Server in the standby node and the processes that were stopped until then are started.

(2) Processes to be invoked on the respective machines

The following subsections describe the software and processes required on the respective machines:

(a) Application Server machine (executing node)

The following are the processes that must be started for using the cluster software:

- Administration Agent

The required software and the processes to be invoked on the Application Server machine are different for each system configuration depending on the used functions. Allocate the required software and processes depending on the used functions.

(b) Application Server machine (standby node)

You must match the configuration of software installed on the Application Server machine in the standby node with the executing node. However, the process will not start unless node switching takes place.

(c) Management Server machine

You must install Application Server on the Management Server machine.

Invoke the following process:

- Management Server

(d) Client machine

Install the following software on client machine:

For Web client configuration

Web browser

For EJB client configuration

uCosminexus Client (in Windows), Application Server

3.11.2 Configuration in which executing node and standby node of Application Server are in 1-to-1 ratio (when using transaction service)

This subsection describes the configuration where the executing node and the standby node of Application Server are in 1-to-1 ratio and the transaction service is used.

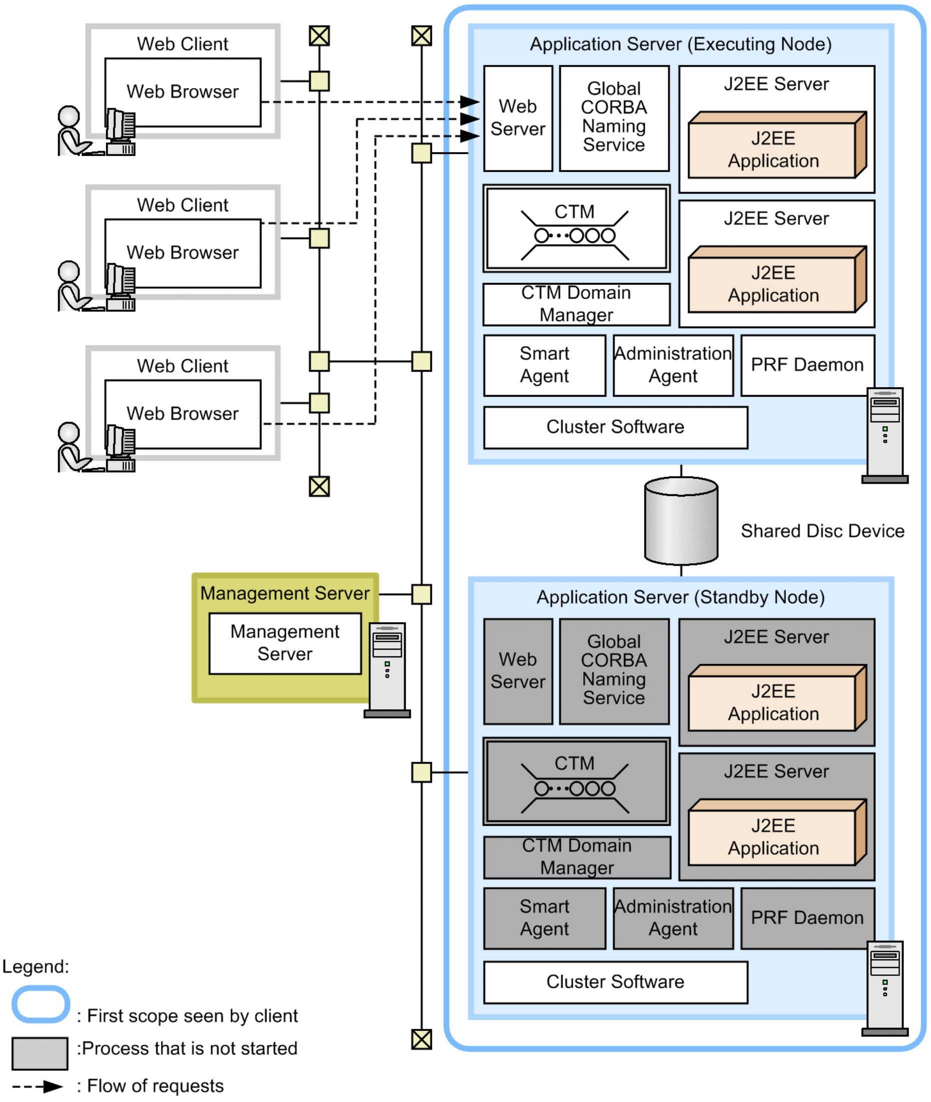
(1) Features of the system configuration

This is a system configuration that uses a transaction service when using a global transaction. A shared disk device is required in such cases.

This subsection also describes the configuration where node switching is performed by Application Server. For details about the configuration where node switching is performed by Management Server, see *3.11.3 Configuration in which executing node and standby node of Management Server are in 1-to-1 ratio*.

The following figure shows an example of a configuration where a transaction service is used:

Figure 3–50: Example of system configuration where executing node and standby node of Application Server are in 1-to-1 ratio and cluster software is used (when transaction service is used)



Note: For other legend items, see *3.2 Description of the system configuration*.

Features

- The executing node and standby node are configured in a 1-to-1 ratio.
- A shared disk device is required.
- When the database has a cluster configuration and if Application Server identifies only the virtual address (Logical address), you can still establish the connection.
- For details about allocating Management Server and Application Server on the same machine, see *3.11.3 Configuration in which executing node and standby node of Management Server are in 1-to-1 ratio*.

Flow of requests

The client recognizes one machine for both the Application Server instance in the executing node and the instance in the standby node. All the requests are sent to Application Server in the executing node.

Node switching flow

Node switching is executed when there is a failure in the Application Server machine in executing node. Node switching is executed when the OS or Administration Agent is down. Also, node switching is performed when a command is executed explicitly for node switching, or when there is an event where node switching is to be performed for cluster software.

On executing node switching, Administration Agent of Application Server in the standby node and the processes that were stopped until then, are started. Also, the information saved to the shared disk is inherited for transaction information.

(2) Processes to be invoked on the respective machines

The software and processes required on respective machines are the same as when the transaction service is not used.

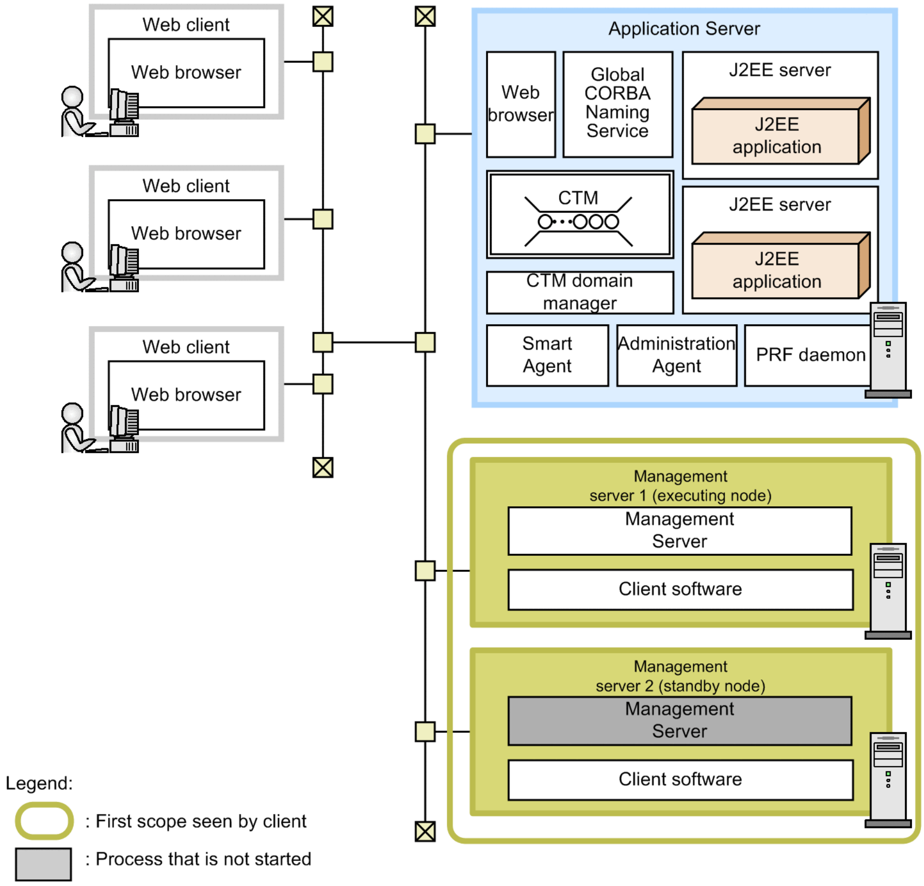
3.11.3 Configuration in which executing node and standby node of Management Server are in 1-to-1 ratio

This subsection describes the node switching of Management Server. The executing node and the standby node of the management server are in a 1-to-1 Ratio.

(1) Features of the system configuration

The following figure shows an example of the configuration in which the executing node and the standby node of Management Server are in 1-to-1 ratio.

Figure 3–51: Example of the system configuration in which the executing node and the standby node of the management server are set to 1-to-1 ratio using cluster software



Note: For other legend items, see *3.2 Description of the system configuration*.

Features

- The executing node and the standby node are configured in a 1-to-1 ratio. Deploy Management Server on the respective machines.
- When you use the Windows Server Failover Cluster, a shared disk device is required.

Node switching flow

Node switching is executed when an error occurs on the Management Server machine of the executing node. Node switching is executed when the OS or Management Server is down. In other cases, node switching is executed when a command for node switching is execute explicitly, and when an event occurs where the node switching of cluster software is required.

When the node switching is executed, the processes of Management Server on the Management Server machine in the standby node are invoked.

(2) Process to be started on the respective machines

This subsection describes about the software required and the process to be started on the respective machines.

(a) Application server machine

The required software and the processes to be started on the Application Server differ in each system configuration according to the used functions. Deploy the required software and processes according to the used functions.

(b) Management server machine (executing node)

You must install Application Server on the Management Server machine.

The process to be started is as follows:

- Management Server

(c) Management server machine (standby node)

Match the configuration of the software installed on the Application Server machine in the standby node with the executing node. However, the process will not start until the node switching occurs.

(d) Client machine

Install the following software on the client machine:

For Web client configuration

Web browser

For EJB client configuration

uCosminexus Client (in Windows), Application Server

3.11.4 Configuration in which executing node and standby node of Application Server are mutually standby

This subsection describes the configuration in which the executing node and the standby node of Application Server are in 1-to-1 ratio and the respective Application Server instances are standby for each other while being operated as the executing node. This configuration is called a *mutual standby configuration*. Note that the system structured with this configuration is called a *mutual node switching system*.

(1) Features of the system configuration

This is one of the configurations in which one Application Server machine in the standby node is prepared for one Application Server machine in the executing node. However, a different J2EE server is operated as the executing node instead of stopping the Application Server in the standby node. This is called *mutual standby.*

When you deploy a J2EE server of the same type on each Application Server machine and start a different J2EE server, the respective Application Server instances operate as an active node and a standby node for each other. If an error occurs on any of the machines, the nodes are switched and processes of both J2EE servers are executed on the other Application Server machine.

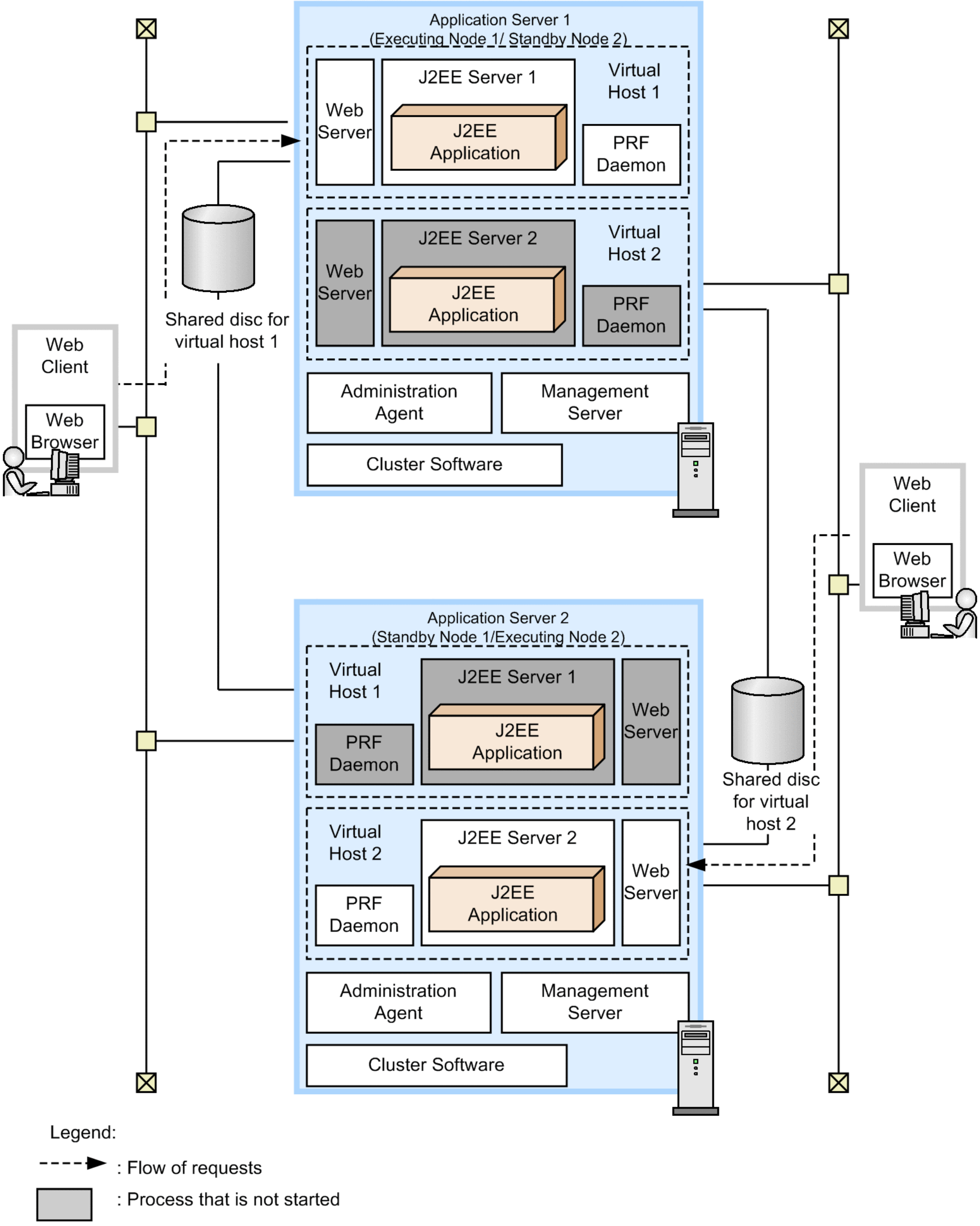
The prerequisites of this configuration are as follows:

- The CORBA Naming Service must be started in the in-process
- When you use a global transaction, you must store the transaction status file on the shared disk
- Operate using Management Server

Management Server is deployed one by one on the same machine as Application Server and is started on both the hosts. The respective Management Server manages different management domains. The scope of the respective management domains is within the Application Server machine where Management Server is deployed.

The following figure shows an example of the mutual node switching system. In this example, the transaction service is used. The shared disk is required when you use the transaction service.

Figure 3–52: Example of the configuration of mutual node switching system (for using transaction service)



Note: For other legend items, see *3.2 Description of the system configuration*.

Features

- The executing node and the standby node of this configuration are in a 1-to-1 ratio. Management Server is deployed on the respective machines.
- Define two virtual hosts in one management domain (one machine). The system is built by considering these virtual hosts as the host of Application Server in the executing node and the host of Application Server in the standby node.
- One Administration Agent controls starting and stopping of the respective hosts in the management domain. However, the IP address allocated to the respective virtual hosts is different from the IP address that is actually used in the operation. Hence, it is defined in such a way that it is appears to operate on a different host.

- The IP addresses for each Application Server operation are dynamically allocated by the cluster software. For sending the request from Management Server to Administration Agent, an IP address that does not move to the other node after node switching is used.
- When you use a global transaction, a shared disk device is required. A shared disk is required for each virtual host.
- The figure shows a LAN divided into virtual host units, but division is not required.

Node switching flow

If an error occurs on any of the Application Server machines, node switching is executed. The node switching is executed if an error occurs in Management Server, Administration Agent, or the Cluster software. Furthermore, node switching is also executed when a command for node switching is executed explicitly, and when an event occurs where the node switching of the cluster software becomes mandatory.

When the node switching is executed, Administration Agent of the Application Server machine in the standby node starts the processes of the logical server that were stopped until now.

For example, as shown in the figure, if an error occurs in Application Server 1, node switching to Application Server 2 is executed by the cluster software, and processes of the J2EE server 1 and other logical servers are started on Application Server by the Administration Agent of Application Server 2.

(2) Processes to be started on each machine

This subsection describes the required software and processes of the respective machines.

(a) Application server machine (executing node1/standby node 2)

You must install Application Server on the Application Server machine.

The processes that must be started on the application server machine in a mutual node switching system are as follows:

- Administration Agent
- Management Server

The other required software and processes of Application Server differ for each system configuration according to the used functions. Deploy the required software and processes as per the used functions.

Start the process of virtual host 1 only for node switching.

(b) Application server machine (standby node 1/executing node 2)

The configuration of the software and processes to be installed on the Application Server machine in the standby node should be executing node1/standby node 2.

For the node switching of processes, start the process of virtual host 2 only.

(c) Client machine

Install the following software on the client machine:

For Web client configuration

Web browse

For EJB client configuration

uCosminexus Client (in Windows), Application Server

3.11.5 Configuration using server exclusive for recovery (N-to-1 recovery system)

This subsection describes the configuration with one machine in the standby node for N machines in the executing nodes (*server exclusive for recovery*). This configuration is called an *N-to-1 recovery system*.

(1) Features of the system configuration

This is a system configuration for resolving the transaction and releasing the resources when there is a problem in certain J2EE server and a global transaction is used in a configuration with redundant J2EE servers. Use load balancer to create redundant J2EE servers for N machines.

The prerequisites of this configuration are as follows:

- Used in a system using global transaction
- The CORBA Naming Service is started as in-process
- The status file of transaction is saved to shared disk
- The operations are performed using Management Server

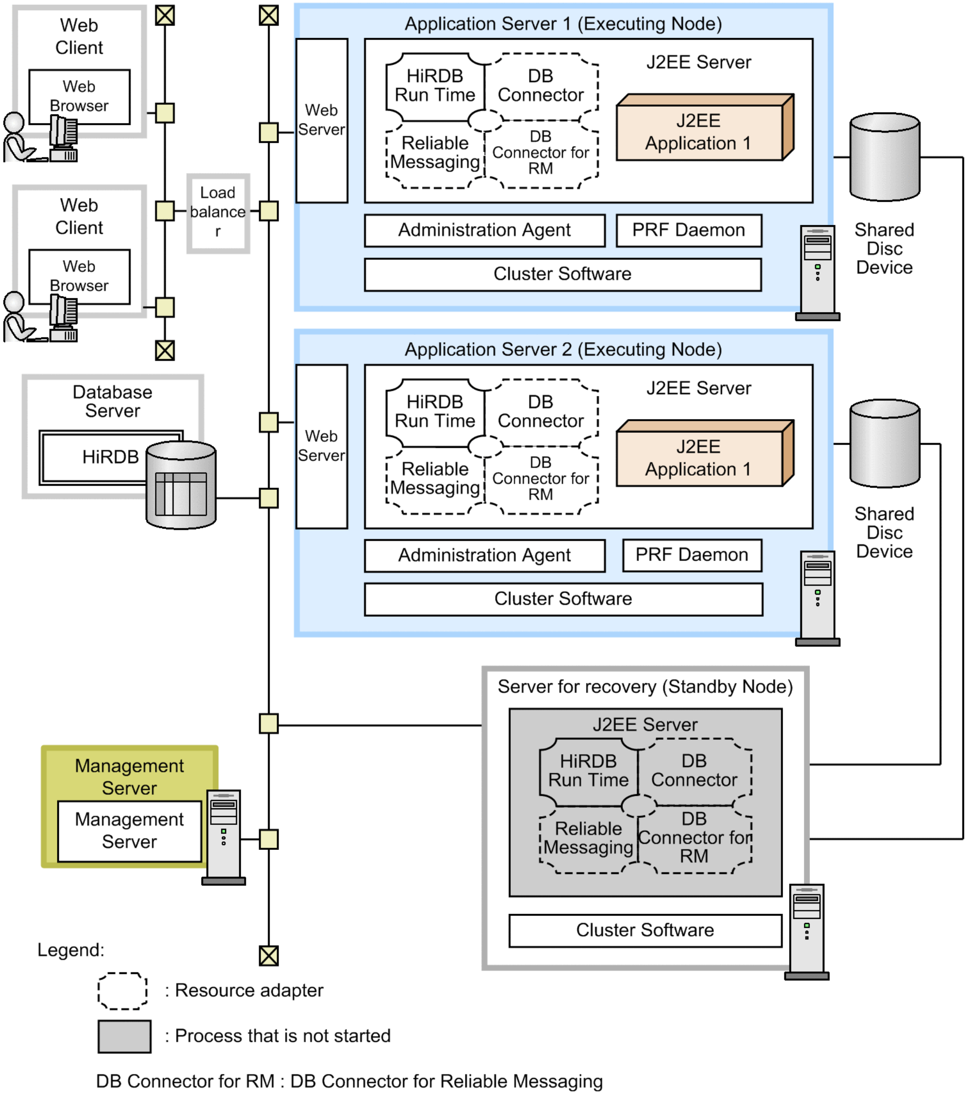
Application Server is subject to node switching in N-to-1 recovery system.

The resource adapters used in executing node are required for standby node as well when configuring N-to-1 recovery system. The following three types of configurations are described as an example for allocation of resource adapters and relation with system configuration:

- When the configuration of J2EE applications and resource adapters is the same for N executing node machines
- When the configurations of J2EE applications differ and all resource adapters are identical for N executing node machines
- When the configurations of J2EE applications and resource adapters differ for N executing node machines

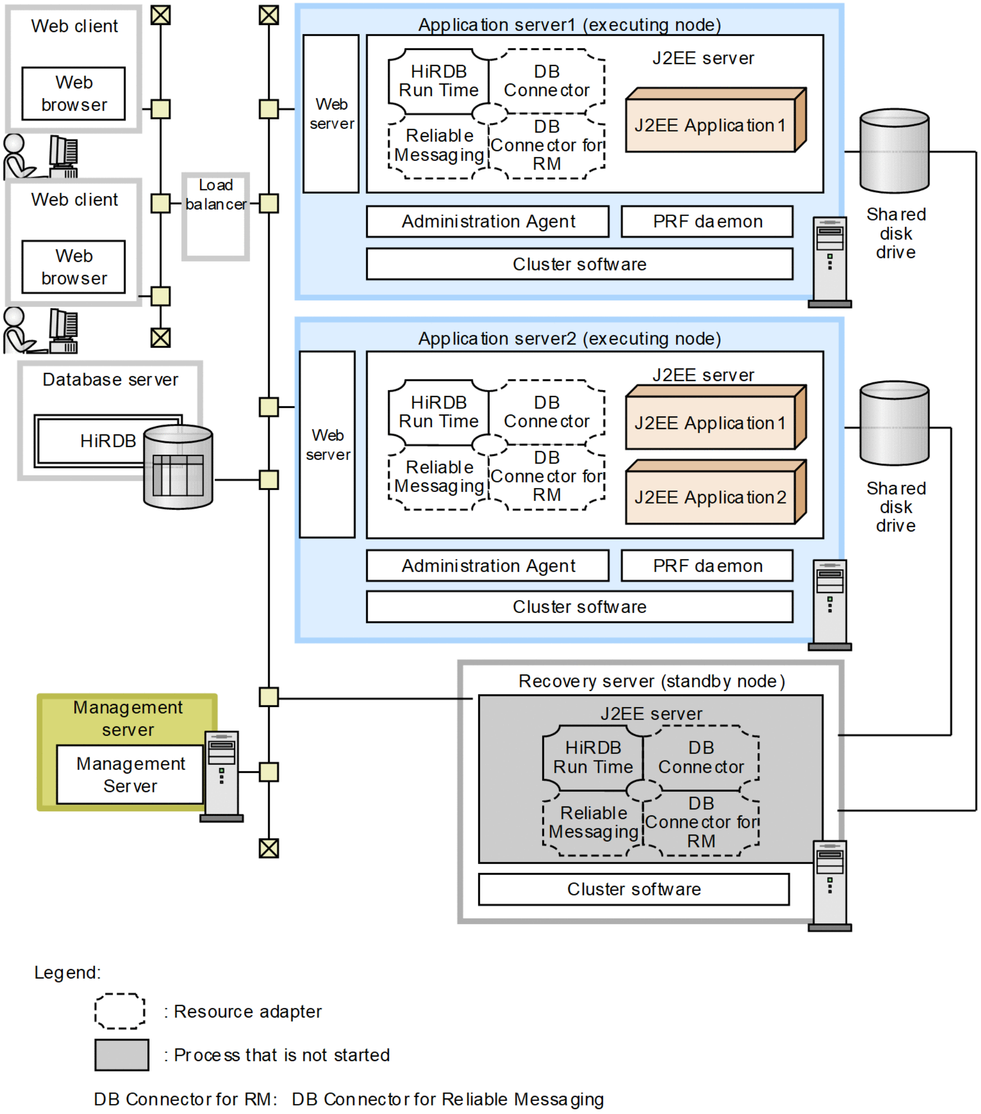
The following are the examples of respective configurations:

Figure 3–53: Example of system configuration of N-to-1 recovery system (when configuration of J2EE applications and resource adapters is same for N executing node machines)



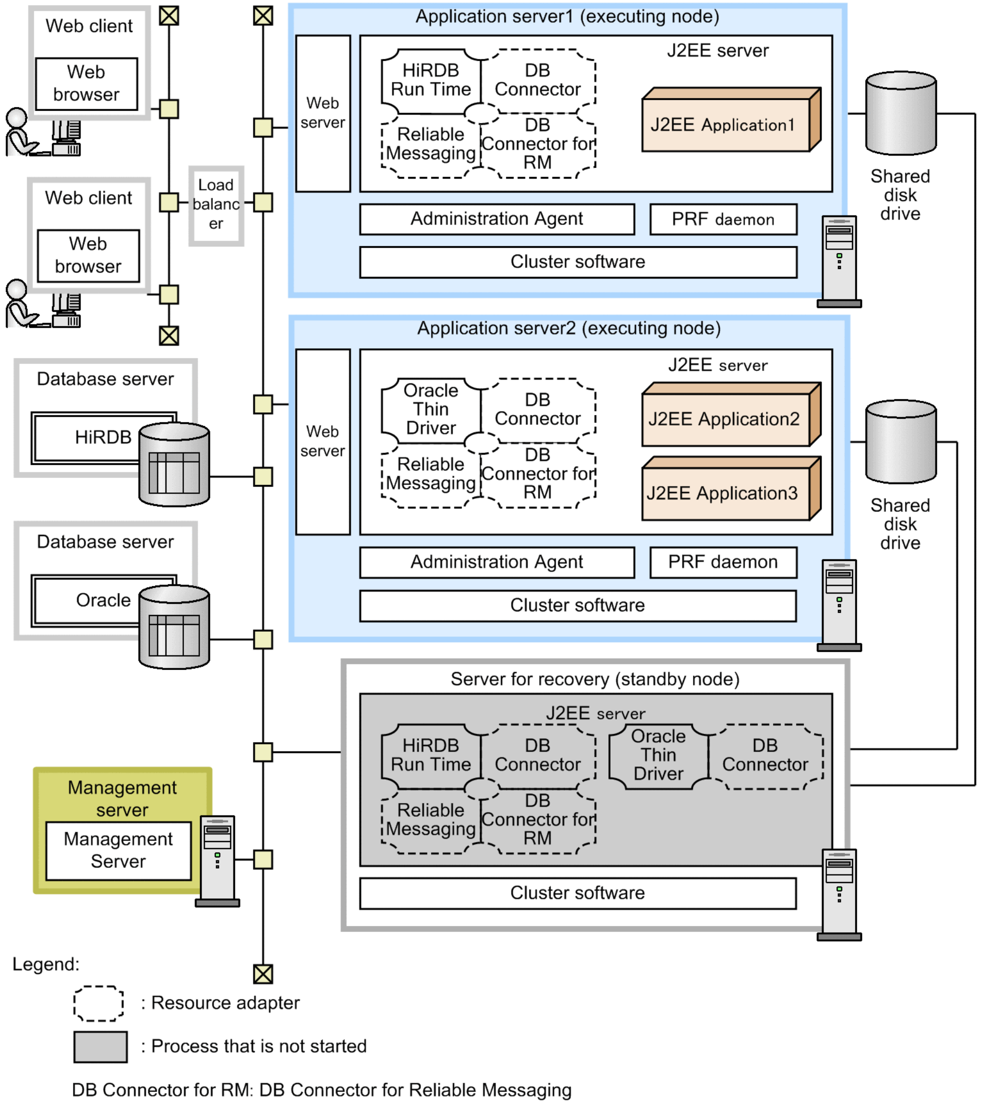
Note: For other legend items, see *3.2 Description of the system configuration*.

Figure 3–54: Example of system configuration of N-to-1 recovery system (when configuration of J2EE applications differ and all resource adapters are identical for N executing node machines)



Note: For other legend items, see *3.2 Description of the system configuration*.

Figure 3–55: Example of system configuration of N-to-1 recovery system (when configuration of J2EE applications and resource adapters differ for N executing node machines)



Note: For other legend items, see *3.2 Description of the system configuration*.

Features

- The executing node and standby node are in an N-to-1 ratio.
- A shared disk device is required. A volume group for the number of executing nodes (N) is required in the shared disk device.
- The CORBA Naming Service is started as in-process.
- The same resource adapters as in the executing node are required in the standby node. If a separate resource adapter is imported for each J2EE server in the executing node, you must import respective resource adapters for J2EE server in the standby node.

  Also, a J2EE application is not required in the J2EE server on the standby node.
- When the database has a cluster configuration and if Application Server identifies only the virtual address (Logical address), you can still establish the connection.

Node switching flow

When there is a failure in any of the J2EE servers started by Application Server in the executing node and when a J2EE server of the server exclusive for recovery is started by cluster software, the transaction used by the failed J2EE server is stopped. Thereafter, the cluster software of the J2EE server machine that failed and the cluster services of the corresponding standby nodes are stopped.

(2) Processes to be started on each machine

This subsection describes the required software and processes of the respective machines.

(a) Application Server machine (executing node)

You must install Application Server on the Application Server machine in the executing node.

The processes that must be started for using cluster software are as follows:

- Administration Agent

The required software and the processes to be invoked in the Application Server machine are different for each system configuration, depending on the used functions. Allocate the required software and the processes depending on the used functions.

(b) Application Server machine (Standby node)

You must install Application Server on the Application Server machine in the standby node.

Also, import all the resource adapters imported by J2EE server on Application Server machine of executing node.

The Application Server machine of standby node is in cold standby status.

However, the process does not start unless node switching takes place. When node switching is performed, the J2EE server process is started in recovery mode.

(c) Management Server machine

You must install Application Server on the Management Server machine.

Invoke the following process:

- Management Server

(d) Client machine

You must install the following software on Client machine:

For a Web client configuration
Web browser

For an EJB client configuration
uCosminexus Client (In Windows), Application Server

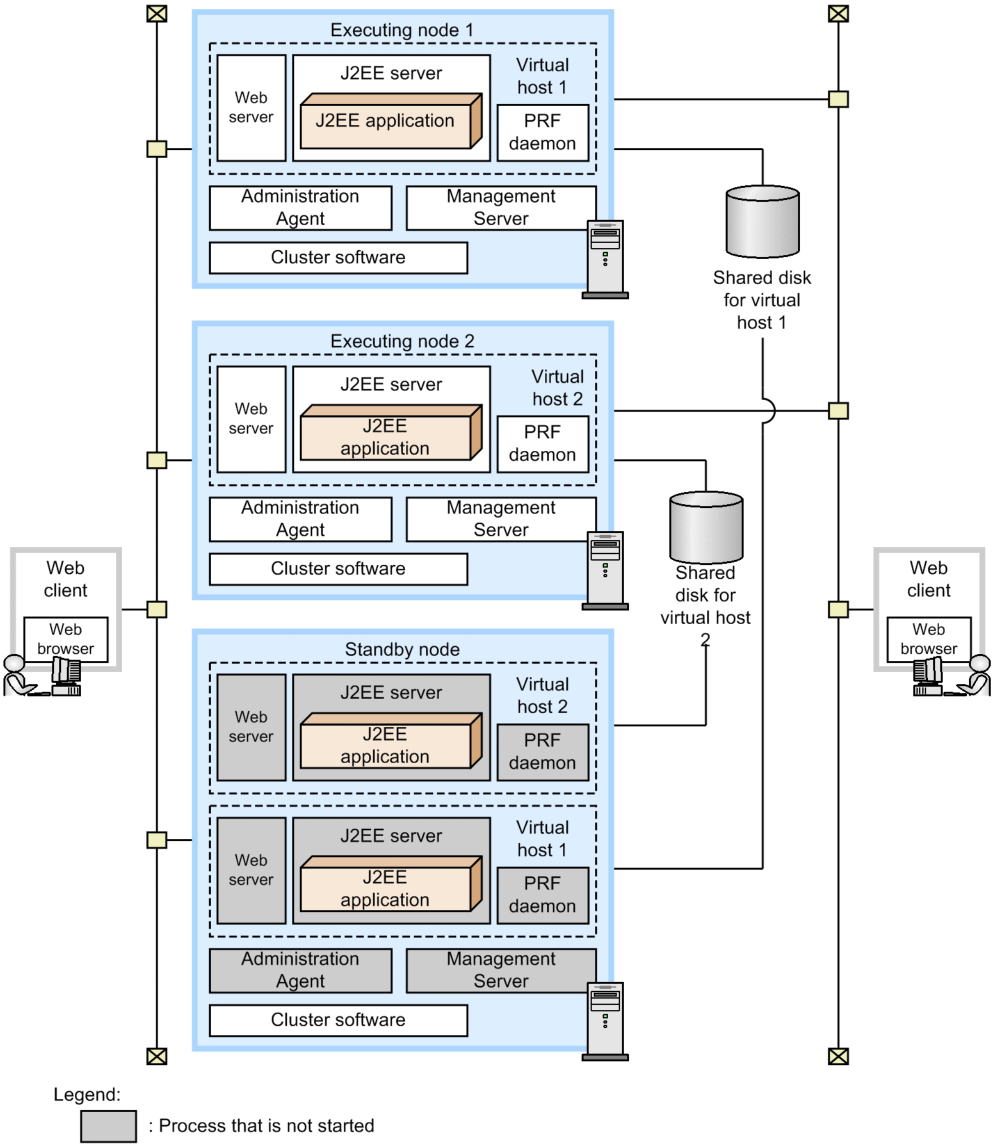
3.11.6 Configuration in which the executing node and standby node of the host unit management model are in an N-to-1 ratio

This subsection describes the configuration where node switching is performed for the host unit management model.

(1) Features of the system configuration

Configure the host unit management model in an N-to-1 ratio. The following is an example of the system configuration:

Figure 3–56: Example of the system configuration of node switching system for host unit management model using cluster software



Note: For other legend items, see *3.2 Description of the system configuration*.

Features

- Configuration where N machines for the executing node and one machine for the standby node of Application Server are allocated, and Management Server and Administration Agent are respectively allocated.
- Logical servers are allocated on separate hosts; however, externally they act as one logical server, whereas the Management Server instances operate on separate hosts and each has a respective management domain. Define one virtual host in one management domain, and respectively configure a host for the active node Application Server and a host for the standby node Application Server.
- Use the virtual IP address allocated by the cluster software as IP address used for operations of each Application Server, and use the actual IP address as the IP address of Management Server and Administration Agent.
- A shared disk device is required for using a global transaction. One shared disk is required for each virtual host.

> **Reference note**
>
> The virtual host controls the starting and stopping of Application Server according to Administration Agent. The same IP address as the IP address used for operations is allocated, and the virtual host is defined in such a manner so that visually it seems to be the same host.

Node switching flow

Node switching is execute when there is a failure on the Application Server machine in the executing node. Execute node switching when there is a failure in Management Server, Administration Agent, or cluster software.

On executing node switching, the Administration Agent and Management Server are started by cluster software of Application Server machine of standby node. The processes of the logical server that were stopped until then are started by the started Administration Agent.

(2) Processes to be started on each machine

This subsection describes the required software and processes of the respective machines.

(a) Application Server machine (executing node/ standby node)

You must install Application Server on the Application Server machine.

The processes that must be started on the Application Server machine of a node switching system in the host unit management model are as follows:

- Administration Agent
- Management Server

Depending on the functions to be used, the software required on the Application Server machine and the processes to be invoked differ for each system configuration. Allocate the software and processes required according to the functions used.

(b) Client machine

You must install the following software on the client machine:

For Web client configuration

Web browser

For EJB client configuration

uCosminexus Client (In Windows), Application Server

3.12 Deploying a process for the output of the performance analysis trace file

This section describes the deployment of a program that outputs the performance analysis trace file. This process must be deployed always on the Application Server system. Make sure that you deploy the process on the Application Server machine. You may or may not deploy the process on the EJB client machine.

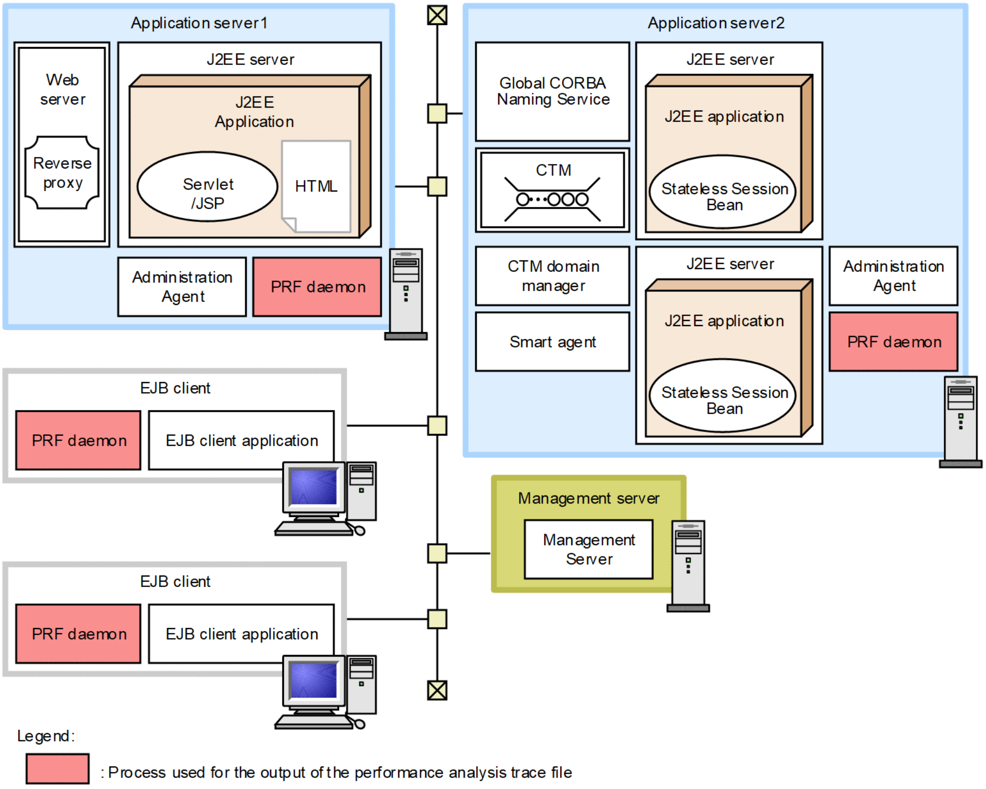
The performance analysis trace file is output by the PRF daemon (performance tracer).

3.12.1 Features of the system configuration

Deploy the PRF daemon on a machine with Application Server and the EJB client application running.

The following figure shows an example of deploying the PRF daemon:

Figure 3–57: Example of deploying the PRF daemon



Note: For other legend items, see *3.2 Description of the system configuration*.

Features

- The trace file for analyzing the system performance can be output.
- The trace file can be used for identifying the points where error has occurred.

Mechanism to output the performance analysis trace

When a request is processed between the client and the server, each functionality layer of Application Server or the EJB client application outputs the performance analysis information in a buffer. The PRF daemon outputs this trace information in the files of each machine. You can use Management Server to collect the trace information output in the files.

3.12.2 Required software and the processes to be invoked on the respective machines

Invoke the following process to output the performance analysis trace file:

- PRF daemon

The other required software and the processes to be invoked in Application Server are different for each system configuration, depending on the functionality used. Deploy the required software and processes as per the functionality used.

3.13 Determining integration with products other than Application Server

This section describes the system configuration for integrating with products other than Cosminexus Application Server.

The following configuration is explained here:

- Configuration when using JP1 for operation
  This configuration uses JP1 as the operation management program.
- Configuration for invoking Message-driven Beans from SUP of OpenTP1 using the TP1 inbound integrated function
  Configuration for invoking Message-driven Beans of Application Server from SUP of OpenTP1 using the TP1 inbound integrated function.
- Configuration where Stateless Session Beans are invoked from a client other than the EJB client using the CTM gateway function.
  Configuration where TPBroker client or TPBroker OTM client is used as client.

3.13.1 Configuration when using JP1 for operations

This subsection describes the deployment of the operation management program when you use JP1 for operations.

When integrating with JP1/IM or using functionality, such as custom job definition and scenarios, provided by Application Server to integrate with JP1/AJS, Management Server must be used to perform operations. For details about determining a system configuration using Management Server, see *3.9.1 Configuration wherein the Management Server is deployed on each Management Server* or *3.9.2 Configuration for deploying Management Server on each machine* and deploy the JP1 programs. For details about the programs required for integrating with JP1, see *12. Operating a JP1 Integrated System* in the *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*.

Note that if you use JP1/AJS, you can integrate the system without using Management Server, when you define a job or a scenario without using the Application Server functionality. In this case, first study the system configuration described in *3.9.3 Configuration when operating with commands*, see the JP1 manuals, and then decide the system configuration.

3.13.2 Configuration for invoking a Message-driven Bean from SUP of OpenTP1 using the TP1 inbound integrated function

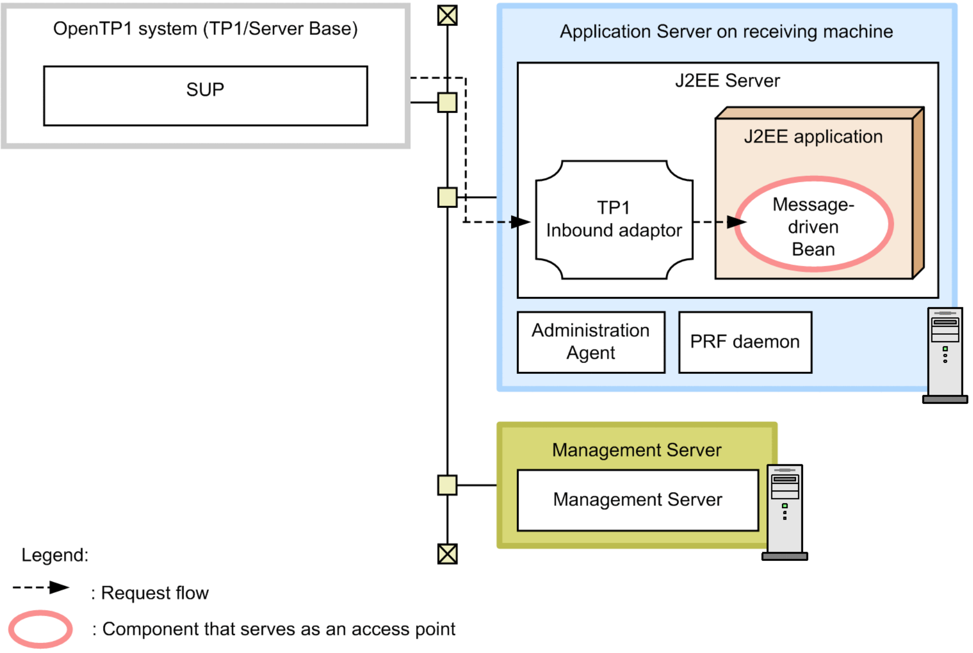
This subsection describes the configuration for invoking a Message-driven Bean of a J2EE server from SUP of an OpenTP1 system using the TP1 inbound integrated function.

(1) Features of the system configuration

The is one type of message driven system.

The following figure shows an example of a message driven system configuration using the TP1inbound integrated function.

Figure 3–58: Example of message driven system configuration (when using the TP1 inbound integrated function)



Note: For other legend items, see *3.2 Description of the system configuration*.

Features

The TP1 inbound adapter running on a J2EE server receives the requests sent by SUP running on TP1/Server Base and executes the process. You can access the Message-driven Bean on the J2EE server from SUP by the same procedure# as invoking the SPP.

#

Some of the procedures are different. For details, see *4. Calling Application Server from Open TP1 (TP1 Inbound Linkage Functionality)* in the *uCosminexus Application Server Common Container Functionality Guide*.

Request flow

The Message-driven Bean, which is the access point, and the library of the TP1 inbound adapter, which is the resource adapter, run on the J2EE server of Application Server.

The TP1 inbound adapter of Application Server receives the requests from SUP running on TP1/Server Base of the OpenTP1 system and invokes the Message-driven Bean.

(2) Processes to be invoked and software required on the respective machines

This subsection describes the software required on the respective machines and processes to be started.

(a) Application Server machine

You must install Application Server on the Application Server machine (Application Server on the server machine).

Invoke the following processes:

- J2EE server
- Administration Agent
- PRF daemon

(b) OpenTP1 system (TP1/Server Base)

You must install TP1/ Server Base on the OpenTP1 system.

You must invoke the processes required for SUP execution.

(c) Management Server machine

You must install Application Server on the Management Server machine.

Invoke the following process:

- Management Server

3.13.3 Configuration in which a Stateless Session Bean is invoked from other than an EJB client using the CTM gateway function

For a system using CTM, the J2EE application operating on the Application Server machine is invoked from the following client other than the EJB client:

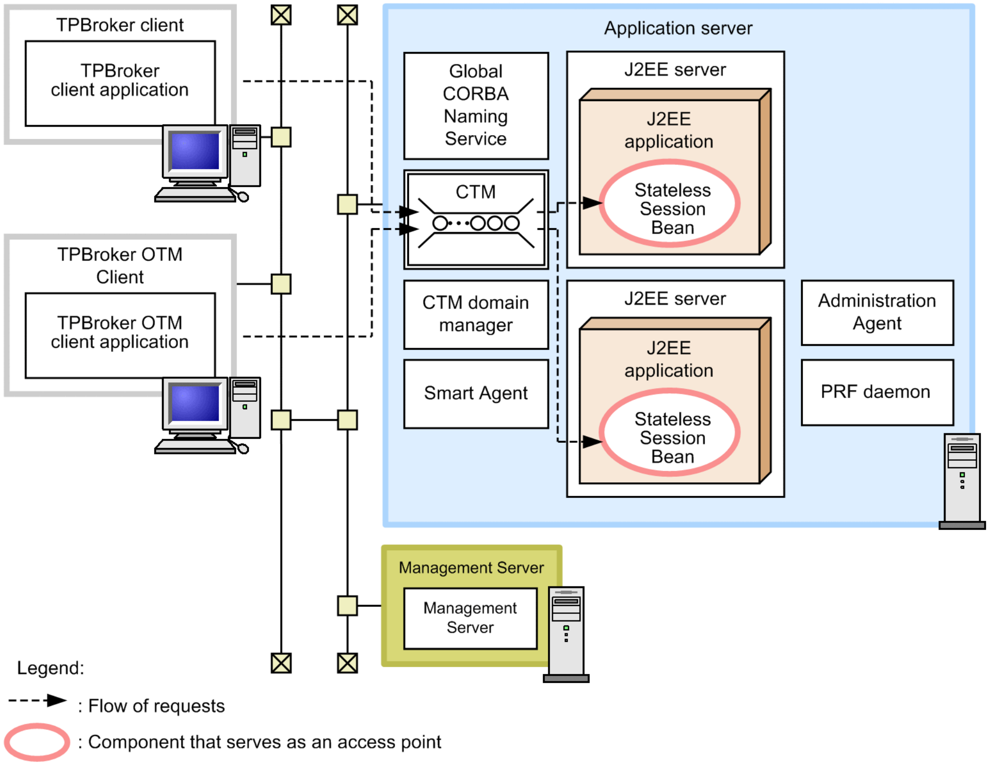
- *TPBroker client*
  Client application developed with TPBroker Version 5 or later.
- *TPBroker OTM client*
  Client application developed with TPBroker Object Transaction Monitor.

For details about how to develop the client application to invoke the application on the J2EE server from these clients, see the documentation for the CORBA/OTM gateway function of CTM.

In this configuration, CTM provides a function as a gateway to invoke the J2EE application from TPBroker client or TPBroker OTM client.

The following figure shows an example of a system configuration that invokes the J2EE application through CTM from TPBroker client or TPBroker OTM client:

Figure 3–59: Example of configuration that invokes J2EE application through CTM from TPBroker client or TPBroker OTM client



Note: For other legend items, see *3.2 Description of the system configuration*.

The request from the TPBroker client or TPBroker OTM client is passed to the J2EE application on the J2EE server through the processes provided by CTM.

Among the CTM processes, the process of receiving requests is different for TPBroker client and TPBroker OTM client. For TPBroker client, the CTM regulator process receives the request. For TPBroker OTM client, the request is received by the OTM gateway process. The CTM regulator process and OTM gateway process are started concurrently when starting the CTM daemon.

3.14 Managing optional processes with operation management

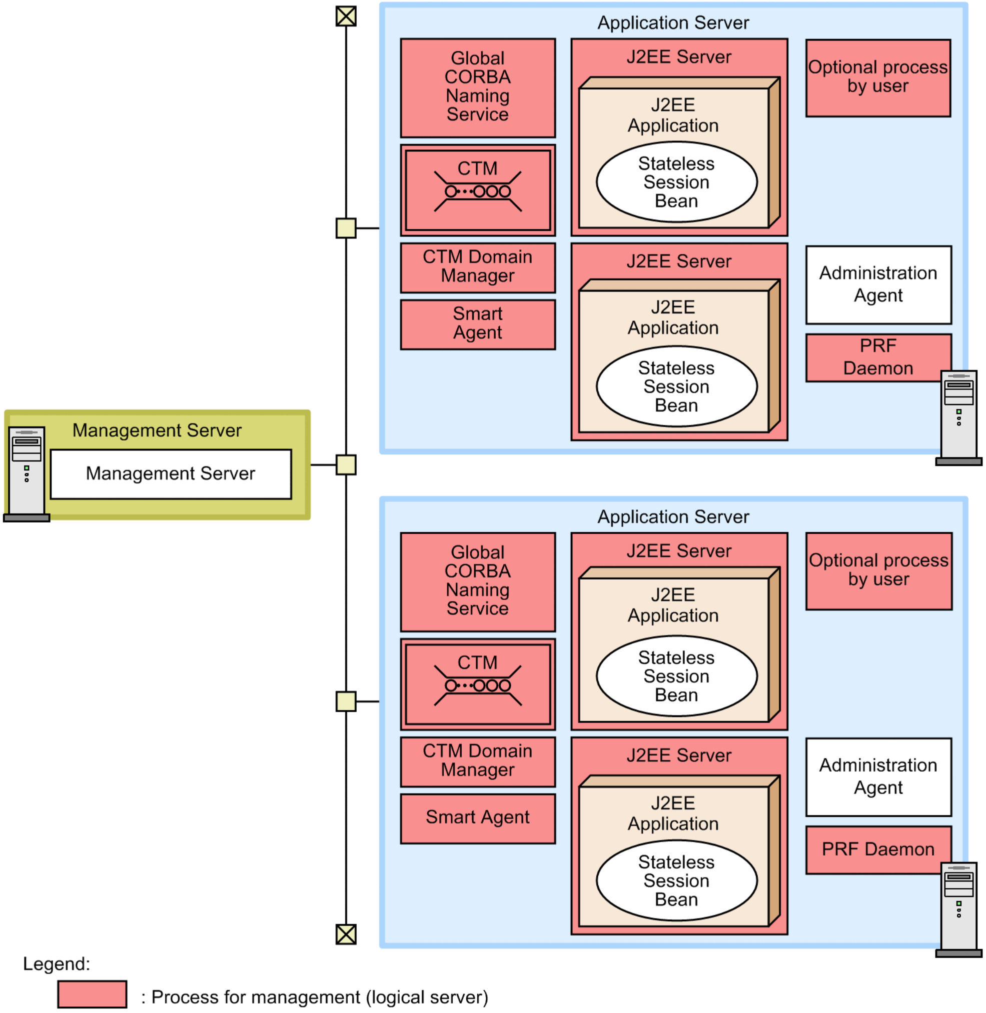
This section describes a system configuration in which Management Server can manage operations of a user-defined optional process. In such a configuration, you deploy the optional process as a user server defined as a logical server. A user server that is defined as a logical server is called a *logical user server*. When an optional process is defined as a logical user server, you can use the Smart Composer functionality commands for starting, stopping, and monitoring the status of the optional process.

3.14.1 Features of the system configuration

In this system configuration, a user server defined as a logical server is deployed on a system where Management Server performs operation management. You can deploy a Management Server instance to be used for operation management either on the management server or on each machine. For details about deploying Management Server, see *3.9 Determining the deployment of the operation management process*.

The following figure shows an example of a configuration for deploying the user server. In this configuration example, a user optional process is executed on the Application Server machine. This user optional process is defined and deployed as a logical user server so that Management Server can manage the process.

Figure 3–60: Example of a configuration with a user server deployed



Legend:

🟥 : Process for management (logical server)

Note: For other legend items, see *3.2 Description of the system configuration*.

Feature

- You can use the Management Server to perform operation management of an optional process.

3.14.2 Processes to be invoked on the respective machines

The software required on the respective machines and the processes to be invoked in a configuration with an optional process deployed as a user server are explained below:

(1) Application server machine

Depending on the functions to be used, the software required on the Application Server machine and the processes to be invoked are different for each system configuration. Deploy the software and processes required for each function.

(2) Management server machine

You need to install Application Server on the Management Server machine.

Invoke the following process:

- Management Server

3.15 TCP/UDP port numbers used by Application Server processes

This subsection describes the TCP/UDP port numbers used by Application Server processes.

When the port number is not explicitly defined, a port number is automatically assigned by Application Server to a port with the default value `(Floating)`.

The following table describes the TCP/UDP port numbers used by Application Server. Depending on the OS being used, the firewall setting may be at the host level instead of the network level. In the case of such firewalls, filtering takes place for all communication other than that with the localhost (127.0.0.1), including communication within the same host. In such cases, even for the ports that communicate only within the host, you need to set the filter settings so as to permit that communication.

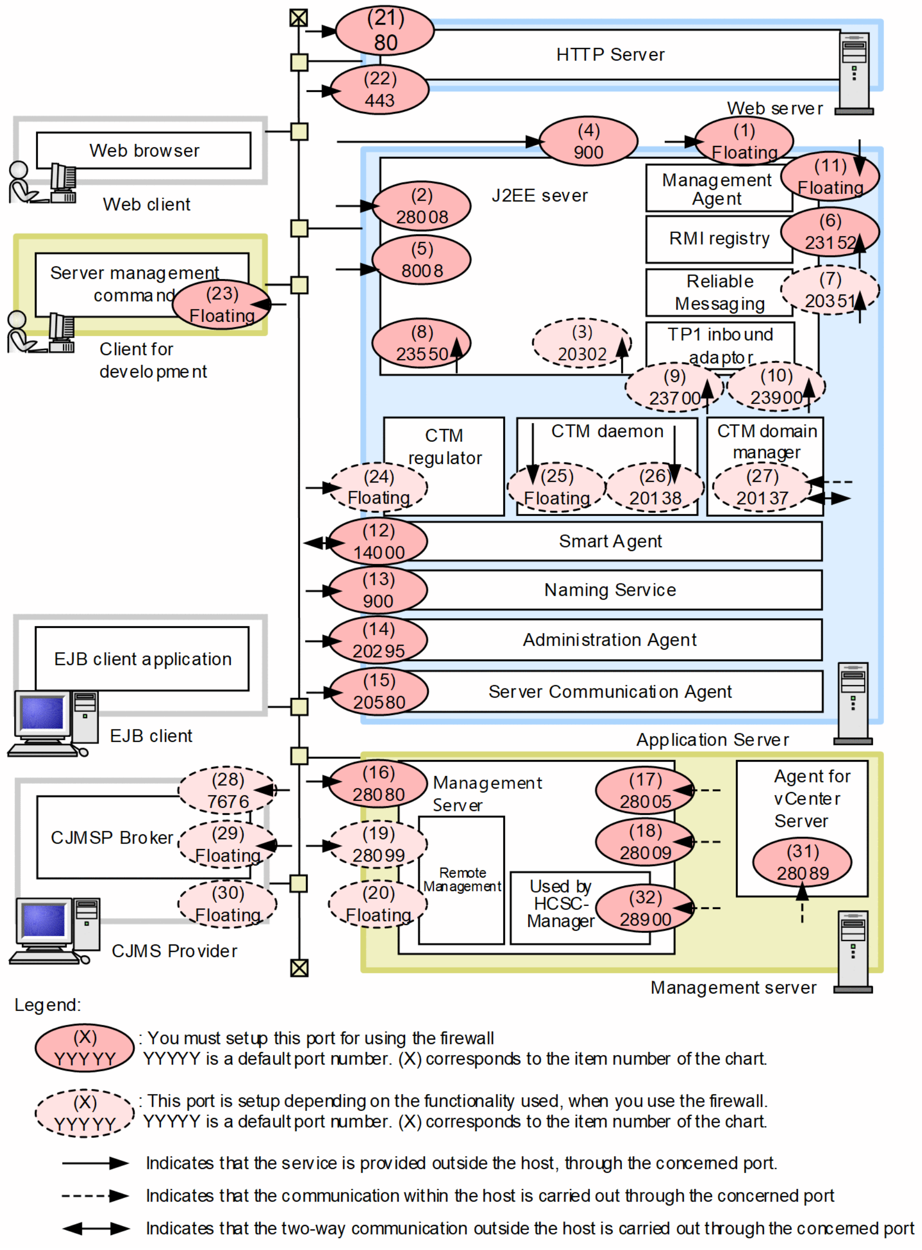
Table 3–3: TCP/UDP port numbers used by Application Server

No.	Processes	Explanation	Default values
(1)	J2EE server	Request reception port of an EJB container.	`(Floating)`
(2)		Management communication port.	`28008`
(3)		Transaction recovery processing communication port when using transaction services. Required when using transaction services.	`20302`
(4)		Request reception port of the Naming Service that is invoked by the in-process.	`900`
(5)		Request reception port of communication by HTTP and WebSocket protocols.	`8008`
(6)		Request reception port of the RMI registry.	`23152`
(7)		A port for the event reception during application integration between multiple systems using shared queue.	`20351`
(8)		Request reception port when acquiring operation information.	`23550`
(9)		A port awaiting RPC request from OpenTP1.	`23700`
(10)		A port awaiting the synchronization request from OpenTP1.	`23900`
(11)	Management Agent	A port used for the communication of Management agent.	`(Floating)`
(12)	Smart agent	Port environment variable for communication of smart agent. Smart Agent is required for the two-way communication through UDP.	`14000`
(13)	Naming Service	Request reception port argument of Naming Service (use Cosminexus TPBroker).	`900`
(14)	Administration Agent	A port used by the Administration Agent in the communication with the Management Server.	`20295`
(15)	Server Communication Agent	A port used by Server Communication Agent for communicating with Virtual server manager.	`20580`
(16)	Management Server	An http port of the Management Server.	`28080`
(17)		A port for termination request of Management Server. Required for communication within the host.	`28005`
(18)		A port for internal communication of Management Server. Required for communication within the host.	`28009`
(19)		A port for connection to the Manager remote management function.	`28099`
(20)		A port for client connection to the Manager remote management function.	`(Floating)`

No.	Processes	Explanation	Default values
(21)	Cosminexus HTTP Server	An `http` port of Cosminexus HTTP Server.	`80`
(22)		An `https` port of Cosminexus HTTP Server.	`443`
(23)	Server management command	A port with which the server management command communicates with the J2EE server.	`(Floating)`
(24)	CTM regulator	Basic value of the port where CTM regulator receives requests from the EJB client. Use basic value + process count only. Mandatory when using CTM.	`(Floating)`
(25)	CTM daemon	Port where the CTM daemon receives the requests from the EJB client. Mandatory when using CTM.	`(Floating)`
(26)		Port for communication by CTM daemon with other daemon or J2EE server. Mandatory when using CTM.	`20138`
(27)	CTM domain manager	Port for communication by CTM domain manager with other CTM domain manager. Mandatory when using CTM, to communicate TCP and UDP (broadcast).	`20137`
(28)	CJMSP Broker	Port of broker of Cosminexus JMS provider for receiving requests from resource adapter or commands.	`7676`
(29)		Port of broker of Cosminexus JMS provider for establishing connection with resource adapter.	`(Floating)`
(30)		Port of broker of Cosminexus JMS provider for establishing connection with commands.	`(Floating)`
(31)	Management Server	Internal communication port of Management Server used from HCSC-Manager.	`28900`

The following figure shows the TCP/UDP port numbers used by the Application Server process. (x) corresponds to the item numbers in the table:

Figure 3–61: TCP/UDP port numbers used by Application Server



: You must setup this port for using the firewall
YYYYY is a default port number. (X) corresponds to the item number of the chart.

: This port is setup depending on the functionality used, when you use the firewall.
YYYYY is a default port number. (X) corresponds to the item number of the chart.

Indicates that the service is provided outside the host, through the concerned port.

Indicates that the communication within the host is carried out through the concerned port

Indicates that the two-way communication outside the host is carried out through the concerned port

Note: For other legend items, see *3.2 Description of the system configuration*.

The following table lists the locations for specifying port numbers. The item number in the table corresponds to the item number in the figure.

Table 3–4: Locations for specifying the TCP/UDP port numbers used in Application Server

No.	Definition files	Setup target	Parameter name[#1]
(1)	Easy Setup definition file	Logical J2EE server (`j2ee-server`)	`vbroker.se.iiop_tp.scm.iiop_tp.listener.port`
(2)	Easy Setup definition file	Logical J2EE server (`j2ee-server`)	`ejbserver.http.port`
(3)	Easy Setup definition file	Logical J2EE server (`j2ee-server`)	`ejbserver.distributedtx.recovery.port`

No.	Definition files	Setup target	Parameter name#1
(4)	Easy Setup definition file	Logical J2EE server (j2ee-server)	inprocess.ns.port
(5)	Easy Setup definition file	Logical J2EE server (j2ee-server)	webserver.connector.nio_http.port
(6)	Easy Setup definition file	Logical J2EE server (j2ee-server)	ejbserver.rmi.naming.port
(7)	Connector property file	Reliable Messaging	RMSHPort specified in <config-property> tag#2
(8)	Easy Setup definition file	Logical J2EE server (j2ee-server)	ejbserver.rmi.remote.listener.port
(9)	Connector property file	TP1 inbound adapter	scd_port specified in <config-property> tag#3
(10)	Connector property file	TP1 inbound adapter	trn_port specified in <config-property> tag#3
(11)	Easy Setup definition file	Logical J2EE server (j2ee-server)	mngagent.connector.port
(12)	Easy Setup definition file	Logical smart agent (smart-agent)	smartagent.port
(13)	Easy Setup definition file	Logical J2EE server (j2ee-server)	ejbserver.naming.port
(14)	Adminagent.properties	Administration Agent	adminagent.adapter.port key
(15)	sinaviagent.properties #4	Server Communication Agent	sinaviagent.port key
(16)	mserver.properties	Management Server	webserver.connector.http.port key
(17)	mserver.properties	Management Server	webserver.shutdown.port key
(18)	mserver.properties	Management Server	webserver.connector.ajp13.port key
(19)	mserver.properties	Management Server	com.cosminexus.mngsvr.management.port key
(20)	mserver.properties	Management Server	com.cosminexus.mngsvr.management.listen.port key
(21)	Easy Setup definition file	Logical Web server (web-server)	Listen
(22)	Easy Setup definition file	Logical Web server (web-server)	Listen
(23)	usrconf.properties (system property file for server management command)	Server management command	vbroker.se.iiop_tp.scm.iiop_tp.listener.port key
(24)	Easy Setup definition file	Logical CTM (component-transaction-monitor)	ctm.RegOption
(25)	Easy Setup definition file	Logical CTM (component-transaction-monitor)	ctm.EjbPort

No.	Definition files	Setup target	Parameter name[#1]
(26)	Easy Setup definition file	Logical CTM (component-transaction-monitor)	ctm.port
(27)	Easy Setup definition file	Logical CTM domain manager (ctm-domain-manager)	cdm.port
(28)	config.properties	CJMSP Broker	imq.portmapper.port key
(29)	config.properties	CJMSP Broker	imq.jms.tcp.port key
(30)	config.properties	CJMSP Broker	imq.admin.tcp.port key
(31)	mserver.properties	Management Server	ejbserver.naming.port key

Legend:

--: Not applicable

#1

If the setup file is an Easy Setup definition file, specify the value specified in <param-name> in the <configuration> tag.

#2

RMSHPort is a configuration property specified in the property definition of the resource adapter Cosminexus RM. For details about RMSHPort, see *6. Configuration Properties* in the manual *Cosminexus Reliable Messaging*.

#3

scd_port and trn_port are the configuration properties specified in the property definition of the resource adapter TP1 inbound adapter. For details about scd_port and trn_port, see *4.12.2 Setting up a resource adapter* in the *uCosminexus Application Server Common Container Functionality Guide*.

#4

For details about Server Communication Agent, see the documents related to Server Communication Agent.

Reference note

The following table describes the locations for specifying TCP/UDP port numbers when setting up Application Server:

Table 3–5: Locations for specifying TCP/UDP port numbers when setting up Application Server

No.	Setup location when setting up by editing the file
(1)	vbroker.se.iiop_tp.scm.iiop_tp.listener.port key in usrconf.properties
(2)	ejbserver.http.port key in usrconf.properties
(3)	ejbserver.distributedtx.recovery.port key in usrconf.properties
(4)	ejbserver.naming.port key in usrconf.properties
(5)	ebserver.connector.nio_http.port key in usrconf.properties
(6)	ejbserver.rmi.naming.port key in usrconf.properties
(7)	RMSHPort# specified in the <config-property> tag of the Connector property file
(8)	ejbserver.rmi.remote.listener.port key of usrconf.properties
(9)	scd_port specified in the <config-property> tag of the resource adapter of Connector property file of TP1 inbound adapter

No.	Setup location when setting up by editing the file
(10)	`trn_port` specified in the `<config-property>` tag of the resource adapter of Connector property file of TP1 inbound adapter
(11)	`mngagent.connector.port` key in the `mngagent.`*real-server-name*`.properties` file
(12)	Environment variable *OSAGENT_PORT*
(13)	- When auto-starting CORBA Naming Service in in-process or out-process `ejbserver.naming.port` key of `usrconf.properties` - When manually starting CORBA Naming Service Specify `-Dvbroker.se.iiop_tp.scm.iiop_tp.listener.port=`*port-number* in the command argument of the `nameserv` command.
(14)	`adminagent.adapter.port` key in `adminagent.properties`
(15)	`sinaviagent.port` key in `sinaviagent.properties`
(16)	`webserver.connector.http.port` key in `mserver.properties`
(17)	`webserver.shutdown.port` key in `mserver.properties`
(18)	`com.cosminexus.mngsvr.management.port` key in `mserver.properties`
(19)	`com.cosminexus.mngsvr.management.listen.port` key in `mserver.properties`
(20)	`Listen` directive or `Port` directive of `httpsd.conf`
(21)	`Listen` directive or `Port` directive of `httpsd.conf`
(22)	`vbroker.se.iiop_tp.scm.iiop_tp.listener.port` key in `usrconf.properties` (system properties file for server management commands)
(23)	Argument of the `ctmregltd` command or `ctmstart` command - `CTMEjbPort`
(24)	Argument of the `ctmstart` command - `CTMEjbPort`
(25)	Argument of the `ctmstart` command - `CTMPort`
(26)	Argument of the `ctmdmstart` command - `CTMPort`.
(27)	`imq.portmapper.port` key in `config.properties`
(28)	`imq.jms.tcp.port` key in `config.properties`
(29)	`imq.admin.tcp.port` key in `config.properties`
(30)	`vmx.vcenterserver.agent.port` key in `vmx.properties`
(31)	`ejbserver.naming.port` key in `mserver.properties`

#

`RMSHPort` is a configuration property specified in the property definition of the resource adapter Cosminexus RM. For details about `RMSHPort`, see *6. Configuration Properties* in the manual *Cosminexus Reliable Messaging*.

Important note

Notes on the standby port for a server (In UNIX)

In UNIX, when all the following conditions are satisfied, a connection might be successfully established with a TCP port that is not in a standby status:

- An attempt is made to establish a connection with a port that is not in a standby status
- The host itself is the connection target, and the port is in the range of the temporary port numbers (the range of port numbers that are dynamically allocated by the OS)

When this event occurs, the assumed process communication cannot be executed and a timeout occurs. To avoid this event, specify a value outside the range of the temporary port numbers as the standby port of the server. You can check the range of the temporary port numbers in the following files:

In AIX

Minimum value (32768): `no -o tcp_ephemeral_low`

Maximum value (65535): `no -o tcp_ephemeral_high`

In Linux

```
/proc/sys/net/ipv4/ip_local_port_range
```

For details on how to set up the standby port of a server, see the documentation for the OSs.

4 Determining the System Configuration (Batch Application Execution Platform)

This chapter explains how to determine a system configuration when building the batch application execution platform. The standard system configuration patterns are described according to the flow of system designing. This chapter also describes the components, processes, and the processing flow that you must consider.

For determining the system configuration of the J2EE application execution platform, see *3. Determining the System Configuration (J2EE Application Execution Platform)*.

4.1 Points to be considered when determining the system configuration

This section describes the points to be considered for determining the system configuration, when you use Application Server as a batch application execution platform.

4.1.1 Purpose and configuration of system

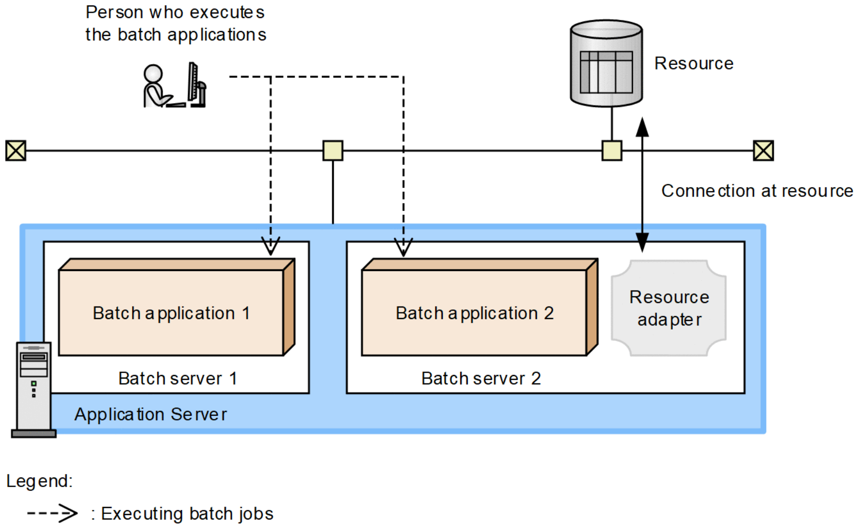
The batch application execution platform is an environment to execute batch applications implemented as Java applications. The batch applications are implemented as Java applications that do not use the J2EE applications such as JSPs, servlets, and Enterprise Beans. Note that you can invoke Enterprise Beans running on J2EE servers from batch applications.

The batch applications run on batch servers. The resource adapters used when connecting the batch application to the resource also run on batch servers.

The batch applications are executed with batch execution commands.

The following figure shows a system configuration for executing batch applications. In this figure, two batch servers are operating on one application server for each batch application.

Figure 4–1: Example of a system configuration for executing batch applications

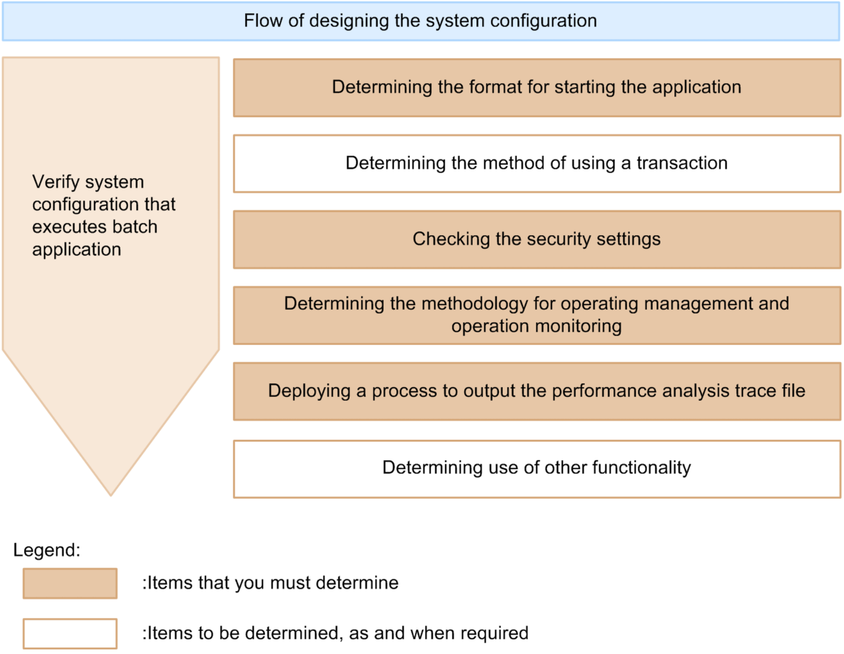


Besides this, when you connect to Enterprise Beans running on J2EE server from batch applications, you can also connect to back-end systems executing online processing. For details about back-end systems, see *3.1.1 Purpose and configuration of the system*.

4.1.2 Procedure for designing the system configuration

Design the system configuration according to the following procedure:

Figure 4–2: Flow of designing the system configuration (for a batch application execution platform)


Flow of designing the system configuration
Verify system configuration that executes batch application
Determining the format for starting the application
Determining the method of using a transaction
Checking the security settings
Determining the methodology for operating management and operation monitoring
Deploying a process to output the performance analysis trace file
Determining use of other functionality
Legend:
▨ :Items that you must determine
□ :Items to be determined, as and when required

You can choose any order to determine configuration for the functionality to be used.

(1) Determining the format for starting the application

Determine the format for starting batch applications. Choose from the following two types:

- **Starting on batch servers**

  This is the format for starting Java applications on batch servers. You can reduce overhead involved in starting JavaVM using a batch server that is a resident-type JavaVM process. Also, you can use CTM to schedule the execution of the batch applications running on the batch server.

  When applications run on batch servers, you can also use DB Connector as the resource adapter.

- **Starting individually using the cjclstartap command**

  This is the format for starting Java applications in the same way as `Java` commands. For this format, you must start JavaVM each time batch applications are executed.

If you choose to start the application on batch servers, it will lead to the following items to be determined. If you choose to use the `cjclstartap` command, the following items to be determined will not be applicable:

(2) Determining the method of using a transaction

For connecting to resources, determine the method of using a transaction. Choose from the following two types:

- **Using a DB Connector**

  In this method, a DB Connector is used that is a resource adapter provided in the Application Server. You can use connection pooling and statement pooling as the functionality of the DB Connector. You can also use the functionality to control the occurrence of full garbage collection. Note that you can manage a local transaction on batch servers.

- **Directly using a JDBC driver**

  In this method, the processing required for managing the transaction is implemented using APIs provided in a JDBC driver.

For details about the configuration of the resource adapter and resources if you are using DB Connectors, see *3.3.2 Resource types and resource adapters*. However, for a batch server, you can connect only to the database, as a resource.

(3) Checking the security settings

Start the batch server after disabling security protection with `SecurityManager`.

(4) Deploying a process to output the performance analysis trace file

Check if the PRF daemon (performance tracer) that is a process to output the performance analysis trace file is deployed. Deploy a PRF daemon on each batch server.

(5) Determining the methodology for operating management and operation monitoring

Check if the Management Server that is a process for executing operating management and operation monitoring is deployed. For a batch application execution platform, the Management Server is deployed on the same machine as the batch server (use the host unit management model).

(6) For using other functionality

Determine the following configurations according to the functionality to be used:

- **Configuration in which servers are integrated**
  Determine this configuration when you want to invoke an Enterprise Bean running on the J2EE server from the batch application. See *3.5 Determining integration between servers*.
- **Configuration for node switching using cluster software when a failure occurs**
  Determine this configuration for node switching on batch servers.
  See the following explanation:
  - *3.11.4 Configuration in which executing node and standby node of Application Server are mutually standby*.

  When reading the description provided in the above references, please substitute *J2EE server* for *batch server*.

4.1.3 Precautions for the TCP/UDP port used in a system for executing batch applications

This subsection describes the TCP/UDP port used in a system for executing batch applications for each process. The following table describes the TCP/UDP port used in a system for executing batch applications:

Table 4–1: TCP/UDP port used in a system for executing batch applications

No.[#1]	Process	Explanation
(1)	Batch server	Request-receiving port of the EJB container.
(2)		Communication port for management.
(3)		Port for receiving requests from the Web server (redirector).
(5)		Request-receiving port of the Naming Service invoked with in-process.
(7)		Request receiving port of the RMI registry.

No.#1	Process	Explanation
(9)		Port for receiving requests when acquiring the operation information.
(13)	Smart Agent#2	Port environment variable for Smart Agent communication. Smart Agent is required for the two-way communication through UDP.
(25)	CTM regulator#2	Fiducial value of the port where the CTM regulator receives requests from the EJB client. Only *fiducial value* + *number of processes* are used. Required when CTM is used.
(26)	CTM daemon#2	Port where CTM daemon receives requests from the EJB client. Required when CTM is used.
(27)		Port for communicating CTM daemons with other daemons and J2EE servers. Required when CTM is used.
(28)	CTM domain manager#2	Port for communicating the CTM domain manager with other CTM domain managers. Required for TCP and UDP communication (broadcast) when CTM is used.

Note

When you are instructed to use Smart Agent, some ports other than those described in the above table are used to communicate with the Smart Agent. For details on the Smart Agent, see the manual *Borland(R) Enterprise Server VisiBroker(R) Programmers Reference*.

#1

Corresponds to the numbers in *Table 3-3 TCP/UDP port numbers used by Application Server* of *3.15 TCP/UDP port numbers used by Application Server processes*.

#2

This process is required for using the scheduling functionality of batch applications.

In the following cases, make sure that the used ports are not duplicated:

- When concurrently using the J2EE server and batch server on the same machine
- When concurrently using multiple batch servers on the same machine

For details about the TCP/UDP ports used in each process, see *3.15 TCP/UDP port numbers used by Application Server processes*.

Important note

Notes on the standby port for a server (In UNIX)

In UNIX, when all the following conditions are satisfied, a connection might be successfully established with the TCP port that is not in a standby mode:

- An attempt is made to establish a connection with a port that is not in a standby status
- The host itself is the connection target, and the port is in the range of the temporary port numbers (range of port numbers that are dynamically allocated by the OS)

When this event occurs, the assumed process communication cannot be executed and a timeout occurs. To avoid this event, specify a value outside the range of the temporary port numbers as the standby port of the server. You can check the range of the temporary port numbers in the following files:

In AIX

Minimum value (32768): `no -o tcp_ephemeral_low`

Maximum value (65535): `no -o tcp_ephemeral_high`

In Linux

```
/proc/sys/net/ipv4/ip_local_port_range
```

For details on how to set up the standby port of a server, see the documentation for the OSs.

4.2 System configuration when using a batch server

This section describes the system configuration when you use a batch server. The system configuration for using a batch server differs depending on whether to use the scheduling functionality of the batch application.

4.2.1 System configuration of a system that does not use the scheduling functionality of batch applications

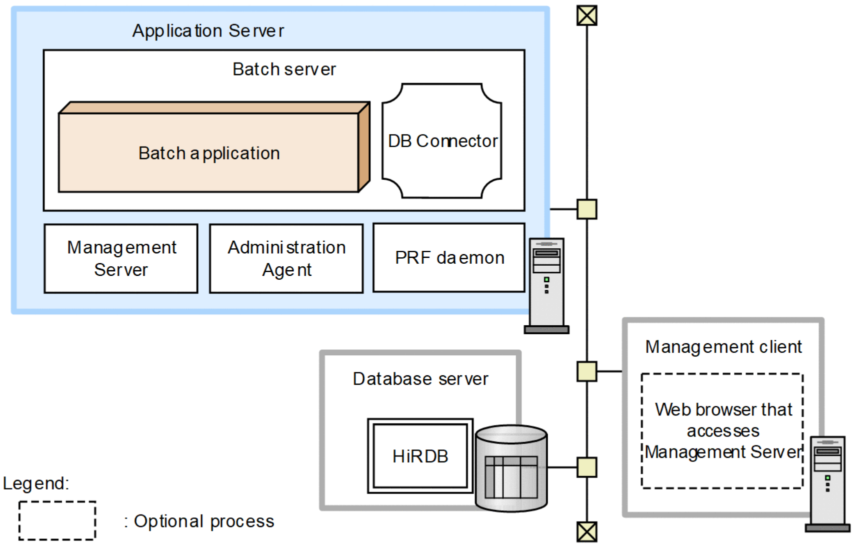
This subsection describes the system configuration that does not use the scheduling functionality of the batch applications, for using a batch server.

(1) Features of system configuration

This is a system configuration for using a batch server and not a CTM. In this case, the batch server is deployed on the Application Server. The batch server is built and operated as a Web system (j2ee-tier) of the Smart Composer functionality. Use batch execution commands to execute a batch application.

The following figure shows an example of the system configuration for deploying a batch server. In this example, HiRDB is accessed from the batch application using a resource adapter.

Figure 4–3: Example of the system configuration for deploying a batch server



For other legend items, see *3.2 Description of the system configuration*.

Features

You can use a DB Connector for connecting to a resource.

Flow of the process

The batch application executed accesses HiRDB via the DB Connector on the batch server.

(2) Processes and software required on each machine

The software and processes required on each machine are as follows:

(a) Application server machine

You must install Application Server on the Application Server machine. Note that for the development environment, you must install Developer.

The following processes are to be started:

- Batch server
- PRF daemon
- Management Server
- Administration Agent

When connecting to the database, you also require the software for connecting to the database that you want to use. For details about the products required to connect to the database, see *3.6.1 Configuration when using a local transaction*. However, you can use only the following databases as resources in a batch server:

- HiRDB
- Oracle
- SQL Server
- XDM/RD E2

(b) Database server machine

For details about the products required in a database server machine, see the explanation about the machine on which the resource manager operates, in the section *3.6.1 Configuration when using a local transaction*. However, you can use only the following databases as resources in a batch server:

- HiRDB
- Oracle
- SQL Server
- XDM/RD E2

4.2.2 System configuration of a system that uses the scheduling functionality of batch applications

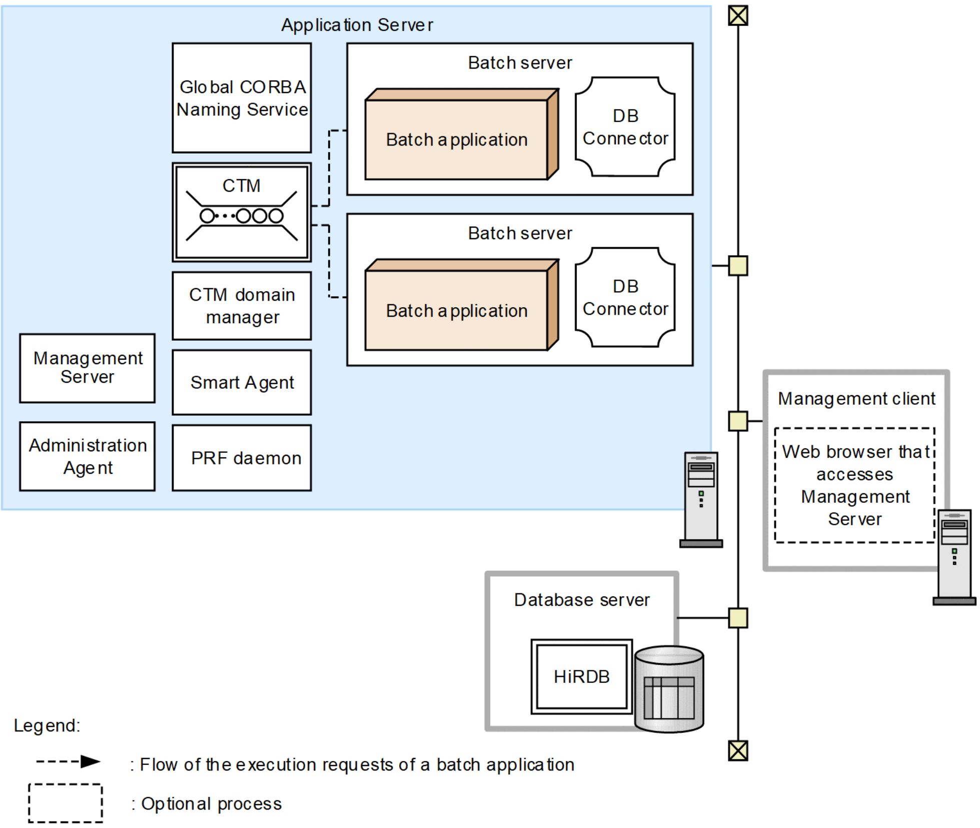
This subsection describes the system configuration that uses the scheduling functionality of the batch applications, for using a batch server.

(1) Features of system configuration

This is a system configuration for using a batch server and CTM. In this case, the batch server and CTM are deployed on the Application Server. The batch server and CTM is built and operated as a Web system (ctm-tier) of the Smart Composer functionality. You use batch execution commands for executing batch applications. A batch application executed by directly by command is scheduled using a CTM, and then distributed to the batch servers.

The following figure shows an example of the system configuration for using a CTM. In this example, two batch servers are deployed and the execution request of the batch application is scheduled using a CTM.

Figure 4–4: Example of system configuration for using CTM



For other legend items, see *3.2 Description of the system configuration*.

Features

You can execute multiple batch execution commands concurrently by scheduling the execution requests of a batch application using a CTM. Moreover, you must not use batch execution commands for specifying a batch server.

Flow of the process

A batch application executed by directly by the batch execution command is registered as an execution request of the batch application in the schedule queue of CTM. The execution request of the batch application in the schedule queue is distributed to an appropriate batch server using CTM. Note that the execution request of the batch application is retained (standby) in the schedule queue when the batch server, where the execution request is to be distributed, does not exist.

(2) Processes and software required on each machine

The software and processes required on each machine are as follows:

(a) Application server machine

You must install Application Server on an Application Server machine. Note that for the development environment, you must install Developer.

The following processes are to be started:

- Batch server
- PRF daemon
- Global CORBA Naming Service

- Group of CTM processes (CTM daemons and CTM regulators)
- CTM domain manager
- Smart Agent
- Management Server
- Administration Agent

When connecting to the database, you also require the software for connecting to the database to be used. For details about the products required for connecting to the database, see *3.6.1 Configuration when using a local transaction*. However, you can use only the following databases as resources in a batch server:

- HiRDB
- Oracle
- SQL Server
- XDM/RD E2

(b) Database server machine

For details about the products required in a database server machine, see the description about the machine on which the resource manager operates, in the subsection *3.6.1 Configuration when using a local transaction*. However, you can use only the following databases as resources in a batch server:

- HiRDB
- Oracle
- SQL Server
- XDM/RD E2

5 Estimating Resources to be Used (J2EE Application Execution Platform)

This chapter describes how to estimate the resources and virtual memory used in the systems for executing J2EE applications. Reference this chapter for calculating the disk and memory capacity required for operating a system.

When you set the number of file descriptors, you must specify the maximum number of processes that operate on the same host.

For details about estimating the resources and memory used for a batch application execution platform, see *6. Estimating Resources to be Used (Batch Application Execution Platform)*.

5.1 Resources used for each system configuration

Set up of the hosting environment components, such as the OS and database, might be required for operating the system. This section describes the resources used for each system configuration and the estimates for the required resources, as the resources required in a system differ as per the system configuration. The following table describes the reference locations for the resources used for each system configuration and resource estimates:

Table 5–1: References of resources used for each system configuration and resource estimates

Resources used for each system configuration	Reference
Resources used for deploying Web server and J2EE server on same machine	*5.1.1*
Resources used when Web server and J2EE server are deployed on different machines	*5.1.2*
Resources used by database	*5.1.3*
Resources used by Management Server	*5.1.4*
Resources used for CTM	*5.1.5*

For details about the estimation of resources used for each process, see *5.2 Resources used for each process*.

For details about the estimation of memory used for each process, see *5.3 Estimating memory used for each process*. For details about the disk occupancy, see the release notes of Application Server.

Tip

For a Windows system, among the items described in this section, see only *5.1.3 Resources used by the database*. Note that there are no particular restrictions on the number of processes available in the system, shared memory, number of file descriptors, and the number of threads available in a Windows system or process.

5.1.1 Resources used for deploying Web server and J2EE server on the same machine

This subsection describes the estimation of resources used for each OS when the Web server and J2EE server are deployed on the same machine.

Note that the *Example of option settings* in each table of estimation of resources used differs according to the OS and the kernel versions in use. Reference the manual of the OS being used and set up the estimates based on the estimation formula in the table. The settings are not required when the corresponding kernel parameter is not applicable in the OS being used.

(1) In AIX

The following table describes the estimation of resources used in AIX:

Table 5–2: Estimation of resources used (In AIX)

System resource	Parameter	Requirement	Example of option settings
Shared memory	--	`PrfTraceBufferSize`[#1] × `1,024 + 18,496` + *maximum-number-of-concurrently- processed-requests*[#2] × `14` × `1,024`	--
Number of processes	--	*maximum-number-of-concurrently-processed-requests*[#2] × `2 + 5`	--
Number of threads	--	*maximum-number-of-concurrently- processed-requests*[#2] × `2 + 41 +` *number-of-J2EE-server-threads*[#3]	--
Number of file descriptors	`nofiles`	*number-of-J2EE-server-file-descriptors*[#3] `+ 76` + *maximum-number-of-concurrently-processed-requests*[#2] × `4`	`/etc/security/limits`

Legend:

--: Not applicable.

#1

Specify the buffer memory size of performance tracer in the range of 512 kilobytes to 102,400 kilobytes. For details about `PrfTraceBufferSize`, see *4.12 Parameters applicable to logical performance tracers* in the *uCosminexus Application Server Definition Reference Guide*.

#2

Specify the maximum number of threads that can be processed concurrently in the Web server.

#3

For calculating the number of threads and file descriptors of the J2EE server, see *5.2.1 Estimating the resources used by J2EE server*.

(2) In Linux

The following table describes the estimation of resources used in Linux:

Table 5–3: Estimation of resources used (In Linux)

System resource	Parameter	Requirement	Example of option settings
Shared memory	`SHMMAX`	`PrfTraceBufferSize`[#1] × `1,024 + 18,496` + *maximum-number-of-concurrently- processed-requests*[#2] × `14` × `1,024`	`/proc/sys/kernel/shmmax`
Number of processes	`threads-max`[#3]	*maximum-number-of-concurrently-processed-requests*[#2] × `2 + 5`	`/proc/sys/kernel/threads-max`
Number of threads	`threads-max`[#3]	*maximum-number-of-concurrently-processed-requests*[#2] × `2 + 41 +` *number-of-J2EE-server- threads*[#4]	--
Number of file descriptors	`fs.file-max`	*number-of-J2EE-server-file-descriptors*[#4] `+ 76` + *maximum-number-of- concurrently-processed-requests*[#2] × `4`	`/proc/sys/fs/file-max`

Legend:

--: Not applicable.

#1

Specify the buffer memory size of performance tracer in the range of 512 kilobytes to 102,400 kilobytes. For details about `PrfTraceBufferSize`, see *4.12 Parameters applicable to logical performance tracers* in the *uCosminexus Application Server Definition Reference Guide*.

#2

Specify the maximum number of requests that can be processed concurrently in the Web server.

#3

Specify the total of number of processes and threads in the `threads-max` parameter.

#4

For calculating the number of threads and file descriptors of the J2EE server, see *5.2.1 Estimating the resources used by J2EE server*.

5.1.2 Resources used when Web server and J2EE server are deployed on different machines

This subsection describes the estimation of resources used for each OS when the Web server and J2EE server are deployed on different machines. When deploying the Web server and J2EE server on different machines, estimate the resources used respectively for the Web server machine and the Application Server machine.

Note that the *Example of option settings* in each table of estimation of resources used differs according to the OS and the kernel versions in use. Reference the manual of the OS being used and set up the estimates based on the estimation formula in the table. The settings are not required when the corresponding kernel parameter is not applicable in the OS being used.

(1) In AIX

This section describes the estimation of resources used by the Web server machine and the Application Server machine in AIX.

(a) Estimation of resources used by the Web server machine

The following table describes the estimation of resources used by the Web server machine:

Table 5–4: Estimation of resources used by the Web server machine (In AIX)

System resource	Parameter	Requirement	Example of option settings
Shared memory	--	`PrfTraceBufferSize`#1 × `1,024 + 18,496` + *maximum-number-of-concurrently- processed-requests*#2 × `14` × `1,024`	--
Number of processes	--	*maximum-number-of-concurrently- processed-requests*#2 × `2 + 4`	--
Number of threads	--	*maximum-number-of-concurrently-processed-requests*#2 × `2 + 35`	--
Number of file descriptors	`nofiles`	*maximum-number-of-concurrently-processed-requests*#2 × `4 + 75`	`/etc/security/limits`

Legend:

--: Not applicable.

#1

Specify the buffer memory size of performance tracer in the range of 512 kilobytes to 102,400 kilobytes. For details about `PrfTraceBufferSize`, see *4.12 Parameters applicable to logical performance tracers* in the *uCosminexus Application Server Definition Reference Guide*.

#2

Specify the maximum number of requests that can be processed concurrently in the Web server.

(b) Estimation of resources used by the Application Server machine

The following table describes the estimation of resources used by the Application Server machine:

Table 5–5: Estimation of resources used by the Application Server machine (In AIX)

System resource	Parameter	Requirement	Example of option settings
Shared memory	--	`PrfTraceBufferSize`[#1] × `1,024 + 18,496`	--
Number of processes	--	`4`	--
Number of threads	--	*number-of-J2EE-server-threads* [#2] `+ 34`	--
Number of file descriptors	`nofiles`	*number-of-J2EE-server-file-descriptors*[#2] `+ 43`	`/etc/security/limits`

Legend:

--: Not applicable.

#1

Specify the buffer memory size of performance tracer in the range of 512 kilobytes to 102,400 kilobytes. For details about `PrfTraceBufferSize`, see *4.12 Parameters applicable to logical performance tracers* in the *uCosminexus Application Server Definition Reference Guide*.

#2

For calculating the number of threads and file descriptors of the J2EE server, see *5.2.1 Estimating the resources used by J2EE server*.

(2) In Linux

This section describes the estimation of resources used by the Web server machine and the Application Server machine in Linux.

(a) Estimation of resources used by the Web server machine

The following table describes the estimation of resources used by the Web server machine:

Table 5–6: Estimation of resources used by the Web server machine (In Linux)

System resource	Parameter	Requirement	Example of option settings
Shared memory	`SHMMAX`	`PrfTraceBufferSize`[#1] × `1,024 + 18,496` `+` *maximum-number-of-concurrently- processed-requests*[#2] × `14` × `1,024`	`/proc/sys/kernel/shmmax`
Number of processes	`threads-max`[#3]	*maximum-number-of-concurrently-processed-requests*[#2] × `2 + 4`	`/proc/sys/kernel/threads-max`
Number of threads	`threads-max`[#3]	*maximum-number-of-concurrently-processed-requests*[#2] × `2 + 35`	--

System resource	Parameter	Requirement	Example of option settings
Number of file descriptors	`fs.file-max`	*maximum-number-of-concurrently-processed-requests*[#2] × `4 + 75`	`/proc/sys/fs/file-max`

Legend:

--: Not applicable.

#1

Specify the buffer memory size of performance tracer in the range of 512 kilobytes to 102,400 kilobytes. For details about `PrfTraceBufferSize`, see *4.12 Parameters applicable to logical performance tracers* in the *uCosminexus Application Server Definition Reference Guide*.

#2

Specify maximum number of requests that can be processed concurrently in the Web server.

#3

Specify total number of processes and threads in the `threads-max` parameter.

(b) Estimation of resources used by the Application Server machine

The following table describes the estimation of resources used by the Application Server machine:

Table 5–7: Estimation of resources used by the Application Server machine (In Linux)

System resource	Parameter	Requirement	Example of option settings
Shared memory	`SHMMAX`	`PrfTraceBufferSize`[#1] × `1,024 + 18,496`	`/proc/sys/kernel/shmmax`
Number of processes	`threads-max`[#2]	`4`	`/proc/sys/kernel/threads-max`
Number of threads	`threads-max`[#2]	*number-of-J2EE-server-threads*[#3] `+ 34`	--
Number of file descriptors	`fs.file-max`	*number-of-J2EE-server-file-descriptors*[#3] `+ 43`	`/proc/sys/fs/file-max`

Legend:

--: Not applicable.

#1

Specify the buffer memory size of performance tracer in the range of 512 kilobytes to 102,400 kilobytes. For details about `PrfTraceBufferSize`, see *4.12 Parameters applicable to logical performance tracers* in the *uCosminexus Application Server Definition Reference Guide*.

#2

Specify the total of number of processes and threads in the `threads-max` parameter.

#3

For calculating the number of threads and file descriptors of the J2EE server, see *5.2.1 Estimating the resources used by J2EE server*.

5.1.3 Resources used by the database

This subsection describes the estimation of resources used by the DBMS.

For details about the estimation of memory used for each process, see *5.3 Estimating memory used for each process*. For details about disk occupancy, see the Release notes of Application Server or Developer.

The following table describes the estimation of resources used by the DBMS:

Table 5–8: Estimation of resources used by DBMS

DBMS	Used resources	Requirement
HiRDB	Maximum number of concurrent connections (`pd_max_users`)	$\sum_{i=1}^{n^{\text{※1}}} (\text{Highest value of the connection pool of the resource adaptor i}^{\text{※2}} \times 2^{\text{※3}} + 1^{\text{※4}}) + \alpha^{\text{※5}}$
Oracle	Maximum number of concurrent connections (`SESSIONS`)	$\sum_{i=1}^{n^{\text{※1}}} (\text{Highest value of the connection pool of the resource adaptor i}^{\text{※2}} +1^{\text{※4}}) +\alpha^{\text{※5}}$

#1

n is the total number of resource adapters deployed on the J2EE server in the system.

#2

Specifies the value of the `MaxPoolSize` parameter in the Connector property file.

#3

If one of the following conditions is applicable, multiply by 2:

1. Use `XATransaction` in the transaction support level.
2. Use a connection# within the transaction that is managed by Application Server to access the database.
3. Before the transaction of step 2 concludes, use the connection# outside the transaction to access the database.

# This connection is the one that is acquired from the DB Connector in step 1 and is the same connection.

#4

Add 1 only for the resource adapters with `XATransaction` specified in the transaction support level.

#5

The maximum value of + α is the total of the maximum values of the connection pools of the DB Connector you are using.

This value indicates connections that might temporarily exceed the maximum value for the connection pool. The details are as follows:

- When using the functionality for detecting a connection failure

  When implementing the functionality to detect a connection failure, the unused connections removed from a connection pool are not counted as the connections within the connection pool. Therefore, the total number of connections in a connection pool and the unused connections removed from the connection pool might temporarily exceed the maximum number of connections in the connection pool.

- When using the `cjclearpool` command

  In the normal mode, the connections being used that are removed from a connection pool are not included in the connection count. Therefore, the total number of connections in the connection pool and the used connections removed from the connection pool might exceed the maximum value for the connection pool.

5.1.4 Resources used by Management Server

This subsection describes the estimation of resources used by Management Server for each OS.

Note that the *Example of option settings* in each table of estimation of resources used differs according to the OS and the kernel versions in use. Reference the manual of the OS being used and set up the estimates based on the estimation formula in the table. The settings are not required when the corresponding kernel parameter is not applicable in the OS being used.

(1) In AIX

The following table describes the estimation of resources used by Management Server:

Table 5–9: Estimation of resources used by Management Server (In AIX)

System resource	Parameter	Requirement	Example of option settings
Number of processes	--	`5 + A`[#]	--
Number of threads	--	`101 +` *number-of-logical-servers* `×4 + A`[#]	--
Number of file descriptors	`nofiles`	`218 +` *number-of-logical-servers* `×3`	`/etc/security/limits`

Legend:

--: Not applicable.

#

`A` represents the number of Management actions executed concurrently. Add the following value only if Management actions are executed when a Management event occurs:

- The total of the values specified for the `manager.mevent.send.max` keys in the property file for issuing Management events for each J2EE server.

(2) In Linux

The following table describes the estimation of resources used by Management Server:

Table 5–10: Estimation of resources used by Management Server (In Linux)

System resource	Parameter	Requirement	Example of option settings
Number of processes	`threads-max`[#1]	`5 + A`[#2]	`/proc/sys/kernel/threads-max`
Number of threads	`threads-max`[#1]	`101 +` *number-of-logical-servers* `×4 + A`[#2]	--
Number of file descriptors	`fs.files-max`	`218 +` *number-of-logical-servers* `× 3`	`/proc/sys/fs/file-max`

Legend:

--: Not applicable.

#1

Specify the total number of processes and threads in the `threads-max` parameter.

#2

`A` represents the number of Management actions executed concurrently. Add the following value only if Management actions are executed when a Management event occurs:

- The total of the values specified for the `manager.mevent.send.max` keys in the property file for issuing Management events for each J2EE server.

5.1.5 Resources used for CTM

The following subsection describes the estimation of resources used for CTM, for each OS.

Note that the *Example of option settings* in each table of estimation of resources used differs according to the OS and the kernel versions in use. Reference the manual of the OS being used and set up the estimates based on the estimation formula in the table. The settings are not required when the corresponding kernel parameter is not applicable in the OS being used.

(1) In AIX

The following table describes the estimation of resources used for CTM:

Table 5–11: Estimation of resources used for CTM (In AIX)

System resource	Parameter	Requirement	Example of option settings
Shared memory	--	`PrfTraceBufferSize`[#1] × 1,024 + 18,496 + *shared-memory-of-CTM-domain-manager*[#2] + *shared-memory-of-CTM-daemon*[#2]	--
Number of processes	--	7 + *number-of-J2EE-servers*[#3]	--
Number of threads	--	72 + (*number-of-J2EE-server-threads*[#4] + 7) × *number-of-J2EE-servers*[#3] + *number-of-threads-required-for-CTM- daemon*[#5]	--
Number of file descriptors	`nofiles`	88 + (*number-of-J2EE-server-file-descriptors*[#4] + 6) × *number-of-J2EE-servers*[#3] + *number-of-file-descriptors-required-for-CTM- daemon*[#5]	`/etc/security/limits`

Legend:

--: Not applicable.

#1

Specify the buffer memory size of the performance tracer in the range from 512 kilobytes to 102,400 kilobytes. For details about `PrfTraceBufferSize`, see *4.12 Parameters applicable to logical performance tracers* in the *uCosminexus Application Server Definition Reference Guide*.

#2

For calculating the values, see *5.1.5(1)(a) Formula for calculating the file size for a shared memory*.

#3

Indicates the value specified in the `<j2ee-server-count>` tag of Easy Setup definition file.

#4

For calculating the number of threads and file descriptors of the J2EE server, see *5.2.1 Estimating the resources used by J2EE server*.

#5

For calculating the number of threads and file descriptors required in the CTM daemon, see *5.1.5(1)(b) Formula for calculating the number of threads and file descriptors required in the CTM daemon*.

(a) Formula for calculating the file size for a shared memory

You must calculate the shared memory of the CTM domain manager and CTM daemon to calculate file size of shared memory. The respective formula for calculations is as follows:

Use the following value for variable values in the calculations formula. For details about the parameters starting with `ctm.`, see *4.14 Parameters applicable to the logical CTM* in the *uCosminexus Application Server Definition Reference Guide*.

Values used in the formula for calculations:

-CTMMaxCTM: 64

-CTMQueueCount: ctm.QueueCount

-CTMClientConnectCount: 256

-CTMServerConnectCount: ctm.ServerConnectCount

-CTMEntryCount: -CTMClientConnectCount + -CTMServerConnectCount

-CTMServerCacheSize: ctm.ServerCacheSize

-CTMQueueRegistCount: ctm.QueueRegistCount

-CTMDispatchParallelCount: ctm.DispatchParallelCount

- Formula for calculating file size for shared memory of CTM domain manager

  The formula for calculating the file size for the shared memory of the CTM domain manager is as follows:

  File-size-for-shared-memory (Unit: Bytes) =

  1,018,320 + (2,362×-CTMMaxCTM *specified-value*)

- Formula for calculating file size for shared memory of CTM daemon

  For the CTM daemon, you must secure files for shared memory of fixed length and files for shared memory of variable lengths in each CTM daemon. The formula for the respective calculations is as follows:

  File-size-for-shared-memory-of-fixed-length (Unit: Bytes) =

  551,840+(1,208×-CTMQueueCount *specified-value*)

  File-size-for-shared-memory-of-variable-length (Unit: Bytes) =

  1,027,008
  +(928×-CTMClientConnectCount *specified-value*)
  +(256×-CTMServerConnectCount *specified-value*)
  +(512×-CTMEntryCount *specified-value*)
  +(1,024×-CTMServerCacheSize *specified-value*)
  +(512×-CTMQueueCount *specified-value*)
  +(544×-CTMQueueCount *specified-value*×-CTMQueueRegistCount *specified-value*)
  +(512×-CTMDispatchParallelCount *specified-value*)

(b) Formula for calculating the number of threads and file descriptors required in the CTM daemon

For calculating the number of threads and file descriptors, you must calculate the number of threads and file descriptors required in the CTM daemon. The formula for the respective calculations is as follows:

- Formula for calculating number of threads in the CTM daemon

  Maximum-value =

  (*A*×4+*B*×3+*C*×2+*D*×*E*+*F*+*G*+ 32) / 0.8

  Legend:

  A: -CTMMaxCTM value (Value specified in ctmdmd to which the ctmd belongs)

  B: -CTMClientConnectCount value

  C: -CTMServerConnectCount value

  D: -CTMQueueCount value

  E: -CTMQueueRegistCount value

F: `-CTMDispatchParallelCount` value

G: Total number of EJB clients issuing Create

- Formula for calculating number of file descriptors required in the CTM daemon

  Maximum-value =

  $(A\times 2+B\times 4+C\times 2+D\times E+F\times \textit{number-of-EJB-interfaces} +G+100)\ /\ 0.8$

  Legend:

  A:`-CTMMaxCTM` value (Value specified in `ctmdmd` to which the `ctmd` belongs)

  B: `-CTMClientConnectCount` value

  C: `-CTMServerConnectCount` value

  D: `-CTMQueueCount` value

  E: `-CTMQueueRegistCount` value

  F: `-CTMDispatchParallelCount` value

  G: Total number of EJB clients issuing Create

(2) In Linux

The following table describes the estimation of resources used for CTM:

Table 5–12: Estimation of resources used for CTM (In Linux)

System resource	Parameter	Requirement	Example of option settings
Shared memory	`SHMMAX`	`PrfTraceBufferSize`[#1] × 1,024 + 18,496 + *shared-memory-of-CTM-domain-manager*[#2] + *shared-memory-of-CTM-daemon*[#2]	`/proc/sys/kernel/shmmax`
Number of processes	`threads-max`	7 + *number-of-J2EE-servers*[#3]	`/proc/sys/kernel/threads-max`
Number of threads	`threads-max`	72 + (*number-of-J2EE-server-threads*[#4] + 7) × *number-of-J2EE-servers*[#3] + *number-of-threads-required-in-CTM-daemon*[#5]	--
Number of file descriptors	`fs.file-max`	88 + (*number-of-J2EE-server-file-descriptors*[#4] + 6) × *number-of-J2EE-servers*[#3] + *number-of-file-descriptors-required-in-CTM-daemon*[#5]	`/proc/sys/fs/file-max`

Legend:

--: Not applicable.

#1

Specify the buffer memory size of the performance tracer in the range from 512 kilobytes to 102,400 kilobytes. For details about `PrfTraceBufferSize`, see *4.12 Parameters applicable to logical performance tracers* in the *uCosminexus Application Server Definition Reference Guide*.

#2

For calculating the values, see *5.1.5(1)(a) Formula for calculating the file size for a shared memory*.

#3

Indicates the value specified in the `<j2ee-server-count>` tag of the Easy Setup definition file.

#4

For calculating the number of threads and file descriptors of the J2EE server, see *5.2.1 Estimating the resources used by J2EE server*.

#5

For calculating the number of threads and file descriptors required in the CTM daemon, see *5.1.5(1)(b) Formula for calculating the number of threads and file descriptors required in the CTM daemon*.

5.2 Resources used for each process

This section describes the estimation of the required amount of resources used in each process of Application Server.

For details about the resources used by the management server, see *5.1.4 Resources used by Management Server*.

5.2.1 Estimating the resources used by J2EE server

This subsection describes how to estimate the number of threads and file descriptors of the J2EE server process.

(1) Number of threads

The formula for calculating number of threads is as follows. The total of (a) and (b) is the number of threads used by J2EE server.

(a) Basic number of threads

The formula is as follows:

- For a J2EE server
  Maximum-number-of-threads = `76+A+B+C+D+E+F+G+H+I+J+K+L+M+N+O+P+Q+R+S`
- For a batch server
  Maximum-number-of-threads = `73+D+E+F+G+H+I+O+P`

Legend:

- *A*: Total number of deployed Entity Beans
- *B*: Maximum number of instances of Message-driven Bean when using Message-driven Bean (Total of instances when multiple Message-Driven Beans exist) #

  #: Value of `<pooled-instance><maximum>` in the Message-Driven Bean property file
- *C*: *Maximum-number-of-EJB-clients-performing-remote-invocation* ×2+*Total-number-of-maximum-concurrent-requests-of-each-EJB-client* +1
- *D*: Number of threads of CORBA Naming Service (=*number-of-connections-between-client-and-CORBA-Naming-Service* ×2+*number-of-requests-received-concurrently* +*number-of-threads-generated-during-initialization* (if the value of `vbroker.agent.enableLocator` is `true`, then `6`, if the value is `false`, then `4`) +1)

  However, added only when CORBA Naming Service is invoked as in-process (Specify `inprocess` in the `ejbserver.naming.startupMode` key of `usrconf.properties`).
- *E*: Maximum number of database connections used concurrently

  If the connection pooling functionality is used, the maximum number of connection pools (value of `MaxPoolSize` specified in the Hitachi Connector Property file. Total of connection pools if there are multiple resource adapters) is used as the value.

  If the connection pooling functionality is not used, the value is obtained from the maximum number of concurrent requests or the number of connections used for each request (if one connection is used for one request, the maximum number of concurrent requests is used as the value).
- *F*: Maximum number of concurrently executed transactions, if the JTA transaction is used (one thread is used for each transaction in which a transaction timeout occurs. In the case of one request in one transaction, the maximum number of concurrent requests is used as the value)

- *G*: *Maximum-number-of-connection-pools*[#] (total of connection pools if there are multiple resource adapters) × 2

  #: Value of `MaxPoolSize` specified in the Hitachi Connector Property file
- *H*: Number of resource adapters using the connection pooling functionality
- *I*: Number of threads used for the conclusion and recovery processing of a global transaction (add 16 if a global transaction is used)
- *J*: Number of deployed Web applications
- *K*: Number of threads used in the automatic reload functionality of a J2EE application

  Specify one of the following values:
  - **For** `ejbserver.deploy.context.reload_scope=app`**, and** `ejbserver.deploy.context.check_interval=1` **or more**

    Number-of-running-J2EE-applications-in-exploded-archive-format + *Number-of-WARs-included-in-running-J2EE-applications-in-exploded-archive-format* × 2
  - **For** `ejbserver.deploy.context.reload_scope=web`**, and** `ejbserver.deploy.context.check_interval=1` **or more**

    Number-of-WARs-included-in-running-J2EE-applications-in-exploded-archive-format × 2
  - **For** `ejbserver.deploy.context.reload_scope=jsp`**, and** `ejbserver.deploy.context.check_interval=1` **or more**

    Number-of-WARs-included-in-running-J2EE-applications-in-exploded-archive-format
- *L*: Number of threads of the TP1 inbound adapters calculated using the following formula:
  (4 + *number-of-threads-specified-in-TP1-inbound-adapter-property-rpc_max_thread_count* + *number-of-threads-specified-in-TP1-inbound-adapter-property-trn_max_thread_count* + *total-number-of-deployed-Message-driven-Bean-(services)-integrated-with-TP1-inbound-adapters* + *number-of-threads-specified-in-TP1-inbound-adapter-property-MaxTPoolSize*)

  Add only when using the TP1 inbound integrated functionality. This thread count can be controlled by the Hitachi Connector Property file. The number 4 added at the beginning is the number of threads used internally for the TP1 inbound integrated functionality.
- *M*: Sum of the maximum number of thread pools used for invoking the asynchronous Session Beans (Value of `<cosminexus-app><ejb-async-props><max-thread-pool-size>` in `cosminexus.xml`)
- *N*: *Sum-of-the-number-of-J2EE-applications-containing-asynchronous-Session-Beans* × 2
- *O*: If settings are specified to start the thread that manages the reply receiving thread (`vbroker.ce.iiop.ccm.htc.threadStarter=true`), add 5.
- *P*: Add the following value if settings are specified to control the closing of connections when a timeout occurs (`vbroker.ce.iiop.ccm.htc.readerPerConnection=true`):

  (*Number-of-J2EE-servers-with-remote-invocation-destination-EJBs* + 1) × 2 If CTM is used, add the following values in addition to the above:
  - **For scheduling of J2EE applications**

    Number-of-running-J2EE-applications + 1
  - **For scheduling of Stateless Session Beans**

    Number-of-Stateless-Session-Beans-to-be-scheduled + 1
- *Q*: Maximum number of threads of the NIO HTTP server (the value of `webserver.connector.nio_http.max_threads`)
- *R*: Maximum number of threads used by Java Batch

  Add only when running an application that uses Java Batch.

Specify the value of `ejbserver.javaee.batch.executorService.`*JNDI-name*`.maxThreads`. If multiple JNDI names are defined, specify the total of the values for each JNDI name.

- *S*: Maximum number of threads used by Concurrency Utilities for Java EE

  Add only when running an application that uses Concurrency Utilities for Java EE.

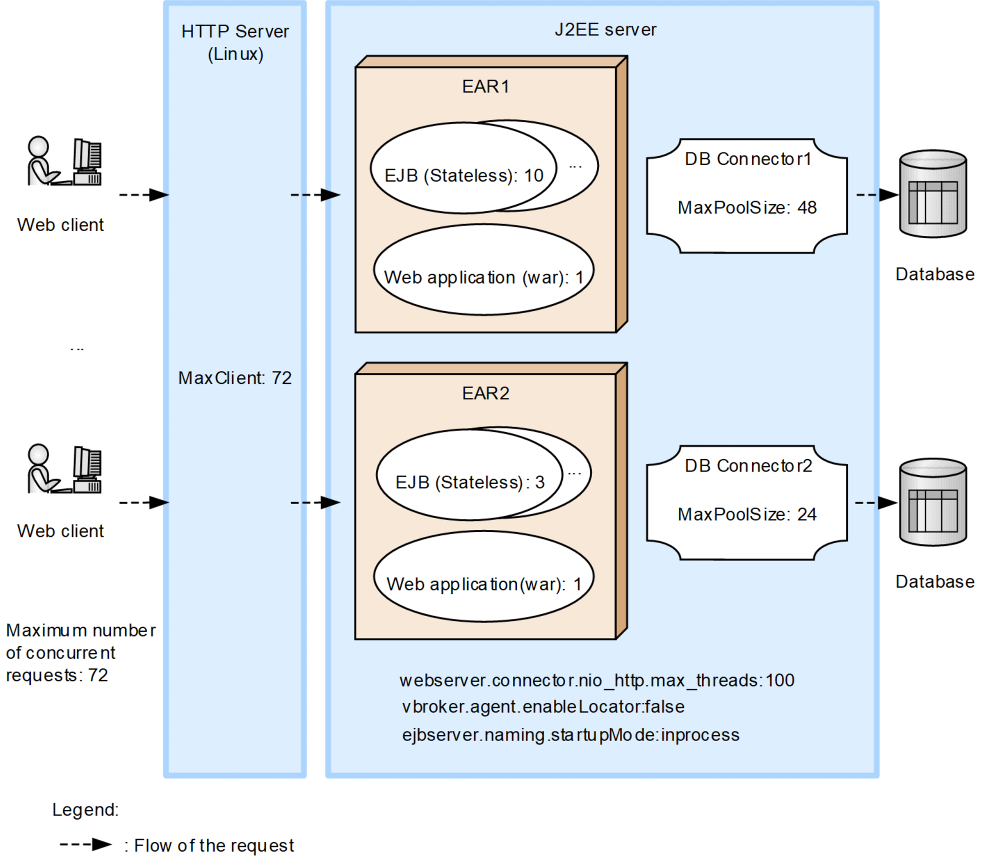
  This value is the total of the following values. If multiple JNDI names are defined, specify the total of the values for each JNDI name.

  - Maximum number of `ManagedExecutorService` threads

    (The value of `ejbserver.javaee.concurrent.managedExecutorService.`*JNDI-name*`.maxPoolSize`)
  - The maximum number of tasks executed concurrently by `ManagedScheduledExecutorService`
  - The maximum number of threads generated concurrently by `ManagedThreadFactory`

The following figure shows the example estimation when CORBA Naming Service is started as in-process and Cosminexus HTTP Server is used.

Figure 5–1: Example estimation of the number of threads when **Cosminexus HTTP Server** is used



webserver.connector.nio_http.max_threads:100
vbroker.agent.enableLocator:false
ejbserver.naming.startupMode:inprocess

The following is an example of estimating the number of threads for using Cosminexus HTTP Server shown in the figure.

Formula for calculating the number of threads, when CORBA Naming Service is started as in-process and Cosminexus HTTP Server is used

```
Maximum-number-of-threads = 76+A+B+C+D+E+F+G+H+I+J+K+L+M+N+O+P+Q+R+S
```

The results calculated using the above formula and the contents that are set up are as follows:

Maximum-number-of-threads = `76+0+0+1+5+72+72+144+2+0+2+0+0+0+0+0+0+100+0+0=474`

Table 5–13: Contents estimated for the number of threads when Cosminexus HTTP Server is used (Example)

Setup item	Set value	Explanation
A	0	Sets 0 because Entity Bean is not used.
B	0	Sets 0 because Message-driven Bean is not used.
C	1	A remote invocation is not performed. Also, if the EJB is invoked locally from a Web application, the threads are not generated for executing the EJBs.
D	4+1	Such as C, the EJB client-related value is 0.
E	48+24	--
F	72	Sets the value of `MaxClient`, which is the maximum number of concurrent requests, if there is one request in one transaction.
G	(48+24) × 2	--
H	2	--
I	0	Sets 0 because the global transaction is not used.
J	2	--
K	0	Sets 0 because the application is not in the exploded archive format.
L	0	Sets 0 because the TP1 inbound adapter is not used.
M	0	Sets 0 because the asynchronous Session Bean is not used.
N	0	Sets 0 because the asynchronous Session Bean is not used.
O	0	Sets 0 because the settings for receiving the response message using a dedicated thread are not specified.
P	0	Sets 0 because the settings for controlling the closing of connections are not specified.
Q	100	--
R	0	Sets 0 because Java Batch is not used.
S	0	Sets 0 because Concurrency Utilities for Java EE is not used.

(b) Number of threads used according to JavaVM option specifications

According to JavaVM option specifications, calculate the maximum number of threads using the following formula. Add A only when `-XX:+UseG1GC` option is specified, and add B only when `-XX:+HitachiUseExplicitMemory` option is specified.

Maximum-number-of-threads = A + B

Legend:

- *A*: Number of threads used by G1 GC *(Value specified in* `-XX:ParallelGCThreads` option. When this option is not specified, the default value of `-XX:ParallelGCThreads` option based on the number of logical CPUs. Note that the value is determined by the number of logical CPUs that exist when starting the J2EE server. Hence, the number of threads does not change even if the number of logical CPUs changes after the server is started.)

- *B*: Number of threads used by the Explicit Memory Management functionality (The number of logical CPUs. However, this number is 8 when the number of logical processors is 8 or more. This number is determined by the number of logical CPUs that exist when starting the J2EE server. Hence, the number of threads does not change even if the number of logical CPUs changes after the server is started.)

For the JavaVM options, see the following sections in the *uCosminexus Application Server Definition Reference Guide*:

- *14.5 Java HotSpot VM options that can be specified in Cosminexus*
- *-XX:[+|−]HitachiUseExplicitMemory (Explicit Memory Management functionality option)*

(2) Number of file descriptors

The formula for calculating number of file descriptors is as follows:

- For a J2EE server
  maximum-number-of-file-descriptors `=` `(153+`*A*`+`*B*`×3+`*C*`+`*D*`+`*E*`+`*F*`×2+`*G*`+`*H*`+`*I*`+`*J*`+`*K*`) / 0.8`
- For a batch server
  maximum-number-of-file-descriptors `=` `(149+`*A*`+`*F*`×2+`*G*`) / 0.8`

Legend:

- *A*: Number of database connections
- *B*: Number of EJB client processes
- *C*: Total number of connections to J2EE servers
- *D*: Number of file descriptors used by the TP1 inbound adapters calculated using the following formula (add only when you use the TP1 inbound integrated functionality. Also, the fixed value calculated at the beginning shows the number of file descriptors in the TP1 inbound integrated functionality)
  `8 +` *value-specified-in-TP1-inbound-adapter-property* `max_connections`
  +*value-specified-in-TP1-inbound-adapter-property* `trn_max_connections`
  +*total-of-values-specified-in* `<pooled-instance><maximum>` *of-each-MDB (services)-Message-driven-Bean-attribute file* × 2
  +*number-of-threads-specified-in-TP1-inbound-adapter-property* `rpc_max_thread_count` × 2
  +*number-of-threads-specified-in-TP1-inbound-adapter-property* `trn_max_thread_count`× 2
- *E*: Number of JAR files included in J2EE application
- *F*: Number of resource adapters
- *G*: Number of JAR files specified in `add.class.path` key of `usrconf.cfg`
- *H*: Number of JAR files included in WEB-INF/lib of a Web application
- *I*: Maximum number of NIO HTTP server threads
- *J*: Maximum number of Java Batch threads
- *K*: Maximum number of Concurrency Utilities for Java EE threads

(3) Estimating number of threads of CORBA Naming Service (When invoking as in-process)

This section describes the estimation of number of threads of CORBA Naming Service generated on J2EE server when invoking the CORBA Naming Service as in-process while invoking the J2EE server.

Estimate the number of threads of CORBA Naming Service as follows when invoking the CORBA Naming Service as in-process:

Total-number-of-threads = number-of-threads + worker-threads-generated-during-initialization

(a) Number of threads generated during initialization

The number of threads generated during initialization is 6, when the value of `vbroker.agent.enableLocator` key of `usrconf.properties` is `true` and 4, when the value is `false`. The `vbroker.agent.enableLocator` key is setup as `true` automatically when the CTM integration functionality is enabled (`true` is specified in `ejbserver.ctm.enabled` key)

(b) Number of worker threads

The number of worker threads is the total of *number-of-requests-received-concurrently* + `1` and *number-of-connections-between-client-and-CORBA-Naming-Service* × `2`.

The keys related to worker threads are as follows:

- `vbroker.se.iiop_tp.scm.iiop_tp.dispatcher.threadMax`
- `vbroker.se.iiop_tp.scm.iiop_tp.dispatcher.threadMin`
- `vbroker.se.iiop_tp.scm.iiop_tp.dispatcher.threadMaxIdle`

These keys are specified as value of `ejbserver.naming.exec.args` key of `usrconf.properties`. For details on the keys, see the manuals *Borland(R) Enterprise Server VisiBroker(R) Developers Guide* and *Borland(R) Enterprise Server VisiBroker(R) Programmers Reference*.

The number of worker threads when maximum value is specified in the `vbroker.se.iiop_tp.scm.iiop_tp.dispatcher.threadMax` key is the total of *maximum-value-specified-in-this-key* and *number-of-connections-between-client-and-CORBA-Naming-Service*.

However, when minimum value of worker threads is specified in the `vbroker.se.iiop_tp.scm.iiop_tp.dispatcher.threadMin` key, and when the total number of worker threads is not within that minimum value, consider the number of worker threads equal to the minimum value.

When the maximum value is not specified, the worker threads increase in multiples.
However, the worker threads lapse when they exceed the time specified in the `vbroker.se.iiop_tp.scm.iiop_tp.dispatcher.threadMaxIdle` key (Default value is 300 seconds) from the time they are idle, resulting in reduction in the number of threads with the reduction in load.

When the maximum value of number of worker threads is specified and when the number of threads is the same as the maximum number of worker threads, do not perform error handling for the requests received from this point of time but continue the process as follows:

- Continue the process for received requests.
- Do not *read()* the new requests from the socket and retain these requests in the receive buffer of TCP and send buffer at the client side. When the TCP buffer is full, the requests would wait to be sent from the client side.

When the worker thread being processed is NULL (Responded), the next request reception is processed.

5.2.2 Estimating the resources used by Administration Agent

This subsection describes the estimation of resources used by Administration Agent for each OS.

(1) In Windows

The formula for calculating number of threads when using Windows is as follows:

- **Formula for calculating number of threads**
  Number-of-threads-used = `30+7`×*number-of-logical-servers*#
  #: Consider the number of logical servers as 2 when calculating logical CTM.
  Legend:
  - `30`: Number of threads used by Administration Agent
  - `7`: Number of threads used by Administration Agent for one logical server

The formula for calculating the number of threads when the logical server is in running status is as follows:

- **Formula for calculating number of threads at usual time**
  Number-of-threads-used = `30+5`×*number-of-logical-servers*#
  #: Consider the number of logical servers as 2 when calculating logical CTM.
  Legend:
  - `30`: Number of threads used by Administration Agent
  - `5`: Number of threads used by Administration Agent for one logical server

(2) In UNIX

This section describes the formula for calculating the number of threads and file descriptors required when using UNIX.

(a) Number of threads

The formula for calculating number of threads is as follows:

- **Formula for calculating number of threads**
  Number-of-threads-used = `30+5`×*number-of-logical-servers*#
  #: Consider the number of logical servers as 2 when calculating logical CTM.
  Legend:
  - `30`: Number of threads used by Administration Agent
  - `5`: Number of threads used by Administration Agent for one logical server

The formula for calculating the number of threads when the logical server is in running status is as follows:

- **Formula for calculating number of threads at usual time**
  Number-of-threads-used = `30+5`×*number-of--logical-servers*#
  #: Consider the number of logical servers as 2 when calculating logical CTM.
  Legend:
  - `30`: Number of threads used by Administration Agent
  - `5`: Number of threads used by Administration Agent for one logical server

(b) Number of file descriptors

The formula for calculating the number of file descriptors is as follows:

- **Formula for calculating number of file descriptors**
  Number-of-file-descriptors-used = 20+*number-of-processes-configuring-the-logical-server*×6
  Legend:
  - 20: Number of file descriptors used by Administration Agent
  - 6: Number of file descriptors used by Administration Agent for one process each for configuring logical server

The formula for calculating number of file descriptors when the logical server is in running status is as follows:

- **Formula for calculating number of file descriptors at usual time**
  Number-of-file-descriptors-used = 20+*number-of-processes-configuring-the-logical-server*×3
  Legend:
  - 20: Number of file descriptors used by Administration Agent
  - 3: Number of file descriptors used by Administration Agent for one process each for configuring logical server

5.2.3 Estimating the resources used by performance tracer

This subsection describes the estimation of resources used by performance tracer for each OS.

(1) In Windows

This section describes the estimation of resources used by performance tracer when using Windows.

(a) File size for shared memory

Calculate the file size (Unit: Bytes) for shared memory used by performance tracer for each PRF daemon. The formula for calculating is as follows:

- **Formula for calculating shared memory for each PRF daemon**
  File-size-for-shared-memory = -PrfTraceBufferSize-*specified-value* × 1,024+18,496

(b) Disk occupancy of %PRFSPOOL%

Formula for calculating disk occupancy of %PRFSPOOL% is as follows:

- **Formula for calculating disk occupancy of %PRFSPOOL%**
  Disk-occupancy = 2.0 MB
  + {(-PrfTraceBufferSize-*specified-value* + 20KB) ×5
  + -PrfTraceFileSize-*specified-value* × -PrfTraceCount-*specified-value* × *r*} × *n*
  + 224 KB× (256+*m*)
  + 224 KB× (64 + *p*)
  Legend:
  - *n*: Number of PRF daemons
  - *m*: Number of running performance trace output processes and number of performance trace output processes that did not terminate normally

In a performance tracer, the internal trace is output to a file as the maintenance information for each performance trace output process. This file is created when a process starts, but the file remains if the process has abnormal termination. The file is deleted when a PRF daemon is invoked and every 24 hours after the PRF daemon is invoked, but 256 files remain without being deleted. Therefore, the maximum number of files is "`256 +` *number-of-performance-trace-output-processes-executed-in-24-hours*".

- *p*: Number of commands and daemon processes used in the running performance analysis trace

  In a performance tracer, the internal trace is output to a file as the maintenance information for each command and daemon used in the performance analysis trace. This file is created when a process starts. The file is deleted when a PRF daemon is invoked and every 24 hours after the PRF daemon is invoked, but 64 files remain without being deleted. Therefore, the maximum number of files is "`64 +` *number-of-command-and-daemon-processes-used-in-performance-analysis-trace-executed-in-24-hours*".

- *r*: Backup coefficient

  This value calculates the PRF trace backup. If you specify `-PrfNoBackUp 0` in the invocation option for the `cprfstart` command, the backup coefficient is `2`, and in all other cases, the backup coefficient is `1`.

The above mentioned disk capacity is a rough indicator. As a result, create the `%PRFSPOOL%` with enough excess space.

(2) In AIX

You must consider the following values and set up the kernel parameter to use the performance tracer when using AIX. If the settings are not done properly, the processes of performance tracer cannot be invoked or while running the processes they might terminate abnormally due to insufficient resources. For the kernel parameter settings, see the manual of the OS being used.

(a) File size for shared memory

Calculate the file size for the shared memory (Unit: Bytes) used by performance tracer for each PRF daemon. The formula for calculating is as follows:

- **Formula for calculating shared memory for each PRF daemon**

  File-size-for-shared-memory = `-PrfTraceBufferSize`-*specified-value* × `1,024+18,496`

Set up the file size for shared memory using the environment variable *EXTSHM*. Set up in such a way so that shared memory allocated is greater than the value calculated in the formula for calculations.

(b) Number of file descriptors

Set up the number of file descriptors in `nofiles` of the `/etc/security/limits` file. Set up the number of file descriptors used when invoking the PRF daemon as 32 and above.

(c) Disk occupancy of $PRFSPOOL

The formula for calculating the disk occupancy of `$PRFSPOOL` is as follows:

- **Formula for calculating disk occupancy of $PRFSPOOL**

  Disk-occupancy `= 2.0 MB`

  `+ {(-PrfTraceBufferSize-`*specified-value* `+ 20KB) ×5`

  `+ -PrfTraceFileSize-`*specified-value* × `-PrfTraceCount-`*specified-value* × *r*`} ×` *n*

  `+ 224 KB× (256+`*m*`)`

  `+ 224 KB× (64 +` *p*`)`

  Legend:

- *n*: Number of PRF daemons
- *m*: Number of running performance trace output processes and number of performance trace output processes that did not terminate normally

  In a performance tracer, the internal trace is output to a file as the maintenance information for each performance trace output process. This file is created when a process starts, but the file remains if the process has abnormal termination. The file is deleted when a PRF daemon is invoked and every 24 hours after the PRF daemon is invoked, but 256 files remain without being deleted. Therefore, the maximum number of files is "`256 +` *number-of-performance-trace-output-processes-executed-in-24-hours*".
- *p*: Number of commands and daemon processes used in the running performance analysis trace

  In a performance tracer, the internal trace is output to a file as the maintenance information for each command and daemon used in the performance analysis trace. This file is created when a process starts. The file is deleted when the PRF daemon is invoked and every 24 hours after the PRF daemon is invoked, but 64 files remain without being deleted. Therefore, the maximum number of files is "`64 +` *number-of-command-and-daemon-processes-used-in-performance-analysis-trace-executed-in-24-hours*".
- *r*: Backup coefficient

  This value calculates the PRF trace backup. If you specify `-PrfNoBackUp 0` in the invocation option for the `cprfstart` command, the backup coefficient is `2`, and in all other cases, the backup coefficient is `1`.

The above mentioned disk capacity is a rough indicator. As a result, create the `$PRFSPOOL` with enough excess space.

(3) In Linux

You must consider the following values and set up the kernel parameter to use the performance tracer when using Linux. If the settings are not done properly, the processes of performance tracer cannot be invoked or while running the processes they might as well terminate abnormally due to insufficient resources. For the kernel parameter settings, see the manual of the OS being used.

(a) File size for shared memory

Calculate the file size for shared memory (Unit: Bytes) used by performance tracer for each PRF daemon. The formula for calculating is as follows:

- **Formula for calculating shared memory for each PRF daemon**

  File-size-for-shared-memory `= -PrfTraceBufferSize`-*specified-value* × `1,024+18,496`

Set up the file size for shared memory in `kernel.shmmax` of `/etc/sysctl.conf` file. Set up in such a way so that shared memory allocated is greater than the value calculated in formula for calculations.

(b) Number of file descriptors

Set up the number of file descriptors in `nofiles` of `/etc/security/limits.conf` file. Set up the number of file descriptors used when invoking the PRF daemon as 32 and above.

(c) Disk occupancy of $PRFSPOOL

The formula for calculating the disk occupancy of `$PRFSPOOL` is same as AIX. Reference the formula for calculations of AIX.

5.2.4 Estimating the resources used by CTM

This subsection describes the estimation of resources used by CTM for each OS. You cannot use CTM in batch application execution environment.

(1) In Windows

This section describes the estimation of resources used by CTM in Windows.

(a) File size for shared memory

This section describes the formula for calculating file size for shared memory (Unit: Bytes) using CTM. You must secure more than one shared memory for CTM daemon. There are files for shared memory with fixed length and variable length in the CTM daemon unit. The formula for calculating the respective file size is as follows:

- **Formula for calculating file size for shared memory of CTM domain manager**
  File-size-for-shared-memory = 1,018,320 + (2,362×-CTMMaxCTM-*specified-value*)
- **Formula for calculating file size for shared memory of CTM daemon**
  The files for shared memory with fixed length and variable length must be secured in the CTM daemon unit for CTM daemon.
  - *File-size-for-shared-memory-with-fixed-length* =
    551,840+(1,208×-CTMQueueCount-*specified-value*)
  - *File-size-for-shared-memory-with-variable-length* =
    1,027,008
    + (928×-CTMClientConnectCount-*specified-value*)
    + (256×-CTMServerConnectCount-*specified-value*)
    + (512×-CTMEntryCount-*specified-value*)
    + (1,024×-CTMServerCacheSize-*specified-value*)
    + (512×-CTMQueueCount-*specified-value*)
    + (544×-CTMQueueCount-*specified-value*)× -CTMQueueRegistCount-*specified-value*)
    + (512×-CTMDispatchParallelCount-*specified-value*)

(b) Statistics information file size

The formula for calculating statistics information file size (Unit: Bytes) is as follows:

- **Formula for calculating statistics information file size required from online start to end**
  File-size = *A* + *B*
  Legend:
  - *A*: (*number-of-requests-executed-from-online-start-to-end*) × (*information-output-for-one-request*)
  - *B*: (*time-from-online-start-to-end*(*minutes*) / *acquired-interval-specified-in-*ctmstsstart -CTMInterval) × (*volume-of-statistics-information-output-once-for-CTM-node-unit, queue-unit*)

The formula for calculating volume of information output by one request and volume of statistics information output once for CTM node unit, queue node unit is as follows:

- **Formula for calculating volume of information output by one request**
  Volume-of-information =(↑ (80 + *A* + *B* + *C* + *D* + 63)/ 64↑ × 64) × 3

Legend:

- *A*: Length of domain of CTM domain name that manages application executing requests
- *B*: Length of CTM identifier of CTM daemon that manages application executing requests
- *C*: Length of queue name
- *D*: Length of operation name

- **Formula for calculating volume of statistics information output once for CTM node unit, queue unit**
  Volume-of-statistics-information = ↑(2,144 + 344 ×*number-of-queues* + 63) / 64↑×64

(c) Disk occupancy of %CTMSPOOL%

The formula for calculating disk occupancy of `%CTMSPOOL%` is as follows:

- **Formula for calculating disk occupancy of %CTMSPOOL%**
  Disk-occupancy = 7.0MB
  + (18.5MB + 1.0MB × -CTMLogFileSize-*specified-value* × -CTMLogFileCount-*specified-value*) × *n*
  +(1KB + 0.5KB × *k*) × (*m* + *l*)
  +1KB × m × *j*
  +224KB × *p*
  +1,120KB × (64 + *q*)
  +*file-size-for-shared-memory-of-CTM-domain-manager* × 5
  +*file-size-for-shared-memory-of-CTM-daemon* × 5 × *n*
  +*core-size-of-CTM-domain-manager*
  +*core-size-of-CTM-daemon* × *n*
  +*core-size-of-CTM-regulator* × 3 × *m*
  +*core-size-of-OTM-gateway* × 3 × *l*
  + (-CTMStatsFileSize-*specified-value*× -CTMStatsFileCount-*specified-value*) × *n*

  Legend:

  - *j*: Total of EJBs
  - *k*: Number of clients that you can connect to CTM regulator and OTM gateway (`-CTMClientConnectCount` option specified value)
  - *l*: Total of OTM gateways
  - *m*: Total of CTM regulators
  - *n*: Total of CTM daemons
  - *p*: Number of processes of user applications being invoked and number of processes of user applications that did not terminate normally

    In CTM, the internal trace is output to a file as maintenance information for each user application. This file is created while invoking the process. The file for internal trace output is created while invoking the process and remains as it is when the process does not terminate normally. The file deletion process is executed while invoking the CTM domain manager and after 24 hours each on invoking the CTM domain manager. However, 256 files are not deleted and left as is, resulting in the maximum number of files to 256 + *number-of-processes-being-invoked*.
  - *q*: Number of processes of system being invoked

In CTM, the internal trace is output to a file as maintenance information for each process. This file is created while invoking the process. The file deletion process is executed while invoking the CTM domain manager and after 24 hours each on invoking the CTM domain manager. However, 64 files are not deleted and left as is and therefore the maximum number of files is `64 +` *number-of-processes-being-invoked*.

The above mentioned disk capacity is a rough indicator. As a result, create the `%CTMSPOOL%` with enough excess space.

(2) In AIX

You must consider the following values and set up the kernel parameters to use CTM when using AIX. If the settings are not done properly, the CTM process cannot be invoked or while running the CTM processes they might as well terminate abnormally due to insufficient resources. For the kernel parameter settings, see the manual of the OS being used.

(a) File size for shared memory

Set up the file size for shared memory by the environment variable *EXTSHM*. Set up in such a way so that the shared memory allocated is greater that the value calculated by formula for calculations. The formula for calculations is the same as that for Windows. Reference the formula for calculations for Windows.

(b) Statistics information file size

The formula for calculating statistics information file size is the same as the one for Windows. Reference the formula for calculations for Windows.

(c) Number of file descriptors

Based on the options during invocation, add the number of file descriptors that you can use in the process as per the following formula for calculations in the CTM daemon. When the value set up in OS is not within the maximum value of file descriptors required in a CTM daemon setup by formula for calculations, the process invocation terminates with an error. Based on the formula for calculations, change the `nofiles` of `/etc/security/limits.conf` file.

- **Formula for calculating maximum value of number of file descriptors required in the CTM daemon**

  Maximum-value `= (`*A* `× 2 +` *B* `× 4 +` *C* `× 2 +` *D* `×` *E* `+` *F* `×` *number-of-* `EJBs +` *G* `+ 100)/0.8`

  Legend:

  - *A*: `-CTMMaxCTM` value (Value specified in `ctmdmd` to which the `ctmd` belongs)
  - *B*:`-CTMClientConnectCount` value
  - *C*:`-CTMServerConnectCount` value
  - *D*:`-CTMQueueCount` value
  - *E*:`-CTMQueueRegistCount` value
  - *F*:`-CTMDispatchParallelCount` value
  - *G*: Total number of EJB clients issuing Create

(d) Disk occupancy of $CTMSPOOL

The formula for calculating disk occupancy of `$CTMSPOOL` is as follows:

- **Formula for calculating disk occupancy of $CTMSPOOL**

  Disk-occupancy `= 7.0MB`

  `+(18.5MB + 1.0MB ×-CTMLogFileSize-`*specified-value*`×-CTMLogFileCount-`*specified-value*`) ×` *n*

+(1KB + 0.5KB × *k*) × (*m* + *l*)

+1KB × m × *j*

+224KB × *p*

+1,120KB × (64 + *q*)

+*file-size-for-shared-memory-of-CTM-domain-manager* × 5

+*file-size-for-shared-memory-of-CTM-daemon* × 5 × *n*

+*core-size-of-CTM-domain-manager*

+*core-size-of-CTM-daemon* × *n*

+*core-size-of-CTM-regulator* × 3 × *m*

+*core-size-of-OTM-gateway* × 3 × *l*

+(-CTMStatsFileSize-*specified-value* × -CTMStatsFileCount-*specified-value*) × *n*

Legend:

- *j*: Total number of EJBs
- *k*: Total number of clients that can be connected to CTM regulator and OTM gateway (-CTMClientConnectCount option specified value)
- *l*: Total number of OTM gateways
- *m*: Total number of CTM regulators
- *n*: Number of CTM daemons
- *p*: Number of processes of user applications being invoked and number of processes of user applications that did not terminate normally

  In CTM, the internal trace is output to a file as maintenance information for each user application. The file for internal trace output is created while invoking the process and remains as it is when the process does not terminate normally. The file deletion process is executed while invoking the CTM domain manager and after 24 hours each on invoking the CTM domain manager. However, 256 files are not deleted and left as it is, resulting in the maximum number of files to 256 + *number-of-processes-being -invoked*.
- *q*: Number of processes in system that is being invoked

  In CTM, the internal trace is output to a file as maintenance information for each process. This file is created while invoking the process. The file deletion process is executed while invoking the CTM domain manager and after 24 hours each on invoking the CTM domain manager. However 64 files are not deleted and left as it is, resulting in the maximum number of files to 64 + *number-of-processes-being-invoked*.

The above mentioned disk capacity is a rough indicator. As a result, create the $CTMSPOOL with enough excess space.

(3) In Linux

You must consider the following values and set up the kernel parameters to use CTM when using Linux. If the settings are not done properly, the CTM processes cannot be invoked or while running the processes they might terminate abnormally due to insufficient resources. For the kernel parameter settings, see the manual of the OS being used.

(a) File size for shared memory

Set up the file size for shared memory by kernel.shmmax of /etc/sysctl.conf file. Set up in such a way so that the shared memory allocated is greater that the value calculated by formula for calculations. The formula for calculations is the same as that for Windows. Reference the formula for calculations for Windows.

(b) Statistics information file size

The formula for calculating statistics information file size is the same as the one for Windows. Reference the formula for calculations for Windows.

(c) Number of file descriptors

Set up the number of file descriptors by `nofiles` of `/etc/security/limits.conf` file. Change the `nofiles` based on the formula for calculations. The formula for calculations is the same as that for AIX. Reference the formula for calculations for AIX.

(d) Disk occupancy of $CTMSPOOL

The formula for calculating disk occupancy of `$CTMSPOOL` is the same as that for AIX. Reference the formula for calculations for AIX.

5.2.5 Estimating the number of threads used by the cjclstartap process

The following shows how to estimate the number of threads used by the `cjclstartap` process:

- When using the Explicit Memory Management functionality
  number-of-threads-generated-by-Java-applications + 24
- When not using the Explicit Memory Management functionality
  number-of-threads-generated-by-Java-applications + 16

5.3 Estimating memory used for each process

This section explains how to estimate the amount of memory used by each process of the Application Server.

5.3.1 Estimating virtual memory usage of the J2EE server

This section describes how to estimate the virtual memory that is used. For details about the option for invoking JavaVM used in the formula for calculating the virtual memory usage, see *7.2.6 Configuration of memory space used by JavaVM when using serial GC and JavaVM options*.

Moreover, when you use the Explicit Memory Management functionality, apart from the contents described here, you must provide the estimation of the memory size used by the Explicit Memory Management functionality. For details on estimating the memory size used by the Explicit Memory Management functionality, see *7.11 Explicit heap tuning*.

(1) Formula for calculating virtual memory usage

The formula for calculating (Unit: Megabytes) virtual memory usage is as follows:

Virtual-memory-usage-of-J2EE-server $= A + B + C + (D + 22 + E) \times F + G + H$

Legend:

- *A*: Java heap size
  The default value is the value specified in the `-Xms` option when invoking JavaVM. This value is extended up to the maximum value specified in the `-Xmx` option when invoking JavaVM on running J2EE server.
- *B*: Metaspace area size
  The default value is the value allocated by the JavaVM. This value is extended up to the maximum value specified in the `-XX:MaxMetaspaceSize` option when invoking JavaVM on running J2EE server.
- *C*: Area size used by native program
  The value used differs for each OS. The following table describes the value of the area size used by native programs for each OS:

  Table 5–14: List of values of the area size used by native programs

OS type	Value of area used (Unit: megabytes)
Windows	300
AIX	400
Linux	600

- *D*: Number of J2EE server threads
  Number of threads used by J2EE server. For details about the number of threads used by J2EE server, see *5.2.1 Estimating the resources used by J2EE server*.
- *E*: Number of G1 GC threads
  Number of threads created when using G1 GC. G1 GC threads are not created when using Serial GC. When using G1 GC, specify a value equivalent to three times the number of logical CPUs.
- *F*: Stack area size
  This is the value specified in the `-Xss` option while invoking JavaVM.

- *G*: Explicit heap size

  The size of Explicit heap used by explicit heap management functionality. This value can be extended up to the maximum value specified in `-XX:HitachiExplicitHeapMaxSize` when invoking the J2EE server.

- *H*: Maximum size of the code cache area

  This value is specified in the option `-XX:ReservedCodeCacheSize`, when JavaVM is invoked.

(2) Notes when calculating the virtual memory usage

- The value of the area size used by native program keeps on changing. The value changes in the following cases:
  - When you change the trace size of ORB
  - When you change the value of area size used by JDBC driver
  - When you change the value of area size used by native library of products in use

  In such cases, add the memory used for each product to the value of the area size used by native programs. Calculate the memory used for each product as per the documents of the products in use.
- The virtual memory used in the standard configuration of the batch server might exceed the estimates due to the impact of the software products in use. Add this exceeded value to the value of the virtual memory used. To calculate the increased memory, calculate the memory used for each product according to the documentation of the software products.
- In Linux, the system slows down if the memory swap file size is insufficient. We recommend that you specify the real memory size as the upper limit of the virtual memory.

5.3.2 Estimating memory used by CTM daemon process

Use the following formula to estimate the memory usage of the CTM daemon process:

```
Maximum memory usage = A + {B + E + (2 × D)} × C + F × G + (D - 1) × 16 MB
```

Legend:

A: Size of basic memory (128 MB)

B: Maximum number of requests held in CTM daemon queue[#1]

C: Maximum request message size[#2]

D: Value of `-CTMClientConnectCount`[#3]

E: Value of `-CTMDispatchParallelCount`[#4]

F: Maximum number of threads required by CTM daemon. Use the following formula to estimate *F*:

$F = (a \times 4 + D \times 3 + b \times 2 + c \times d + E + e + 32) / 0.8$

Legend:

a: Value of `-CTMMaxCTM` (the value specified in the `ctmdmd` to which the `ctmd` belongs)[#6]

b: Value of `-CTMServerConnectCount`[#7]

c: Value of `-CTMQueueCount`[#8]

d: Value of `-CTMQueueRegistCount`[#9]

e: Total number of EJB clients that invoke the Create method

G: Thread stack size[#5]

#1

The maximum number of requests is the same as the queue length (if there are multiple queues, use the total length of all queues combined).

#2

To ensure there is enough leeway in the memory size, this value assumes that every request in the queue is of the maximum size. As a more realistic value, we recommend that you specify the average request message size.

#3

To ensure there is enough leeway in the memory size, this value assumes the maximum number of connections to the CTM daemon. As a more realistic value, we recommend that you specify the number of CTM regulators and OTM gateways that connect to the daemon.

#4

To ensure there is enough leeway in the memory size, this value assumes the maximum number of concurrent executions that can be registered in the CTM daemon. A more realistic value is the value of the `<parallel-count>` tag in the attribute file (if there are multiple queues, specify the total of the values of this tag for each queue).

#5

The value to use depends on the operating system you are using. The default thread stack size values for each operating system are as follows.

For details about the thread stack size, see the documentation for the operating system concerned.

Windows: 1 MB

AIX: 0.1 MB (96 KB)

Linux: 10 MB

#6

To ensure there is enough leeway in the memory size, this value assumes the maximum number of CTM daemons. As a more realistic value, we recommend that you specify a value equivalent to one fewer than the total number of CTM daemons in the CTM domain.

#7

To ensure there is enough leeway in the memory size, this value assumes the maximum number of connections to the CTM daemon. As a more realistic value, we recommend that you specify the number of J2EE servers that connect to the CTM daemon.

#8

To ensure there is enough leeway in the memory size, this value assumes the maximum number of queues that can be registered in the CTM daemon. As a more realistic value, we recommend that you specify the number of schedule queues registered in the CTM daemon.

#9

To ensure there is enough leeway in the memory size, this value assumes the maximum number of J2EE applications that can share the same schedule queue. As a more realistic value, we recommend that you specify the number of J2EE applications that share the same schedule queue.

> **Tip**
>
> When sending and receiving requests, a CTM daemon can temporarily use an amount of memory approximately three to five times the size of the business process message. In the preceding estimation formula, this excess is accommodated in the 128 MB of basic memory.
>
> However, when sending a large number of very large messages on the scale of 10 or 20 MB concurrently (a large parallel count), simultaneous processing might result in this leeway being exceeded. In these circumstances, attempting to estimate maximum memory usage accurately can yield an unrealistic value. We

recommended that you use the preceding estimation formula to calculate an approximate value, and then run your business processes and track the actual memory usage.

6 Estimating Resources to be Used (Batch Application Execution Platform)

This chapter describes how to estimate the resources and the virtual memory used in the systems for executing the batch applications. Reference this chapter for calculating the disk and memory capacity required for operating a system.

For details about estimating the resources and memory used for a J2EE application execution platform, see *5. Estimating Resources to be Used (J2EE Application Execution Platform)*.

6.1 Resources used for each system configuration

Set up of the hosting environment components, such as the OS and database, might be required for operating the system. This section describes the resources used for each system configuration and the estimates for the required resources, as the resources required in a system differ as per the system configuration. The following table describes the reference locations for the resources used for each system configuration and resource estimates:

Table 6–1: References of resources used for each system configuration and resource estimates

Resources used for each system configuration	Reference
Resources used for deploying batch server	*6.1.1*
Resources used by database	*6.1.2*
Resources for using CTM	*6.1.3*

6.1.1 Resources used for deploying batch server

This subsection describes the estimation of the resources used for deploying batch servers for each OS. When deploying the batch servers, estimate the resources used by the machine on which the batch servers is deployed.

Note that the *Example of option settings* in each table of estimation of resources used differs according to the OS and the kernel versions in use. Reference the manual of the OS being used and set up the estimates based on the estimation formula in the table. The settings are not required when the corresponding kernel parameter is not applicable in the OS being used.

Furthermore, for details about the estimation of the resources used for each process, see *6.2 Resources used for each process*.

For details about the required virtual memory, see *6.3 Estimating virtual memory usage*. For details about the disk occupancy, see the release notes of Application Server.

(1) In AIX

The following table describes the estimation of resources used by the Application Server machine in AIX:

Table 6–2: Estimation of the resources used by the Application Server machine (In AIX)

System resource		Parameter	Requirement	Example of option settings
Service unit[#1]	Shared memory	--	`PrfTraceBufferSize`[#2] × `1,024 + 18,496`	--
	Number of processes	--	`4`	--
	Number of threads	--	*number-of-batch- server-threads*[#3]`+34`	--
	Number of file descriptors	`nofiles`	*number-of-batch-server-file-descriptors*[#3] `+ 43`	`/etc/security/limits`
Management Server	Number of processes	--	`5`	--
	Number of threads	--	`56`	--

System resource		Parameter	Requirement	Example of option settings
	Number of file descriptors	`nofiles`	`43` + *number-of-batch-servers*	`/etc/security/limits`

Legend:

--: Not applicable.

#1

Service unit indicates the following:

Batch-server + *Performance-tracer*

#2

Specify the buffer memory size of performance tracer in the range of 512 kilobytes to 102,400 kilobytes. For details about `PrfTraceBufferSize`, see *4.12 Parameters applicable to logical performance tracers* in the *uCosminexus Application Server Definition Reference Guide*.

#3

For calculating the number of threads and file descriptors of the batch server, see *6.2.1 Estimating the resources used by batch server*.

(2) In Linux

The following table describes the estimation of resources used by the Application Server machine in Linux:

Table 6–3: Estimation of the resources used by the Application Server machine (In Linux)

System resource		Parameter	Requirement	Example of option settings
Service unit#1	Shared memory	`SHMMAX`	`PrfTraceBufferSize`#2 × `1,024 + 18,496`	`/proc/sys/kernel/shmmax`
	Number of processes	`threads-max`#3	`4`	`/proc/sys/kernel/threads-max`
	Number of threads	`threads-max`#3	*number-of- batch-server-threads*#4 + `34`	--
	Number of file descriptors	`fs.file-max`	*number-of-batch-server-file-descriptors*#4 + `43`	`/proc/sys/fs/file-max`
Management Server	Number of processes	`threads-max`#3	`5`	`/proc/sys/kernel/threads-max`
	Number of threads	`threads-max`#3	`56`	--
	Number of file descriptors	`fs.files-max`	`43` + *number-of-batch-servers*	`/proc/sys/fs/file-max`

Legend:

- : Not applicable.

#1

Service unit specifies the following:

Batch-server + *Performance-tracer*

#2

Specify the buffer memory size of performance tracer in the range of 512 kilobytes to 102,400 kilobytes. For details about `PrfTraceBufferSize`, *4.12 Parameters applicable to logical performance tracers* in the *uCosminexus Application Server Definition Reference Guide*.

#3

Specify the total of number of processes and threads in the `threads-max` parameter.

#4

For calculating the number of threads and file descriptors of the batch server, see *6.2.1 Estimating the resources used by batch server*.

6.1.2 Resources used by the database

This section describes the estimation of resources used by the DBMS.

For details about the required virtual memory, see *6.3 Estimating virtual memory usage*. For details about the disk occupancy see release notes of Application Server or Developer.

The following table describes the estimation of the resources used by the DBMS:

Table 6–4: Estimation of resources used by DBMS

DBMS	Used resources	Requirement
HiRDB	Maximum number of concurrent connections (`pd_max_users`)	$\sum_{i=1}^{n^{\text{※}1}}$ (Highest value of the connection pool of the resource adaptor $i^{\text{※}2}$ $\times 2^{\text{※}3} + 1^{\text{※}4}) + \alpha^{\text{※}5}$
Oracle	Maximum number of concurrent connections (`SESSIONS`)	$\sum_{i=1}^{n^{\text{※}1}}$ (Highest value of the connection pool of the resource adaptor $i^{\text{※}2}$ $+ 1^{\text{※}4})+ \alpha^{\text{※}5}$

#1

n is the total number of resource adapters deployed on the batch server in the system.

#2

Specify the value of the `MaxPoolSize` parameter in the Connector property file.

#3

Multiply by two if all of the following conditions are met:

1. Use `XATransaction` in the transaction support level.
2. Use the connection# within the transaction managed by Application Server to access the database.
3. Before the transaction of step 2 concludes, use the connection# outside the transaction to access the database.

#: This connection is the one that is acquired from the DB Connector in step1 and is the same connection.

#4

Add 1 only for the resource adapters with `XATransaction` specified in the transaction support level.

#5

The maximum value of + α is the total of the maximum values of the connection pools of the DB Connector you are using.

This value indicates connections that might temporarily exceed the maximum value for the connection pool. The details are as follows:

- When the functionality for detecting a connection failure is used

  When implementing the functionality to detect a connection failure, the unused connections removed from a connection pool are not counted as the connections within the connection pool. Therefore, the total number of connections in a connection pool and the unused connections removed from the connection pool might temporarily exceed the maximum number of connections in the connection pool.

- When the `cjclearpool` command is used

  In the normal mode, the used connections that are removed from a connection pool are not included in the connection count. Therefore, the total number of connections in the connection pool and the used connections removed from the connection pool might exceed the maximum value for the connection pool.

6.1.3 Resources for using CTM

The following subsection describes the estimation of resources used for CTM (Scheduling functionality for batch applications), for each OS.

Note that the *Example of option settings* in each table of estimation of resources used differs according to the OS and the kernel versions in use. Reference the manual of the OS being used and set up the estimates based on the estimation formula in the table. The settings are not required when the corresponding kernel parameter is not applicable in the OS being used.

(1) In AIX

The following table describes the estimation of resources used for CTM.

Table 6–5: Estimation of the resources used for CTM (In AIX)

System resource	Parameter	Requirement	Example of option settings
Share memory	--	`PrfTraceBufferSize`#1 × `1,024 + 18,496` + *shared-memory-of-CTM-domain-manager*#2 + *shared-memory-of-CTM-daemon*#2	--
Number of processes	--	`7 +` *Number-of- batch -servers*#3	--
Number of threads	--	`72 + (`*number-of-batch-server-threads*#4 `+ 7)` × *number-of-batch-servers*#3 + *number- of-threads-required-for-CTM-daemon*#5	--
Number of file descriptors	`nofiles`	`88 + (`*number-of-batch-server-file-descriptors*#4 + `6)` × *number-of-batch-servers*#3 + *number-of file-descriptors-required-for-CTM-daemon*#5	`/etc/security/limits`

Legend:

-- : Not applicable.

#1

Specify the buffer memory size of the performance tracer from 512 kilobytes to 102,400 kilobytes. For details about `PrfTraceBufferSize`, see *4.12 Parameters applicable to logical performance tracers* in the *uCosminexus Application Server Definition Reference Guide*.

#2

For calculating the values, see *6.1.3(1)(a) Formula for calculating file size for shared memory*.

#3

Indicates the value specified in the `<j2ee-server-count>` tag of Easy Setup definition file.

#4

For calculating the number of threads and file descriptors of the batch server, see *6.2.1 Estimating the resources used by batch server*.

#5

For calculating the number of threads and file descriptors required in the CTM daemon, see *6.1.3(1)(b) Formula for calculating number of threads and file descriptors required in the CTM daemon*.

(a) Formula for calculating file size for shared memory

You must calculate the shared memory of the CTM domain manager and CTM daemon to calculate file size of shared memory. The respective formula for calculations is as follows:

Use the following value for variable values in the calculations formula. For details about the parameters starting with `ctm.`, see *4.3 Easy Setup definition file* in the *uCosminexus Application Server Definition Reference Guide*.

Values used in the formula for calculations:

`-CTMMaxCTM`: `64`

`-CTMQueueCount`: `ctm.QueueCount`

`-CTMClientConnectCount`: `256`

`-CTMServerConnectCount`: `ctm.ServerConnectCount`

`-CTMEntryCount`: `-CTMClientConnectCount` + `-CTMServerConnectCount`

`-CTMServerCacheSize`: `ctm.ServerCacheSize`

`-CTMQueueRegistCount`: `ctm.QueueRegistCount`

`-CTMDispatchParallelCount`: `ctm.DispatchParallelCount`

- Formula for calculating file size for shared memory of CTM domain manager

  The formula for calculating the file size for shared memory of the CTM domain manager is as follows:

  File-size-for-shared-memory `(Unit: Bytes) =`

  `1,018,320 + (2,362 × -CTMMaxCTM` *specified-value*`)`

- Formula for calculating file size for shared memory of CTM daemon

  For CTM daemon, you must secure files for shared memory of fixed length and files for shared memory of variable length in each CTM daemon. The formula for the respective calculations is as follows:

  File-size-for-shared-memory-of-fixed-length `(Unit: Bytes) =`

  `551,840 + (1,208 × -CTMQueueCount` *specified-value*`)`

  File-size-for-shared-memory-of-variable-length `(Unit: Bytes) =`

  `1,027,008`

  `+ (928 × -CTMClientConnectCount` *specified-value*`)`

  `+ (256 × -CTMServerConnectCount` *specified-value*`)`

  `+ (512 × -CTMEntryCount` *specified-value*`)`

  `+ (1,024 × -CTMServerCacheSize` *specified-value*`)`

  `+ (512 × -CTMQueueCount` *specified-value*`)`

  `+ (544 × -CTMQueueCount` *specified-value* `× -CTMQueueRegistCount` *specified- value*`)`

  `+ (512 × -CTMDispatchParallelCount` *specified-value*`)`

(b) Formula for calculating number of threads and file descriptors required in the CTM daemon

For calculating the number of threads and file descriptors, you must calculate the number of threads and file descriptors required in the CTM demon. The formula for the respective calculations is as follows:

- Formula for calculating number of threads required in the CTM daemon

  Maximum-value `=`

  `(`*A* `× 4 +` *B* `× 3 +` *C* `× 2 +`*D* `×` *E* `+` *F* `+` *G* `+ 32) / 0.8`

Legend:

A: `-CTMMaxCTM` value (Value specified in `ctmdmd` to which `ctmd` belongs)

B: `-CTMClientConnectCount` value

C: `-CTMServerConnectCount` value

D: `-CTMQueueCount` value

E: `-CTMQueueRegistCount` value

F: `-CTMDispatchParallelCount` value

G: Total number of EJB clients that issued `Create`

- Formula for calculating number of file descriptors required in the CTM daemon

  Maximum-value =

  $(A \times 2 + B \times 4 + C \times 2 + D \times E + F \times \textit{number-of-EJB-interface} + G + 100) / 0.8$

  Legend:

  A: `-CTMMaxCTM` value (Value specified in `ctmdmd` to which `ctmd` belongs)

  B: `-CTMClientConnectCount` value

  C: `-CTMServerConnectCount` value

  D: `-CTMQueueCount` value

  E: `-CTMQueueRegistCount` value

  F: `-CTMDispatchParallelCount` value

  G: Total number of EJB clients that issued 'Create'

(2) In Linux

The following table describes the estimation of the resources used for CTM:

Table 6–6: Estimation of the resources used for CTM (In Linux)

System resource	Parameter	Requirement	Example of option settings
Shared memory	`SHMMAX`	`PrfTraceBufferSize`#1 × 1,024 + 18,496 + *shared-memory-of-CTM-domain-manager*#2 + *shared-memory-of-CTM-daemon*#2	`/proc/sys/kernel/shmmax`
Number of processes	`threads-max`	7 + *number-of-batch-servers*#3	`/proc/sys/kernel/threads-max`
Number of threads	`threads-max`	72 + (*number-of-batch-server-threads*#4 + 7) × *number-of-batch-servers*#3 + *number-of-threads-required-in-CTM-daemon*#5	--
Number of file descriptors	`fs.file-max`	88 + (*number-of-batch-server-file-descriptors*#4 + 6) × *number-of-batch-servers*#3 + *number-of-file-descriptors-required-in-CTM-daemon*#5	`/proc/sys/fs/file-max`

Legend:

-: Not applicable.

#1

Specify the buffer memory size of the performance tracer from 512 kilobytes to 102,400 kilobytes. For details about PrfTraceBufferSize, see *4.12 Parameters applicable to logical performance tracers* in the *uCosminexus Application Server Definition Reference Guide*.

#2

For calculating the values, see *6.1.3(1)(a) Formula for calculating file size for shared memory*.

#3

Indicates the value specified in the `<j2ee-server-count>` tag of the Easy Setup definition file.

#4

For calculating the number of threads and file descriptors of the batch server, see *6.2.1 Estimating the resources used by batch server*.

#5

For calculating the number of threads and file descriptors required in the CTM daemon, see *6.1.3(1)(b) Formula for calculating number of threads and file descriptors required in the CTM daemon*.

6.2 Resources used for each process

This section describes the estimation of the required amount of resources used in each process of Application Server.

6.2.1 Estimating the resources used by batch server

This subsection describes how to estimate the number of threads and file descriptors of the batch server. Reference this subsection when calculating the disk and memory capacity required for operating Application Server.

(1) Number of threads

The formula for calculating the number of threads is as follows. The total of (a) and (b) is the number of threads used by the batch server.

(a) Basic number of threads

Maximum-number-of-threads = `68 + A + B + C + D + E + F + G + H + I`

Legend:

- *A*: Number of threads of CORBA Naming Service (= *Number-of-connections-between-the-client-and-CORBA-Naming-Service* × `2 +` *Number-of-requests-received-concurrently* `+` *Number-of-threads-generated-during-initialization* (If the `vbroker.agent.enableLocator` value is `true, 6`, if the value is `false, 4) + 1)`

  However, the threads are calculated only when CORBA Naming Service is invoked as inprocess (`inprocess` is specified in the `ejbserver.naming.startupMode` key of `usrconf.properties`).

  For details about the estimation of number of threads of CORBA Naming Service, see *5.2.1(3) Estimating number of threads of CORBA Naming Service (When invoking as in-process)*.
- *B*: Maximum number of database connections used concurrently

  If the connection pooling functionality is used, the maximum number of connection pools (value of `MaxPoolSize` specified in the Hitachi Connector Property file. Total of connection pools if there are multiple resource adapters) is used as the value.

  If the connection pooling functionality is not used, the value is obtained from the maximum number of concurrent requests or the number of connections used for each request (if one connection is used for one request, the maximum number of concurrent requests is used as the value).
- *C*: Maximum number of concurrently executed transactions, if the JTA transaction is used (one thread is used for each transaction in which a transaction timeout occurs. In the case of one request in one transaction, the maximum number of concurrent requests is used as the value)
- *D*: *Maximum-number-of-connection-pools* (total of connection pools if there are multiple resource adapters#) × `2`

  #: Value of `MaxPoolSize` specified in the Hitachi Connector Property file
- *E*: Number of resource adapters using the connection pooling functionality
- *F*: Number of threads used for the conclusion and recovery processing of a global transaction (add `16`, if a global transaction is used)
- *G*: Number of simultaneous client connections to the management server (specify 5 if the number of simultaneous client connections to the management server is 5 or fewer, and 100 if the number is 100 or more)
- *H*: If settings are specified to start the thread that manages the reply receiving thread (`vbroker.ce.iiop.ccm.htc.threadStarter=true`), add `5`.

- *I*: Add the following value if settings are specified to control the closing of connections when a timeout occurs (`vbroker.ce.iiop.ccm.htc.readerPerConnection=true`):

  (*Number-of-J2EE-servers-with-remote-invocation-destination-EJBs* `+ 1`) × `2`

  If CTM is used, add the following values in addition to the above:

  - **For scheduling of J2EE applications**

    Number-of-running-J2EE-applications `+ 1`

  - **For scheduling of Stateless Session Beans**

    Number-of-Stateless-Session-Beans-to-be-scheduled `+ 1`

(b) Number of threads used according to JavaVM option specifications

According to JavaVM option specifications, use the following formula for calculation. Add A only when `-XX:+UseG1GC` option is specified, and add B only when `-XX:+HitachiUseExplicitMemory` option is specified.

Maximum-number-of-threads = A + B

Legend:

- *A*: Number of threads used by G1 GC *(Value specified in* `-XX:ParallelGCThreads` option. When this option is not specified, the default value of `-XX:ParallelGCThreads` option based on the number of logical CPUs. Note that the value is determined by the number of logical CPUs that exist when starting the J2EE server. Hence, the number of threads does not change even if the number of logical CPUs changes after the server is started.)
- *B*: Number of threads used by the Explicit Memory Management functionality (The number of logical CPUs. However, this number is 8 when the number of logical processors is 8 or more. This number is determined by the number of logical CPUs that exist when starting the J2EE server. Hence, the number of threads does not change even if the number of logical CPUs changes after the server is started.)

For JavaVM options, see the following sections in the *uCosminexus Application Server Definition Reference Guide*:

- *14.5 Java HotSpot VM options that can be specified in Cosminexus*
- *-XX:[+|-]HitachiUseExplicitMemory (Explicit Memory Management functionality option)*

(2) Number of file descriptors

The formula for calculating the numbers of file descriptors is as follows:

Maximum-number-of-file-descriptors `= (139 +` *A* `+` *B* `×2 +` *C* `×2 +` *D*`) / 0.8`

Legend:

- *A*: Number of database connections
- *B*: Number of concurrent client connections to the management server
- *C*: Number of resource adapters
- *D*: Number of JAR files specified in `add.class.path` key of `usrconf.cfg`

6.2.2 Estimating the resources used by Administration Agent

The estimation of the resources used by Administration Agent is the same as that for the J2EE application execution platform. For details about the estimation of resources used by Administration Agent, see *5.2.2 Estimating the resources used by Administration Agent*.

6.2.3 Estimating the resources used by performance tracer

The estimation of the resources used by the performance tracer is the same as that for the J2EE application execution platform. For details about the estimation of resources used by performance tracer, see *5.2.3 Estimating the resources used by performance tracer*.

6.2.4 Estimating the resources used by CTM

The estimation of the resources used by CTM is the same as that for the J2EE application execution platform. For details about the estimation of resources used by CTM, see *5.2.4 Estimating the resources used by CTM*.

6.3 Estimating virtual memory usage

This section describes how to estimate the virtual memory that is used. For details about the option for invoking JavaVM used in the formula for calculating the virtual memory usage, see *7.2.6 Configuration of memory space used by JavaVM when using serial GC and JavaVM options*.

Moreover, when you use the Explicit Memory Management functionality, apart from the contents described here, you must provide the estimation of the memory size used by the Explicit Memory Management functionality. For details on estimating the memory size used by the Explicit Memory Management functionality, see *7.11 Explicit heap tuning*.

6.3.1 Formula for calculating virtual memory usage

The formula for calculating (Unit: Megabytes) virtual memory usage is as follows:

$$\textit{Virtual-memory-usage-of-batch-server} = A + B + C + (D + 10) \times E + F + G$$

Legend:

- *A*: Java heap size
  The default value is the value specified in the `-Xms` option when invoking JavaVM. This value is extended up to the maximum value specified in the `-Xmx` option when invoking JavaVM on running J2EE server.
- *B*: Metaspace area size
  The default value is the value allocated by JavaVM. When J2EE server is running, this value increases to a maximum of the value specified in the `-XX:MaxMetaspaceSize` option when JavaVM started.
- *C*: Area size used by native program
  The value used differs for each OS. The following table describes the value of the area size used by native program for each OS:

  Table 6–7: List of values of the area size used by native program

OS type	Value of area used (unit: megabytes)
Windows	300
AIX	400
Linux	600

- *D*: Number of batch server threads
  Number of threads used by the batch server. For details about the number of threads used by batch server, see *6.2.1 Estimating the resources used by batch server*.
- *E*: Stack area size
  This is the value specified in the `-Xss` option when invoking JavaVM.
- *F*: Explicit heap size
  This is the size of the Explicit heap used in the Explicit Memory Management functionality. This value can be extended up to the maximum value specified in `-XX:HitachiExplicitHeapMaxSize` when invoking JavaVM on running J2EE server. Note that by default Explicit heap is invalid.
- *G*: Maximum size of the code cache area
  Specify this value in the `-XX:ReservedCodeCacheSize` option, when invoking JavaVM.

6.3.2 Notes when calculating the virtual memory usage

- The value of the area size used by native programs keeps on changing. The value changes in the following cases:
  - When you change the trace size of ORB
  - When you change the value of area size used by JDBC driver
  - When you change the value of area size used by native library of products in use

  In such cases, add the memory used for each product to the value of the area size used by native programs. Calculate the memory used for each product as per the documents of the products in use.
- The virtual memory used in the standard configuration of the batch server, might exceed the estimates due to the impact of the software products in use. Add this exceeded value to the value of the virtual memory used. To calculate the increased memory, calculate the memory used for each product as per the documents of the software products.
- In Linux, the system slows down if the memory swap file size is insufficient. We recommend that you specify the real memory size as the upper limit of the virtual memory.

7 JavaVM Memory Tuning

To improve the processing performance of systems, you must properly tune JavaVM as Java VM is the platform for the infrastructure. Hitachi JavaVM (called JavaVM hereafter) manages two types of memory spaces.

This chapter describes memory management in GC and JavaVM, and the tuning of the Java heap and Explicit heap.

7.1 Overview of GC and JavaVM memory management

The purpose of JavaVM tuning is to improve the processing performance of the system. Tuning the JavaVM to manage memory optimally with the GC mechanism in mind will yield improvements to system performance. Because the behavior of the GC mechanism differs according to how memory is managed, you need to select the memory management method that best meets the requirements of the system. For an Application Server, you can select from the following three memory management methods.

Table 7–1: Characteristics of memory management methods

No.	Memory management method	Characteristics
1	Serial GC	• Suitable for systems that prioritize throughput. • This method offers high throughput. • There are two approaches to GC: *Full GC* in which GC takes a long time, and *copy GC* in which GC finishes in a shorter time. You cannot control the duration of GC. • You can make Full GC occur less frequently by tuning the memory size.
2	Combination of Serial GC and Explicit Memory Management functionality	• Suitable for typical Web front-end systems that use sessions. • This method offers high throughput. • There are two approaches to GC: *Full GC* in which GC takes a long time, and *copy GC* in which GC finishes in a shorter time. You cannot control the duration of GC. • In session-based systems, you can make Full GC occur less frequently by tuning the memory size and using the Explicit heap to manage sessions.
3	G1 GC	• Suitable for memory-intensive systems and systems that prioritize responsiveness. • This method offers lower throughput than the preceding two. • There are two approaches to GC: *Full GC* in which GC takes a long time, and a combination of *young GC* and *mixed GC* in which GC finishes in a shorter time. You can control the duration of young GC and mixed GC. • You can make Full GC occur less frequently by tuning the memory size and increasing the number of threads that perform GC.

For details about serial GC, see sections 7.2 to 7.10. For details about the Explicit Memory Management functionality, see sections 7.11 to 7.14. For details about G1 GC, see sections 7.15 and 7.16.

7.2 Mechanism of serial GC

This section explains the mechanism by which serial GC works.

7.2.1 Overview of serial GC

GC is a technique that automatically reclaims memory that a program has finished using, making that memory available to other programs.

Program processing stops while GC is occurring. For this reason, appropriate implementation of GC significantly impacts the processing performance of the system.

The Java objects created in the program using `new`, occupy the memory space that is managed by JavaVM. The time period from the point where Java object gets created up to the point where it becomes redundant is called the *lifespan of a Java object*.

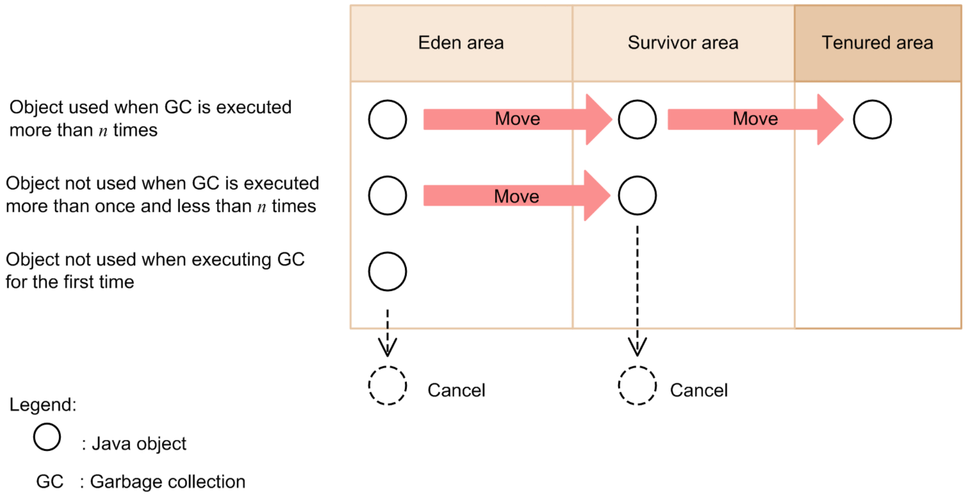
There are two types of Java objects; those with a short lifespan and those with a long lifespan. In the case of a Java application running on the server, a number of Java objects are created by request and response, or by transaction management. These Java objects have a short lifespan, as they become redundant when the processing ends. On the other hand, the Java objects that are continuously being used while the application is running have a long lifespan.

To implement GC effectively, you need to reclaim memory efficiently by targeting objects with a short lifespan. Another key to maintaining system performance is to avoid unnecessary GC for repeatedly used objects with long lifespans. An approach that achieves both these objectives is *generational GC*.

Generational GC manages Java objects in a *New* area which stores objects with a short lifespan, and a *Tenured* area which stores objects with a long lifespan. The New area is further divided into an *Eden* area for objects that were just created by the `new` operator, and a *Survivor* area for objects that have survived GC at least once. Java objects that have undergone GC in the New area a certain number of times are determined to have long-term utility and transferred to the Tenured area.

The following figure shows an overview of the memory spaces managed by generational GC and how Java objects move among them.

Figure 7–1: Overview of memory spaces managed by generational GC and Java objects


Eden area
Survivor area
Tenured area
Object used when GC is executed more than *n* times
Move
Move
Object not used when GC is executed more than once and less than *n* times
Move
Object not used when executing GC for the first time
Cancel
Cancel
Legend:
○ : Java object
GC : Garbage collection

There are two types of GC executed by generational GC under the serial GC technique.

- Copy GC

GC that applies to the Eden area and the Survivor area. Copy GC is triggered when the Eden area becomes full.

> **Important note**
>
> On the Application Server, you can select serial GC or G1 GC (`UseG1GC`). You cannot use parallel GC, concurrent GC, or incremental GC.

- Full GC

  GC that applies to all areas in the JavaVM including the Tenured area. Full GC is triggered when the Tenured area reaches a certain size.

Generally, copy GC takes less time to perform than Full GC.

The following explains GC processing using the example of a specific Java object (Object A).

Processing executed in the Eden area

After creating an object A, the object A is destroyed if not used when the first copy GC is executed.

If object A is used when first copy GC is executed, the object A moves from the Eden area to the Survivor area.

Processing executed in the Survivor area

The object A that moved to the Survivor area, moves from the Survivor area to the Tenured area even when the copy GC is executed for several times afterwards. The threshold value for frequently moving the object differs according to the used state of the JavaVM options and the Java heap. In Figure 7-1, the threshold value is assumed to be `n` times.

After moving to the Survivor area, if object A is not used when the copy GC is executed less than `n` times, then the object A is destroyed by that copy GC.

Processing executed in the Tenured area

Once the object A is moved to the Tenured area, the object A cannot be destroyed by the copy GC, because the copy GC is executed only for the Eden area and the Survivor area.

Moving objects in this way increases the used space of the Tenured area. Full GC is triggered when the used space of the Tenured area reaches a certain size.

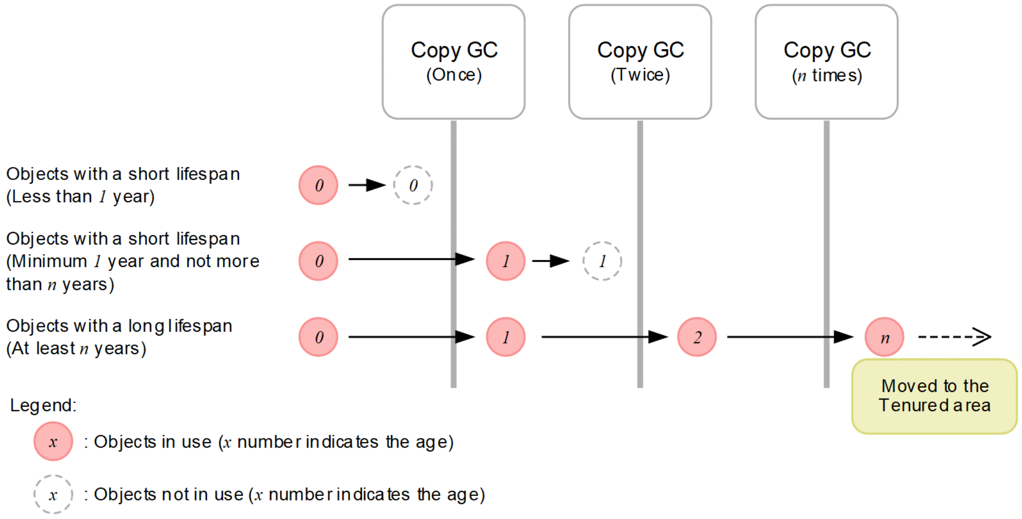
When tuning the JavaVM, by setting the appropriate size and ratio of each memory space in the JavaVM options, you can prevent unnecessary objects from being moved to the Tenured area. This reduces the frequency with which Full GC occurs.

7.2.2 Relation between the lifespan of an object and its age

The number of times copy GC is executed for an object is called the *age of the object*.

The following figure shows the relation between the lifespan and the age of an object:

Figure 7–2: Relation between the lifespan and the age of an object


Copy GC (Once)
Copy GC (Twice)
Copy GC (n times)
Objects with a short lifespan (Less than 1 year)
Objects with a short lifespan (Minimum 1 year and not more than n years)
Objects with a long lifespan (At least n years)
0
1
2
n
Moved to the Tenured area
Legend:
x : Objects in use (x number indicates the age)
x : Objects not in use (x number indicates the age)

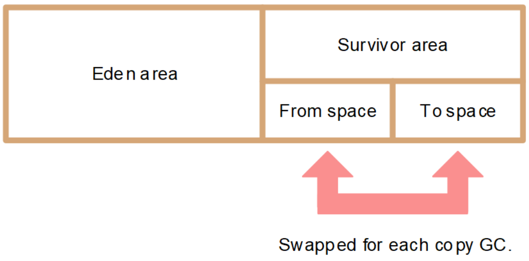
After the application has started, the initialization process has finished, and the copy GC has been executed a number of times, the objects with a long lifespan that are required for a long period, move to the Tenured area. As a result, shortly after starting the application, the Java heap reaches a stable state and most of the Java objects that get created are the objects that have a short lifespan. Especially, if the tuning of the New area has been performed appropriately, then after the Java heap stabilizes, a majority of the objects that are objects with a short lifespan get collected during the first copy GC.

7.2.3 Mechanism of copy GC

In JavaVM, the memory space of the New area for which copy GC is executed, is divided into the Eden area and the Survivor area. The Survivor area is furthermore divided into the From space and the To space. The From space and the To space have the same memory size.

The following figure shows the configuration of the New area:

Figure 7–3: Configuration of the New area



An Eden area is the area where the objects created using `new` are stored initially. When `new` is executed in a program, memory from the Eden area is allocated.

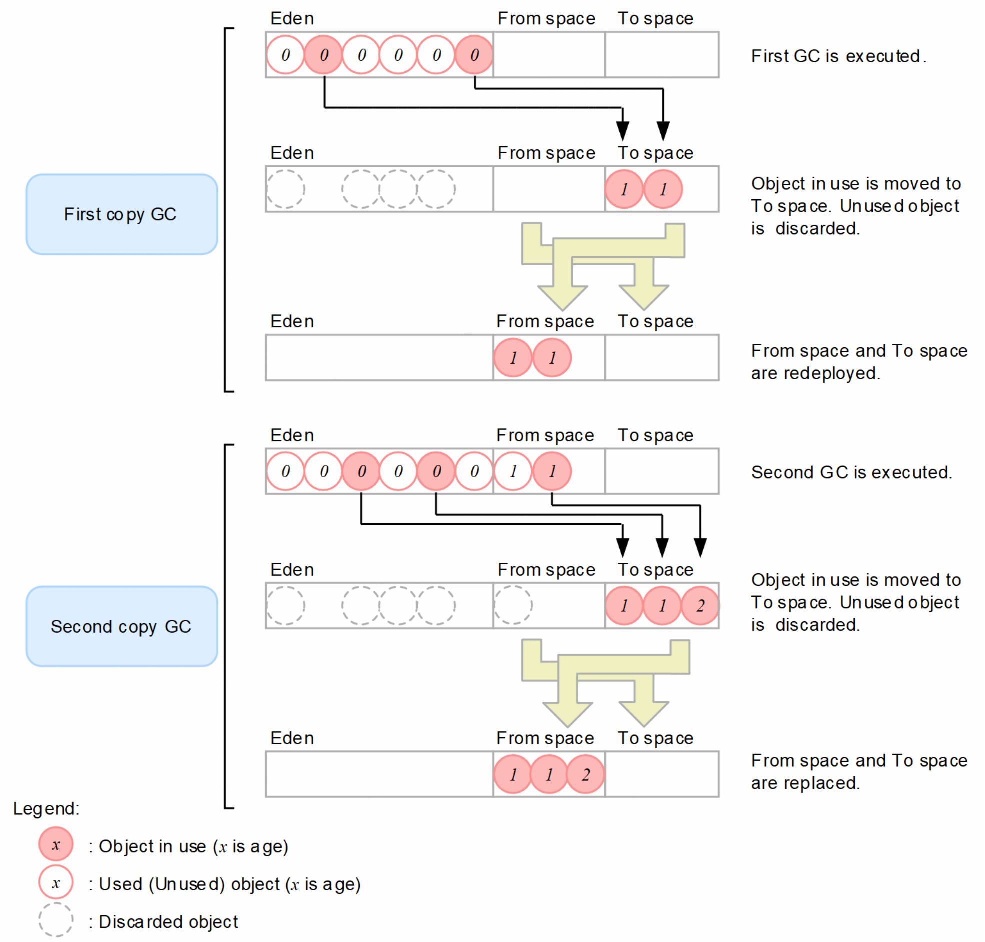
When the Eden area becomes full, copy GC is executed. The following processing is performed in copy GC:

1. Among the Java objects existing in the Eden area and the From space of the Survivor area, the objects that are in use are copied to the To space of the Survivor area. The Java objects that are not in use are destroyed.
2. The To space and the From space of the Survivor area are swapped.

As a result, the Eden area and the To space become empty, and the objects that are in use remain in the From space.

The following figure shows the movement of objects when copy GC is executed:

Figure 7–4: Movement of objects that occurs during copy GC

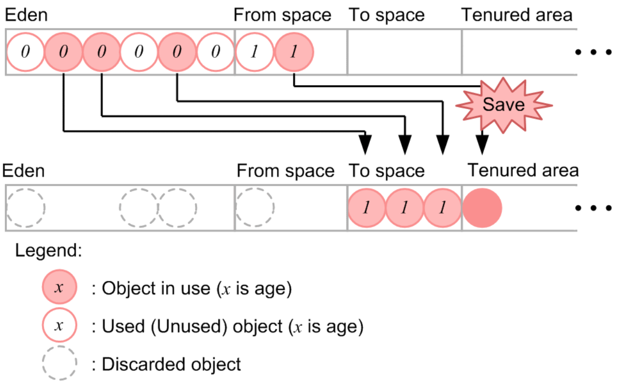

Eden
From space
To space
0
First GC is executed.
1
Object in use is moved to To space. Unused object is discarded.
First copy GC
From space and To space are redeployed.
Second GC is executed.
2
Second copy GC
From space and To space are replaced.
Legend:
x
: Object in use (x is age)
: Used (Unused) object (x is age)
: Discarded object

In this way, when copy GC occurs, the objects that are in use move to and fro between the From space and the To space. If the objects with a long lifespan continue to move to and fro, however, copy processing becomes an overhead. To prevent this, set a threshold value for the frequency of switching the Java objects between the From space and the To space, so that the Java objects whose age has reached the threshold value move to the Tenured area.

7.2.4 Saving objects

The activity of moving those Java objects whose age has not reached the threshold value to the Tenured area, is called *Saving*. Saving occurs when the number of in-use objects in the Eden area and the From space increase and the memory size of the To space, where these objects will be moved to, is not sufficient to hold those objects during copy GC. In such a case, the objects that could not move to the To space move to the Tenured area.

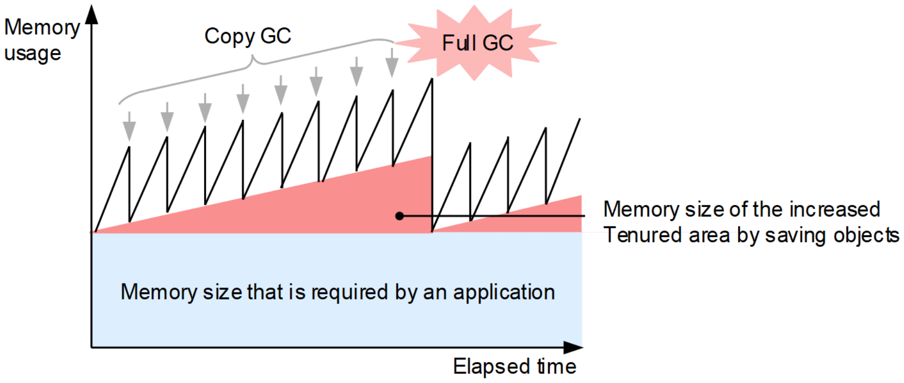
Figure 7–5: Saving objects



When the objects are saved, the objects with a short lifespan that originally were not supposed to be saved in the Tenured area, get saved in the Tenured area. If this process repeats, objects that were meant to be collected by copy GC remain in the memory space. This increases the memory usage of the Java heap, ultimately triggering Full GC.

The following figure shows the change in memory usage when the objects are saved:

Figure 7–6: Change in memory usage when the objects are saved



In Full GC, a system might stop from a few seconds to 10 seconds.

As a result, when determining the configuration of the memory space and memory size, you must achieve a balance between the Eden area and the Survivor area, so that the objects are not saved.

7.2.5 Areas not subject to GC (using an Explicit heap area with the Explicit Memory Management functionality)

In addition to the Eden area, Survivor area, and Tenured area, the JavaVM uses an area called the Explicit heap. The Explicit heap area is not subject to GC.

Specify objects to be saved in the Explicit heap area using the automatic allocation setup file and the Explicit Memory Management functionality API. At the timing of the specified objects being moved from the Survivor area to the Tenured area, the specified objects are moved to the Explicit heap area. By specifying long-life objects to exclude from collection by the copy GC mechanism, you can reduce the memory used by the Tenured area and make Full GC occur less frequently. You can also create the specified objects in the Explicit heap area using the automatic allocation setup file of the Explicit Memory Management functionality or the Explicit Memory Management functionality API.

For details about the Explicit Memory Management functionality, see *7. Suppression of Full GC by Using the Explicit Memory Management Functionality* in the *uCosminexus Application Server Expansion Guide*.

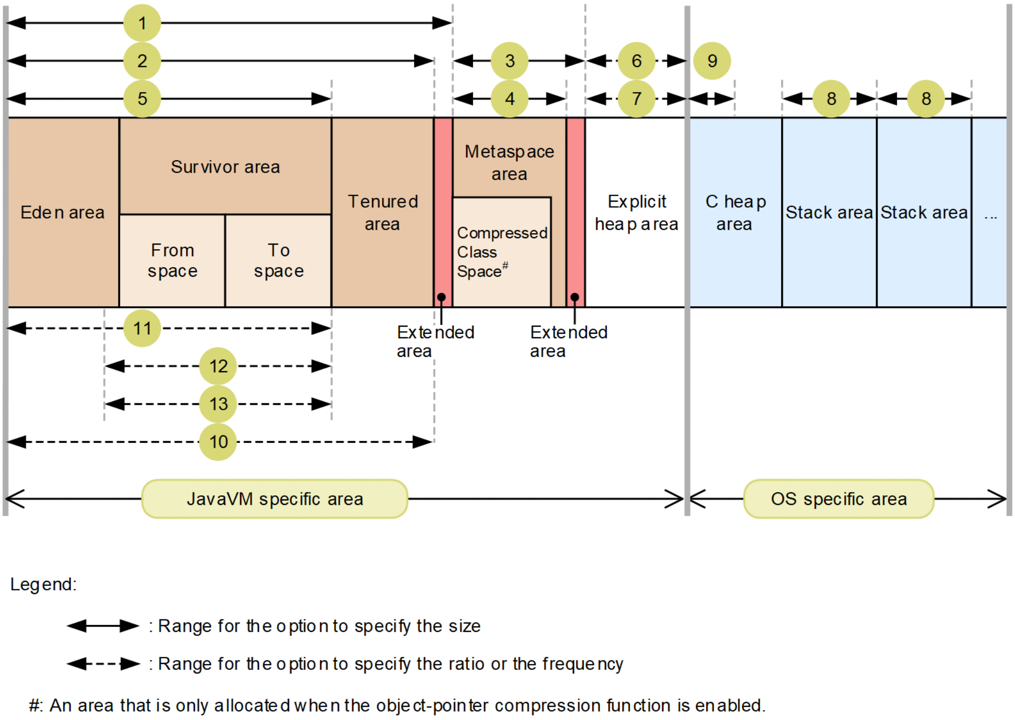
7.2.6 Configuration of memory space used by JavaVM when using serial GC and JavaVM options

This subsection explains the configuration of the memory space used by JavaVM when using serial GC, and the JavaVM options.

JavaVM uses two types of memory space; *JavaVM specific area* and *OS specific area*.

The following figure shows the configuration of the memory space used by JavaVM. The numbers in the figure correspond to the numbers in Table 7-2.

Figure 7–7: Configuration of memory space used by JavaVM



The respective areas are explained below. The area formed by combining the Eden area, the Survivor and the Tenured area is called the *Java heap*.

Eden area

Java objects that are created by `new` are initially stored in the DefNew::Eden area.

Survivor area

Among the Java objects stored in the New area, the DefNew::Survivor area stores those Java objects that are not destroyed when copy GC is executed. The From space and the To space configure the Survivor area. The size of the From space and the To space is same.

Tenured area

The Tenured area is a memory space that stores Java objects that are considered to be necessary for a longer duration. The Java objects for whom a copy GC is executed more than the specified number of times in the Survivor area and that were not destroyed, are moved to this area.

Metaspace area

The Permanent area stores the information of `class` that has been loaded.

If the object-pointer compression function is enabled, an area called the *Compressed Class Space* is created in the Metaspace area. This area contains information about the classes referenced by objects in the Java heap. Other information including method information is stored in the part of the Metaspace area that is outside the Compressed Class Space. For details about the object-pointer compression function, see *9.18 Object-pointer compression function* in the *uCosminexus Application Server Maintenance and Migration Guide*.

Explicit heap area

The Explicit heap area stores Java objects that are not subject to Full GC. The Explicit heap area is the JavaVM-specific memory space and the Explicit heap area is allocated only when using the Explicit Memory Management functionality.

C heap area

The C heap area is used by JavaVM. The C heap is also used by the native library that is called by JNI.

Reference note

Maximum size of the C heap area (In AIX)

In AIX, when you use the JNI to execute the program for invoking JavaVM, set up the maximum size of the C heap area with one of the following methods. If no method is specified, the default value of the shell `datasize` resource becomes the maximum size of the C heap area.

- Specify the C heap area size in the value of the `datasize` resource in shell, and then execute the program.
  For `csh` (C shell): `limit datasize` *C-heap-area-size*
  For `sh` (default shell) and `ksh`: `ulimit -d` *C-heap-area-size*
- Specify `-bmaxdata:` *C-heap-area-size* in the linkage option for program generation.
- Specify the `MAXDATA` value in the environment variable `LDR_CNTRL`, and then execute the program.
  For `csh` (C shell): `setenv LDR_CNTRL MAXDATA=`*C-heap-area-size*
  For `sh` (default shell) and `ksh`: `export LDR_CNTRL=MAXDATA=`*C-heap-area-size*
- Use the `setrlimit()` system call to implement the processing for setting up the C heap area size from within a program.

The examples includes the following area:

Code cache area

This area stores the JIT compilation code generated by JIT compilation.

JavaVM speeds up the processing by JIT compiling and executing the Java methods with a large invocation count and loop count.

Reference note

Maximum size of the code cache area

Specify the maximum size of the code cache area in the `ReservedCodeCacheSize` option.

Specify a value greater than the default value in the `ReservedCodeCacheSize` option. For details on the default value, see *14.4 Default values of the Java HotSpot VM options that can be specified in Cosminexus* in the *uCosminexus Application Server Definition Reference Guide*.

Also, if the code cache area has depleted or might deplete, consider extending the code cache area.

Note the following points when you extend the code cache area:

- You cannot estimate the size of the JIT compilation code by calculation. Therefore, actually measure the amount of the code cache area used in the Java application execution environment, consider the amount of the code cache area (maximum 2 MB) to be used by the system, and then estimate the maximum size of the code cache area.
- For estimating the usage of the virtual memory, see *5.3 Estimating memory used for each process* or *6.3 Estimating virtual memory usage*.

Stack area

Stack area for Java threads.

Reference note

Stack area size of the main thread

To change the stack area size of the main thread, set up a value greater than the value specified in `-Xss`. You cannot use the `-Xss` option to change the stack area size of the main thread. Because the automatic start processing of a J2EE application at J2EE server startup occurs in the main thread, automatic start processing might fail. In this case, you can start the J2EE server with the `-nostartapp` option specified to prevent the J2EE application from starting automatically, and then manually start the J2EE application. Alternatively, you can use a Management action to start the J2EE application.

The following table describes the JavaVM options that specify the size and ratio of the respective areas. The numbers in the table correspond to the numbers in Figure 7-7.

Table 7–2: JavaVM options for specifying the size and ratio of the JavaVM memory space

No.	Option name	Meaning of the option
1	`-Xmxsize`	Sets the maximum size of Java heap.
2	`-Xmssize`	Sets the initial size of Java heap.
3	`-XX:MaxMetaspaceSize=`*size*	Sets the maximum size of the Metaspace area.
4	`-XX:MetaspaceSize=`*size*	Sets the initial size of the Metaspace area.
5	`-Xmnsize`	Sets the initial value and the maximum value of the New area.
6	`-XX:[+|-]HitachiAutoExplicitMemory`	Allocates the memory used in Explicit memory block when starting the JavaVM using automatic allocation function.#
7	`-XX:HitachiExplicitHeapMaxSize=`*size*	Sets the maximum size of the Explicit heap area#.
8	`-Xsssize`	Sets the maximum size of one stack area.
9	`-XX:ReservedCodeCacheSize=`*size*	Sets the maximum size of the code cache area from among the areas used by JavaVM.
10	`-XX:NewRatio=`*value*	Sets the ratio of the Tenured area with respect to the New area. If the *value* is 2, the ratio of the New area and the Tenured area becomes 1:2.
11	`-XX:SurvivorRatio=`*value*	Sets the ratio of the Eden area with respect to From space and To space of the Survivor area.

No.	Option name	Meaning of the option
		If 8 is set in *value*, the ratio of the Eden area, From space, and To space becomes 8:1:1.
12	`-XX:TargetSurvivorRatio=`*value*	Sets the target value for the ratio of the Survivor area occupied by Java objects after copy GC is executed.
13	`-XX:MaxTenuringThreshold=`*value*	Sets the maximum number of times to swap Java objects between the From space and the To space when executing copy GC. The Java objects, for which the frequency of switching exceeds the set number of times, are moved to the Tenured area.

Note:

The unit of *size* is bytes.

#

The prerequisite options for using the Explicit Memory Management functionality must be enabled. For details, see *7.11 Explicit heap tuning*.

Reference note

- How to set JavaVM options:
  Set the JavaVM options at the following places:

Table 7–3: Location for setting the JavaVM options

Target	How to set	Location for setting
J2EE server	Smart Composer functionality	Definition file Easy Setup definition file Setup target Logical J2EE server (`j2ee-server`) Parameter name `add.jvm.arg`
Batch server	Smart Composer functionality	Definition file Easy Setup definition file Setup target Logical J2EE server (`j2ee-server`) Parameter name `add.jvm.arg`
EJB Client Application	Edit file	Definition file `usrconf.cfg`# Parameter name `add.jvm.arg` key

#

The file that is activated when the `cjclstartap` command is used for starting.

Tip

The default value of the respective options depends on the OS. For details about the default values of options, see *14.4 Default values of the Java HotSpot VM options that can be specified in Cosminexus* in the *uCosminexus Application Server Definition Reference Guide*.

Important note

An OutOfMemory exception occurs if there is not enough space available in any of the Java heap area, Metaspace area, Compressed Class Space, or C heap area. The JavaVM will not work normally until the memory shortage is resolved, which can take some time.

By specifying the following options when OutOfMemory occurs, the impact of OutOfMemory on the system can be reduced:

- -XX:+HitachiOutOfMemoryAbort **(forceful termination when OutOfMemory occurs)**
- -XX:+HitachiOutOfMemoryHandling **(OutOfMemory handling functionality)**

The **functionality for forceful termination when OutOfMemory occurs** forcibly terminates the J2EE server when an OutOfMemory exception occurs due to an insufficient Java heap, Metaspace area, Compressed Class Space, or other memory area. When an OutOfMemory exception occurs due to an insufficient Java heap area, Metaspace area, Compressed Class Space, or C heap area, this functionality forcibly terminates the J2EE server and automatically restarts the J2EE server. This helps in fast recovery of a J2EE server so that that server can operate successfully.

OutOfMemory handling functionality is a functionality for which the functionality of forceful termination when OutOfMemory occurs is a prerequisite. Even if you are using the functionality of forceful termination when OutOfMemory occurs, you can continue the execution of J2EE server, if specific conditions are met by using this functionality.

If OutOfMemory occurs due to insufficient Java heap while allocating the large number of objects or while allocating huge objects in request processing, use the OutOfMemory handling functionality to continue the execution of J2EE server.

For details about the option, see *-XX:[+|-]HitachiOutOfMemoryAbort (Forced termination option)* and *-XX:[+|-]HitachiOutOfMemoryHandling (OutOfMemory handling option)* in the *uCosminexus Application Server Definition Reference Guide*.

- Special tuning approach related to the Metaspace area

  The following explains an approach to tuning that uses the characteristics of the Metaspace area. In JDK7 and earlier, the Permanent area always used an amount of memory equivalent to the value specified in the JavaVM options, but the Metaspace area did not. In fact, the Metaspace area only uses the amount of memory required for execution.

  (a) Reducing the risk of OutOfMemoryError exceptions in the Metaspace area

  To reduce the risk of an OutOfMemoryError exception occurring due to an increase in use of the Metaspace area that was unforeseen during the estimation process, specify the estimated value plus a buffer in the -XX:MaxMetaspaceSize and -XX:CompressedClassSpaceSize options. An OutOfMemoryError exception will not occur until the usage of the Metaspace area exceeds this value.

  (b) Reducing the risk of the Metaspace area triggering Full GC

  To reduce the risk of triggering Full GC due to an increase in use of the Metaspace area that was unforeseen during the estimation process, specify the estimated value plus a buffer in the -XX:MetaspaceSize, -XX:MaxMetaspaceSize, and -XX:CompressedClassSpaceSize options. Usage of the Metaspace area will not trigger Full GC until this value is exceeded.

7.2.7 Relationship between GC occurrence and memory space

GC occurs when use of memory space meets certain conditions. This subsection explains the timing with which GC occurs.

> **Important note**
>
> If you use the RMI to register or reference a remote object, GC might occur periodically. You can specify the timing with which GC occurs in milliseconds in either of the following properties. The default value is `3600000` milliseconds (1 hour).
>
> - `sun.rmi.dgc.client.gcInterval` property
> - `sun.rmi.dgc.server.gcInterval` property
>
> To change the timing with which GC occurs, specify a value in milliseconds in either of the following properties. Note that you can specify a value in the range from 1 to Long.MAX_VALUE-1. For details, see the relevant page (`http://download.oracle.com/javase/6/docs/technotes/guides/rmi/sunrmiproperties.html`).

(1) Timing of occurrence of copy GC

Copy GC occurs in the following circumstances:

1. When there is insufficient space for allocation of the Eden area
2. When the `-copygc` option is specified in the `jheapprof` command and executed

(2) Timing of occurrence of Full GC

Full GC occurs in the following circumstances:

1. When the memory size being used in the New area (total of the Eden area and the Survivor area) exceeds the unused memory size for maximum value of the Tenured area, and the free space for allocation to the Eden area is insufficient

   > **Important note**
   >
   > The preceding circumstances will not always trigger Full GC. In these circumstances, the JavaVM determines whether to execute Full GC based on the following value:
   >
   > - The average value is weighed and calculated according to the time when the copy GC occurred for the objects moved from the New area to the Tenured area during a past copy GC

2. When an attempt to move objects from the New area (total of the Eden area and the Survivor area) to the Tenured area fails as a result of copy GC
3. When there is an allocation request of memory size (size of Java object) that exceeds the individual unused memory size of the New area and the Tenured area
4. When one of the following occurs as a result of the copy GC:
   - When the unused memory size of the allocated Tenured area falls below 10,000 bytes
   - When the allocated Tenured area is extended due to the transfer of objects to the Tenured area during the copy GC
5. When the `java.lang.System.gc()` method is executed

6. When the memory size to be allocated to the Metaspace area exceeds the unused memory size of the allocated Metaspace area.
7. When the `javagc` command is executed.
8. When the `jheapprof` command is executed.

Ensure that mainly above step 1 and step 3 do not occur while JavaVM is being tuned.

Reference note

You can use the extended `verbosegc` information for checking the occurrence factors of the Full GC. For details about how to check the factors for occurrence of Full GC, see *7.10 Analyzing the factors of Full GC using the extended verbosegc information*.

7.3 Overview of tuning to avoid Full GC

This section describes how to approach tuning of the Java heap and Explicit heap to avoid Full GC when using serial GC. It also describes how to perform this tuning.

The details of how to tune the Java heap and Explicit heap are described in *7.4* to *7.14*.

> **Reference note**
>
> Tuning does not differ depending on the difference in copy GC.

7.3.1 Concept of tuning

Usually, you can process copy GC in lesser time interval than Full GC.

You can avoid the occurrence of Full GC for the entire Java heap by implementing copy GC to New area and recovering adequate memory. However, avoiding the occurrence of full garbage collection is related to the reduction in stop frequency of system and improving the processing performance. To achieve these objectives, you must set appropriate size or ratio of memory space in the JavaVM options.

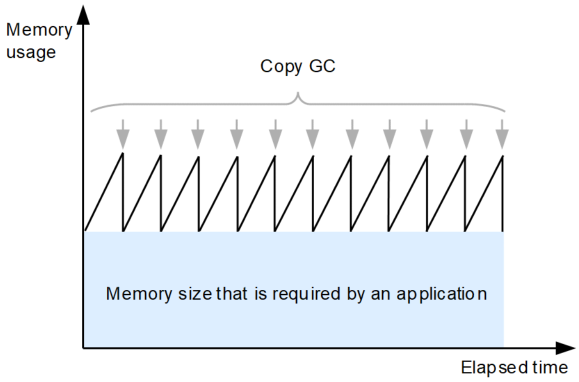
Another effective way to prevent Full GC from occurring is to allocate some of the objects in the Tenured area to the Explicit heap that is managed by the Explicit Memory Management functionality. The Explicit heap can be used from applications by using the automatic allocation configuration file or Explicit Memory Management APIs, and is also used from J2EE servers. The automatic release functionality is enabled with the default settings of the Explicit Memory Management functionality. When the automatic release functionality is enabled, JavaVM automatically calls back the Explicit heap area memory. The automatic release processing might not finish for a long time depending on how the Explicit heap is used. Therefore, the Explicit Memory Management functionality must be used appropriately and the Explicit heap memory size must be tuned suitably so that the automatic release processing, which does not end for a long time, does not occur. For details on the automatic release processing, see *7.7 Releasing Explicit memory blocks when the automatic release functionality is enabled* in the *uCosminexus Application Server Expansion Guide*.

Based on this information, you can implement JavaVM tuning for the following two points:

- Avoid the occurrence of Full GC to the possible extent
- To avoid occurrence of unwanted copy GC after stopping the frequent occurrence of Full GC

The following figure shows the relation of ideal memory usage and lapsed time.

Figure 7–8: Relation of ideal memory usage and lapsed time



In this figure, all the objects with short lifespan are recovered by copy GC and the objects are not expanded or deleted. Therefore, the memory size after executing copy GC is fixed. You can thereby implement operations in stable status without occurrence of Full GC.

In the JavaVM tuning, the tuning is performed by estimating memory size used in each area of memory space used by JavaVM, with this status as an ideal status.

> **Tip**
>
> **Tuning standards**
>
> Figure 7-8 shows an ideal example where Full GC does not occur even once. Actually, estimate the occurrence of copy GC 10 to 20 times for a single occurrence of Full GC and accordingly implement the tuning.

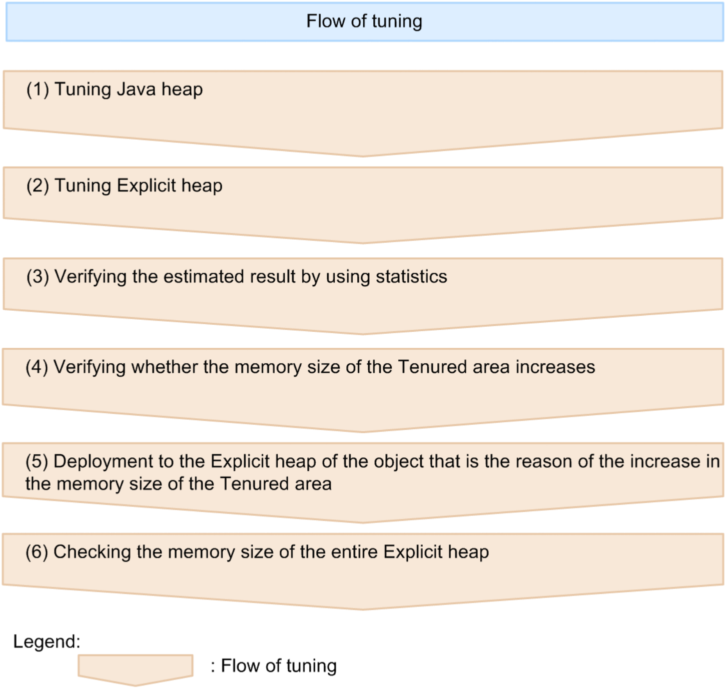
7.3.2 Tuning procedure

This subsection describes how to tune the Java heap and Explicit heap to prevent Full GC from occurring.

Execute the Step (1) always. Execute the Step (2) to (4) after Step (1) when you use the Explicit Memory Management functionality for Explicit heap. From step (5) onwards, check the description for the respective step and implement the step as and when required.

The following figure shows the tuning procedure of Java heap and Explicit heap:

Figure 7–9: Tuning procedure of Java heap and Explicit heap


Flow of tuning
(1) Tuning Java heap
(2) Tuning Explicit heap
(3) Verifying the estimated result by using statistics
(4) Verifying whether the memory size of the Tenured area increases
(5) Deployment to the Explicit heap of the object that is the reason of the increase in the memory size of the Tenured area
(6) Checking the memory size of the entire Explicit heap
Legend:
: Flow of tuning

(1) Java heap tuning

Consider how to prevent Full GC by tuning the Java heap. For details about the Java heap tuning, see *7.4 Java heap tuning*.

When you do not use the Explicit heap area, implement the J2EE server tests after performing Java heap tuning. If Full GC occurs frequently despite you appropriately estimating the memory size of the Java heap, consider whether there are any issues with the tuning such as the Survivor area becoming full. If the problem occurs even after reviewing the Java heap tuning, determine the usage of Explicit heap using the Explicit Memory Management functionality. Proceed with step (2) for using Explicit heap.

(2) Explicit heap tuning

To use the Explicit heap area by the Explicit Memory Management functionality, estimate the memory of the Explicit heap area. For details about the Explicit heap tuning, see *7.11 Explicit heap tuning*.

In J2EE servers, there are default settings to use the Explicit Memory Management functionality. Also, the JavaVM is configured to allocate the objects responsible for increasing the memory size of the Tenured area, such as objects related to the HTTP session, to the Explicit heap. Therefore, always estimate the Explicit heap memory size required for the objects allocated by J2EE server. However, it is not effective if the Explicit Memory Management functionality is used without properly estimating the memory size of Explicit heap.

(3) Confirming the estimation results from statistical information

When using the Explicit Memory Management functionality, execute the J2EE server tests after estimating the JavaVM memory properly in step (1) and step (2). Confirm the use status of Explicit heap by collecting the statistical information obtained in the tests. For details about estimating the Explicit heap based on statistical information, see *7.11.4 How to estimate using statistical information*.

Reference note

Execute the J2EE server tests and confirm the following points when occurrence of Full GC cannot be reduced.

- Whether there are any problems in Java heap tuning such as the Survivor area is full
  Check whether the value estimated in step (1) is a correct value.
- Whether the Explicit heap is full
  Check whether the value estimated in step (2) is a correct value.
- Whether the Web application configuration is correct
  Depending on the Web application configuration, (How to use APIs in applications) the Explicit Memory Management functionality for the objects related to the HTTP session might not be effective. For details, see *7.14 Precautions for using the Explicit Memory Management functionality* in the *uCosminexus Application Server Expansion Guide*.

If the Explicit Memory Management functionality is not effective even after you re-estimate the memory size of the Java heap and Explicit heap, and the automatic release processing is taking a long time, see *Appendix A Efficient Usage of the Explicit Heap Used in an HTTP session* to revise the application design. As an efficient usage of the Explicit heap, this appendix describes the points you might consider in an application design in order to efficiently apply the Explicit Memory Management functionality to the HTTP session-related objects and how to check the log for this purpose.

Proceed to step (4) if the problem is not yet resolved after all these confirmations.

(4) Confirming increase in memory size of the Tenured area

Start the application and check the memory size of the Tenured area. While checking, use the statistical information acquired in step (3) or the information acquired in extended verbosegc information. For details about how to acquire the extended verbosegc information, see *5.7.2 Acquiring the extended verbosegc information* in the *uCosminexus Application Server Maintenance and Migration Guide*.

(5) Allocating the objects responsible for increasing the memory size of the Tenured area to the Explicit heap

If steps (3) and (4) confirm that Full GC occurs frequently and the memory size of the Tenured area increases, consider avoiding Full GC by allocating the objects that are causing the increase in memory size to the Explicit heap. The objects to be studied here are not the "objects that a J2EE server allocates to the Explicit heap by default", but the other "objects created on a Java application". By allocating these objects to the Explicit heap instead of the Java heap, you can expect to reduce the frequency with which Full GC occurs. For details on how to identify the objects responsible for increasing the memory size of the Tenured area, see *7.13.2 When the cause of increase in the used size of the Tenured area is not known*.

The method of allocating the objects responsible for increasing the memory size of the Tenured area to the Explicit heap, includes the following two types:

- Using the APIs of the Explicit Memory Management functionality
- Using the automatic allocation functionality of the Explicit Memory Management functionality

For details on how to use the APIs of the Explicit Memory Management functionality, see 7*.12 Implementing a Java program using the APIs of the Explicit Memory Management functionality*, and for details on how to use the automatic allocation functionality of the Explicit Memory Management functionality, see *7.13.2 Using the Explicit Memory Management functionality by using the automatic placement configuration file* in the *uCosminexus Application Server Expansion Guide*.

If you use this functionality to allocate new objects to the Explicit heap, the memory size of the Explicit heap increases. Therefore, you must review the memory size of the Explicit heap again. Proceed to step (6).

(6) Reviewing memory size of entire Explicit heap

Run the application modified in step (5), and review the memory size of entire Explicit heap used by J2EE servers and applications. For details about the review methods, see *7.12 Estimating the memory size when using the Explicit Memory Management functionality in the application*.

Tip

To use the Explicit Memory Management functionality to effectively suppress the occurrence of Full GC, you need to ensure that objects do not overflow from the Explicit heap. Please confirm the following:

- Set Discard session (by invoking invalidate method) and appropriate session timeout in the Web application.
- The Explicit heap area of appropriate memory size can be allocated separately besides the Java heap area.

For details about the confirmation and measures when the Explicit heap overflow occurs, see *7.14.3 Checking and measures when there is an overflow from the Explicit heap*.

The further sections describe about the Java heap tuning and Explicit heap tuning. For notes other than those mentioned in this document, see *7. Suppression of Full GC by Using the Explicit Memory Management Functionality* in the *uCosminexus Application Server Expansion Guide*.

7.4 Java heap tuning

This section explains how to tune the Java heap when using serial GC.

7.4.1 How to estimate the memory size of Java heap

When tuning JavaVM, you need to properly estimate the memory size of each area of JavaVM specific area.

Estimate the following memory sizes:

- Memory size of the entire Java heap
- Memory size of the Tenured area
- Memory size of the Survivor area
- Memory size of the Eden area

In addition, estimate the Metaspace area, as and when required.

When the size of existing objects after executing copy GC is greater than the size of the Survivor area, the Survivor area becomes full and some of the objects are promoted to the Tenured area on one execution of copy GC. Also, when the size of the Survivor area is small and the usage of the Survivor area increases, the Java objects with a short lifespan (objects with a lifespan lesser than the copy GC interval or objects with a lifespan one to two times of the copy GC interval) are promoted to the Tenured area with multiple executions of copy GC.

Tip

If you observe any of the following issues, the Full GC was triggered by the Survivor area becoming full. Also, the extended `verbosegc` information is output to the JavaVM log file when GC occurs, if the option is set when starting the J2EE server.

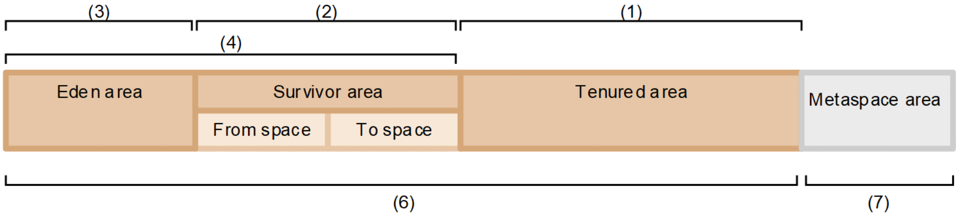
- When the objects with a short lifespan are identified as a reason for increase in the Tenured area
- When it is confirmed that the Survivor area is full during the execution of copy GC in the extended verbosegc information
- When it is observed that the age for promoting objects is always one in extended verbosegc information if specifying `-XX:+HitachiVerboseGCPrintTenuringDistribution`.

To avoid such problems, you must reduce the setup value of the `-XX:SurvivorRatio` option, and optimize the ratio of the Eden area and the Survivor area.

When estimating consider these points and first calculate the memory size of the Tenured area and the New area, and based on this memory size, calculate the memory size of the entire Java heap.

The following figure shows the order for estimating the memory size. Estimate in the sequence of the numbers given in the figure:

Figure 7–10: Order for estimating the memory size



The estimation procedures are given below. The numbers correspond to the numbers in the figure.

1. Estimate the memory size used in the Tenured area.

   For details about how to estimate, see *7.5 Estimating the memory size of the Tenured area in Java heap*.

2. Estimate the memory size used in the Survivor area.

   For details about how to estimate, see *7.6.1 Estimating the memory size of the Survivor area in Java heap*.

3. Estimate the memory size used in the Eden area.

   For details about how to estimate, see *7.6.2 Estimating the memory size of the Eden area in Java heap*.

4. Calculate the memory size of the entire New area by adding 2. and 3.

5. Determine the handling of objects that exist for a fixed time period, and add the necessary memory size to the memory size of either the Tenured area or the New area.

   For details about how to determine, see *7.7 Determining the handling of objects that exist for a fixed time period in Java heap*.

6. Calculate the memory size of entire Java heap by totaling 1, 4, and 5.

7. When required, estimate the memory size of the Metaspace area.

   For details about how to estimate, see *7.9 Estimating the memory size of the Metaspace area in Java heap*.

7.4.2 How to set the memory size in Java heap

Specify the estimated memory size with the options described in *7.2.6 Configuration of memory space used by JavaVM when using serial GC and JavaVM options*. Shown below is how to set the memory size of the respective areas:

Memory size of the entire Java heap

Specify the maximum size with the `-Xmxsize` option and then specify the initial size with the `-Xmssize` option.

Memory size of the Tenured area

Specify the division ratio of the Tenured area and the New area for the entire Java heap with the `-XX:NewRatio=`*value* option. For example, in the case of `-XX:NewRatio=5`, the memory size specified with the `-Xmxsize` option is divided as follows:

```
Memory size of the New area: Memory size of the Tenured area = 1:5
```

Memory size of the Survivor area and the Eden area

Specify the division ratio of the Survivor area and the Eden area for entire New area with the `-XX:SurvivorRatio=`*value* option. Specify the division ratio with a numeric value that indicates how many times the Eden area is to be allocated for the From space and the To space of the Survivor area. For example, in case of

`-XX:SurvivorRatio=`*2*, the memory size of the New area decided by the `-XX:NewRatio=`*value* option is divided as follows:

```
Memory size of the Eden area : Memory size of the From space : Memory siz
e of the To space = 2:1:1
```

Memory size of the Metaspace area

Specify the maximum size with the `-XX:MaxMetaspaceSize=`*size* option and then specify the initial size with the `-XX:MetaspaceSize=`*size* option.

7.4.3 How to check the usage of memory size of Java heap

Keep tuning the respective memory sizes while measuring the memory usage by actually running the application. In the Application Server, you can output the detailed memory size of each area when GC is executed in the form of *extended verbosegc information* by specifying the `XX:+HitachiVerboseGC` option in the `usrconf.cfg` file and invoking the J2EE server. Tune on the basis of this output.

The main contents that can be output in the form of extended verbosegc information are as follows:

- The date and time GC occurred
- GC type
- GC information[#1]
- Elapsed time of GC
- Eden information[#1]
- Survivor information[#1]
- Tenured information[#1]
- Metaspace area information[#1]
- GC factors[#2]

#1

The usage area length and the area size before and after GC are output.

#2

Output, when the -XX:+HitachiVerboseGCPrintCause option is specified.

For details about the output examples of extended verbosegc information and how to analyze the factors responsible for Full GC, see *7.10 Analyzing the factors of Full GC using the extended verbosegc information*. For details about the options, see *-XX:[+|-]HitachiVerboseGC (Option for extended verbosegc information output)* in the *uCosminexus Application Server Definition Reference Guide*.

You can also use the `javagc` command on the Application Server to trigger Full GC at any time. In this case, you can output the same contents as the extended verbosegc information, by specifying the `-v` option. For details about the `javagc` command, see *javagc (forcibly perform GC)* in the *uCosminexus Application Server Command Reference Guide*.

7.5 Estimating the memory size of the Tenured area in Java heap

This section explains how to estimate the memory size of the Tenured area when using serial GC.

Estimate the memory size of the Tenured area in the following manner:

```
Memory-size-of-the-Tenured-area
= memory-size-required-by-the-application + memory-size-of-the-New-area
```

Described below is how to calculate the memory size required by the application.

Also describes the reason for adding the memory size of the New area to estimated memory size.

7.5.1 Calculating the memory size required by an application

The memory size of the Tenured area is estimated based on the minimum memory size required by an application. If the required memory size cannot be allocated, `OutOfMemoryError` occurs and JavaVM stops.

The memory size required by an application can be decided based on the used memory size after Full GC that can be checked from the extended verbosegc information displayed during the execution of Full GC. This is based on the consideration that the memory size, after Full GC has deleted all unnecessary objects from the entire Java heap, is close to the memory size required by the application.

The following example of output shows the extended verbosegc information when Full GC is executed:

```
...
[VGC]Wed May 11 23:12:05 2005[Full GC 31780K->30780K(32704K), 0.2070500secs]
[DefNew::Eden: 3440K->1602K(3456K)][DefNew::Survivor:58K->0K(64K)][Tenured:
28282K->29178K(29184K)][Metaspace:3634K(4492K, 4492K)->3634K(4492K, 4492K)][
class space: 356K(388K, 388K)->356K(388K, 388K)][cause:ObjAllocFail][User: 0
.0156250 secs][Sys: 0.0312500 secs]
...
```

The information `->30780K` output after `Full GC` indicates that 30,780 KB of memory is required after GC is executed.

Collect the extended verbosegc information for several occurrences of Full GC. The largest value is the memory size required by the application.

7.5.2 Reason for adding the memory size of the New area in Java heap

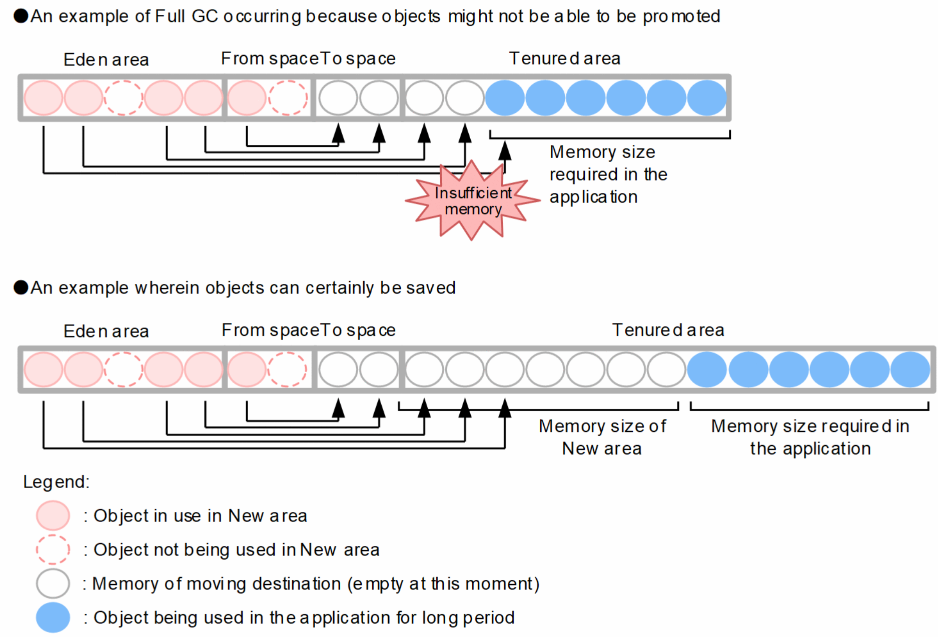
In the memory size of the Tenured area, Hitachi recommends you to add the memory size of the New area to the minimum required memory size of the application. This is to prevent frequent occurrence of Full GC, as the unused memory size of the Tenured area is lesser than the used memory size of the New area.

Usually, copy GC occurs when the Eden area becomes full. At that point, the Java objects in use that exist in the Eden area and the From space of the Survivor area try to move to the To space of the Survivor area. If, however, the unused area of the Tenured area is lesser than the memory size that is in use in the Eden area and the Survivor area, and all the Java objects of the New area are promoted, the Java objects in use do not move to the Tenured area. At this point, the JavaVM triggers Full GC and attempts to increase the unused memory size in the Tenured area.

To prevent this, apart from the memory size required by application, add the memory size equivalent to the New area, in the Tenured area.

The following figure shows this concept:

Figure 7–11: Reason for adding the memory size of the New area


●An example of Full GC occurring because objects might not be able to be promoted
Eden area
From space
To space
Tenured area
Memory size required in the application
Insufficient memory
●An example wherein objects can certainly be saved
Eden area
From space
To space
Tenured area
Memory size of New area
Memory size required in the application
Legend:
: Object in use in New area
: Object not being used in New area
: Memory of moving destination (empty at this moment)
: Object being used in the application for long period

- **An example where Full GC occurs because the objects might not be able to be promoted**
  The free space in the memory of the Tenured area (the area besides the memory area required by the application) is lesser than the memory size of the New area. Therefore, when the number of objects moving from the Eden area and From space is large, the objects cannot be promoted. In this scenario, Full GC occurs.
- **An example, wherein the objects can certainly be promoted**
  The free space in the memory of the Tenured area (the area besides the memory area required by the application) is allocated equivalent to the size same as the New area. Therefore, even when the number of objects moving from the Eden area and From space is large, the objects can be promoted.

For details about estimating the memory size of the New area, see *7.6 Estimating the memory size of the New area in Java heap*.

Tip

If checking the extended verbosegc information and other resources reveals that Full GC occurs frequently without copy GC occurring, you can conclude that the memory size of the Tenured area is too small for the objects promoted from the New area. This situation occurs when the size of the New area is increased. Modify the memory size of the Tenured area, as and when required. Also modify the relation between the Eden area and the Survivor area of the New area.

7.6 Estimating the memory size of the New area in Java heap

This section explains how to estimate the memory size of the New area when using serial GC.

Estimate the memory size of the New area as follows:

```
Memory-size-of-the-New-area
= memory-size-of-the-Survivor-area + memory-size-of-the-Eden-area
```

This section describes about how to estimate the memory size of the Survivor area and the Eden area.

7.6.1 Estimating the memory size of the Survivor area in Java heap

Tune the memory size of the Survivor area while checking the usage of the Survivor area, by actually running the application.

The flow of tuning is as follows:

1. Estimate the memory size used for the request and response processing in the application, and specify this memory size in the memory size of the Survivor area, and then execute the application.

   At this point, to output the information used for tuning, specify the -*XX:+HitachiVerboseGCPrintTenuringDistribution option* and invoke the J2EE server.

2. Check the memory usage based on the memory size allocated to the Survivor area and the actual memory usage when the application is running.

   When the memory usage approaches 100%, there is no longer enough space for in-use objects in the New area and the From space of the Survivor area to enter the To space when copy GC is executed. The objects are saved to the Tenured area instead. In such a case, consider increasing the Survivor area.

3. Check the age distribution of the objects of the Survivor area.

   If you increase the memory size of the Survivor area and threshold value required for promoting, it leads to delay in promoting the objects. Storing the objects with a long lifespan continuously in the Survivor area causes deterioration in the performance. On the other hand, if you reduce the memory size of the Survivor area, and reduce the threshold value required for promoting, the object promoting becomes faster. However, promoting objects with short lifespans can cause Full GC to occur more frequently.

   In such a case, consider balancing both the memory size of the Survivor area, and the threshold value for promoting the objects.

The respective tuning operations are explained below:

(1) Estimating the memory size used in the request and response processing

The Survivor area stores the objects with a short lifespan. In case of applications running on the server side, the Survivor area can be considered as an area that stores the objects with a short lifespan that are used for processing a request and response. As a result, for estimating the memory size of the Survivor area, consider the maximum size of objects with a short lifespan that exist at a certain point, or in other words, consider the maximum size of the memory used for processing a request and response, at a certain point of time. For example, in case of an application configured with stateless servlets, the memory size of the Survivor area can be considered as *the-maximum-memory-size-used-for-processing-one-request* × *the-number-of-concurrent-executions-of-the-request*.

(2) Checking the memory usage

Set the value that is estimated from *(1) Estimating the memory size used in the request and response processing*, as the memory size of the Survivor area, and then execute the application. Check the memory usage for memory size allocated to the Survivor area from the memory usage at runtime.

Reference note

You cannot directly specify the memory size of the Survivor area.

To specify the memory size of the Survivor area, you must first specify the maximum size of Java heap by the `-Xmx` option, then use the `-XX:NewRatio=`*value* option to specify ratio that divides the memory size of Java heap into the New area and the Tenured area, and finally specify the ratio with respect to the Eden area, by the `-XX:SurvivorRatio=`*value* option.

You can check the memory usage with the extended verbosegc information.

The following example of output shows the extended verbosegc information for copy GC at runtime:

```
...
[VGC]Wed May 11 23:12:05 2005[GC 27340K->27340K(32704K), 0.0432900 secs][Def
New::Eden: 3440K->0K(3456K)][DefNew::Survivor: 58K->64K(64K)][Tenured: 23841
K->27282K(29184K)][Metaspace: 3634K(4492K, 4492K)->3634K(4492K, 4492K)][clas
s space: 356K(388K, 388K)->356K(388K, 388K)][cause:ObjAllocFail][User: 0.015
6250 secs][Sys: 0.0312500 secs]
...
```

`DefNew::Survivor: 58K->64K(64K)` means *memory-size-before-GC-execution* `->` *memory-size-after-GC-execution (allocated-memory-size)*. In this case, 64 KB is already in use in the Survivor area of 64 KB, and therefore, the usage becomes 100%. This indicates insufficient memory size of the To space in copy GC, and the Java object is saved. This save process stores an object with a short lifespan that is not meant to be stored in the Tenured area, causing Full GC to occur more frequently. In such a case, consider increasing the memory size of the Survivor area. Estimate the memory size of the new Survivor area in the following manner:

```
Memory-size-of-the-new-Survivor-area
= memory-size-of-the-existing-Survivor-area + total-size-of-the-saved-Java-o
bject
```

The *total-size-of-the-saved-Java-object* can be approximated with the increased memory size of the Tenured area after the GC is executed. For example, `Tenured: 23841K->27282K(29184K)` indicates the increased memory size of the Tenured area, and becomes `27,282 KB - 23,841 KB =3,441 KB`.

If you increase the memory size of the Survivor area, the threshold value required for promoting the Java objects increases, and thereby, the object promotion becomes difficult. For details, see *(3) Checking and estimating the age distribution of objects*. As a result, the number of Java objects that are not collected by copy GC increases and hence, estimate and setup the `-XX:TargetSurvivorRatio=`*value* in the following manner:

```
New -XX:TargetSurvivorRatio=<value>
= existing--XX:TargetSurvivorRatio=<value> × (memory-size-of-the-existing-Su
rvivor-area-and-memory-size-of-the-new-Survivor-area)
```

(3) Checking and estimating the age distribution of objects

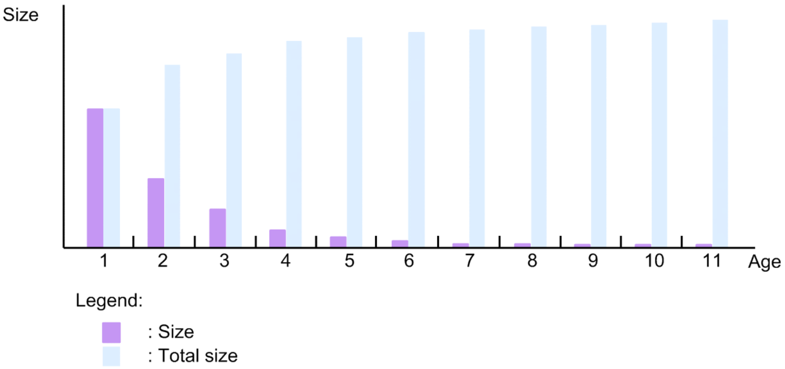
Check the age distribution of the objects of the Survivor area, and ensure that the objects with a long lifespan do not continue to exist, or the objects with a short lifespan are not being promoted. You can check the age distribution of objects with the help of the output results of the `-XX:+HitachiVerboseGCPrintTenuringDistribution` option.

If you specify the `-XX:+HitachiVerboseGCPrintTenuringDistribution` option in `usrconf.cfg` when invoking the J2EE server, the used state of the Survivor area is output to the Java VM log file when copy GC occurs. The output example is as follows:

```
[PTD]Wed May 28 11:45:23 2008[Desired survivor:5467547 bytes][New threshold:
3][MaxTenuringThreshold:31][age1:1357527][age2:1539661]
```

Following the `New threshold`, the minimum age of the Java objects promoted by the next copy GC is output. In this example, the Java object whose age is 3 or more is promoted by the next copy GC. Following the `age<numeric value>:` is the total of memory size used by the Java objects, whose age is between one and that age in the Survivor area. In this example, if the memory size of the Java object with age 1 is 1,357,527 bytes, total 1,539,661 bytes of the memory size of both Java objects with age 1 and 2 is displayed. Also, it is concluded that the memory size of the Java object with age 2 is 182,134 (`1,539,661-1,357,527`) bytes if calculated back from the total. Generally, the age distribution of the object in the Survivor area is similar to the following graph:

Figure 7–12: Age distribution of the objects in the Survivor area



The 'Size' in the graph is a total size of the Java object of a certain 'Age'. Also, the 'Total size' is a total size of the Java objects up to a certain 'Age'.

In this graph, the size that the Java object in the Survivor area occupies, decreases as the age increases. Also, the size that is reduced when age is increased by 1 year, becomes larger, as lesser the age, and based on this, the following conclusions are drawn:

- The size occupied in the Survivor area becomes larger, as lesser the age of the Java object
- Easy to collect the object by GC, as lesser the age of the Java object

In this example, the Java object with age 6 or above is hardly collected by the copy GC. If the threshold value for promoting a Java object is age 7 or greater, copy GC is applied to Java objects with a low probability of collection which reduces system performance. On the other hand, if the threshold value for the Java object promoting is less than age 2, the object having higher possibility of being collected by the copy GC can be promoted, and this causes increase in the occurrences of Full GC. In this example, the threshold value for promoting is between 5 to 6 age that implies a balanced threshold value.

The trend of the graph differs depending on the system, and therefore, you must check the age distribution of the objects in the Survivor area, and decide a suitable promoting age of the objects for each system.

Reference note

The threshold value for the Java object promoting changes dynamically for each copy GC, and is decided based on the `-XX:MaxTenuringThreshold=`*value* option, and the memory size of the Survivor area and the value set in the `-XX:TargetSurvivorRatio=`*value* option. The `-XX:MaxTenuringThreshold=`*value* option is the maximum age of the threshold value required for promoting. Whenever the age of the Java object exceeds this value, the Java objects are promoted without fail. The `-XX:TargetSurvivorRatio=`*value* option is a target value of the memory usage of the Survivor area. To set the memory usage of the Survivor area closer to the target value, the JavaVM decides the threshold value required for object promoting. Specifically, searches for `n` that indicates the target usage of the total size of Java object from age `1` to `n` when the copy GC is completed, and sets the threshold value for promoting in next copy GC, to `n`. The default value of `-XX:TargetSurvivorRatio=`*value* is 50%. If the memory usage of the Survivor area is greater, you can use the Survivor area more effectively, and therefore, it is easy to save the objects as the free space of the Survivor area is not available.

7.6.2 Estimating the memory size of the Eden area in Java heap

The memory size of the Eden area affects the interval between copy GC operations. If the memory size of the Eden area is large, the interval of occurrence of copy GC becomes long. The time taken for the copy GC is affected by the number of objects in use, but is not much affected by the memory size of the Eden area. For this reason, allocate enough memory size for the Eden area to prevent a frequent occurrence of the copy GC. This is effective for improving performance.

7.7 Determining the handling of objects that exist for a fixed time period in Java heap

The explanation in the previous sections is based on the assumptions that the objects are to be saved in the following way in their respective areas, depending on the lifespan of the objects:

- Save the objects with a long lifespan that are necessary for the operations of the applications in the Tenured area.
- Save the objects with a short lifespan that are used for the request and response processing in the New area.

However, there are objects used for a fixed time period with an intermediate lifespan. Although such objects do not have a long lifespan, copy GC is executed a number of times for them.

When estimating the memory size, you must estimate assuming that these objects are to be stored either in the New area or the Tenured area.

The respective features are shown below. Depending on the type of the application and the purpose, estimate which memory size is to be increased.

7.7.1 How to save in the New area in Java heap

This subsection describes how to manage objects that exist for a fixed time period, in the New area. In the memory size of the New area, add the memory size for these objects that exist for a fixed time, and then estimate.

You can make Full GC less likely to occur by increasing the size of the New area to limit the movement of objects to the Tenured area. However, since the number of objects in use within the New area increases when executing copy GC, the Copy processing within the New area takes time, and the time taken for executing one copy GC increases. If the execution time of copy GC is longer than the execution time of Full GC, you need to re-estimate the memory size. Depending on the size setting of the memory space, there might not be enough space to accommodate the objects with short lifespans that would usually be collected by copy GC, potentially triggering the saving of objects to the Tenured area. This will ultimately cause Full GC to occur.

You can use the extended verbosegc information to check whether the objects that exist for a fixed time period can be managed in the New area. Run the application and with the help of extended verbosegc information that is output, ensure that the memory size of the Tenured area does not increase immensely after the occurrence of copy GC.

In case of failure in managing the objects in the New area, the system performance declines immensely. Also, there are limitations for the maximum age of objects managed in the New area (Limitations differ according to the platform and version. For details, see `-XX:MaxTenuringThreshold=`*value* of *14.4 Default values of the Java HotSpot VM options that can be specified in Cosminexus* in the *uCosminexus Application Server Definition Reference Guide*.). If you realize that the objects are not being managed in New area, consider saving and managing the objects that exist for a fixed time period, in the Tenured area. For details about how to manage objects in the Tenured area, see *7.7.2 How to save in the Tenured area in Java heap*.

7.7.2 How to save in the Tenured area in Java heap

You can use the `-XX:MaxTenuringThreshold=`*value* option for setting the maximum age of objects that are managed in the New area. For example, if you specify `-XX:MaxTenuringThreshold=`*2*, all the objects for which the third copy GC is executed, move to the Tenured area.

If you use this method, fewer objects are subject to copy GC and execution time is reduced. However, because a large number of objects are moved to the Tenured area, Full GC will routinely occur when the Tenured area becomes full. To ensure the stable operation of the system, forcibly trigger Full GC at appropriate times, such as when the load on the system is low. You can trigger Full GC using the following methods:

- Call the `System.gc()` method within the program.
- Execute the `javagc` command.

  For details about the `javagc` command, see *javagc (forcibly perform GC)* in the *uCosminexus Application Server Command Reference Guide*.

7.7.3 How to save in the Explicit heap

This subsection describes how to save and manage the objects that exists for a fixed time period, in the Explicit heap by changing Java programs and not by performing tuning. The Explicit heap is an area that is not subject to GC. By using the Explicit heap, you can prevent objects from moving to the Tenured area and make Full GC occur less often. You can also use automatic allocation setup file to allocate the object directly in Explicit heap. For details, see *7.11 Explicit heap tuning*. On Application Server, the objects related to HTTP session are managed by Explicit heap.

7.8 Deciding the maximum size or the initial size of Java heap

If you have estimated the Tenured area and the New area, then based on the estimated area, decide the maximum size and the initial size of Java heap.

Decide the maximum size of Java heap in the following manner:

```
Maximum-size-of-Java-heap
= memory-size-of-the-Tenured-area + memory-size-of-the-New-area
```

When setting the memory size of Java heap, first, specify the maximum size of Java heap, including the size of the extended area, in the `-Xmx` option. Next, specify the initial size of the Java heap in the `-Xms` option. The size specified in the `-Xms` option has to be lesser than that specified in the `-Xmx` option.

At the time of startup, JavaVM allocates only that much memory area as the Java heap that has been specified in the `-Xms` option. Later, if memory area more than what is specified in the `-Xms` option is required during execution of an application, JavaVM keeps adding and allocating the heap area until it reaches the size specified in the `-Xmx` option. Conversely, if there is some unnecessary memory space in an application, then JavaVM keeps reducing the area that has been allocated as the Java heap, until it reaches the size specified in the `-Xms` option.

For stable operations of a system, Hitachi recommends that you specify the same value in the `-Xmx` option and the `-Xms` option.

7.9 Estimating the memory size of the Metaspace area in Java heap

This section explains how to estimate the memory size of the Metaspace area when using serial GC. The Metaspace area stores loaded classes and other objects.

As mentioned in *7.2.6 Configuration of memory space used by JavaVM when using serial GC and JavaVM options*, the memory size of the Metaspace area is allocated separately from the memory size specified by the `-Xmx` option (Java heap).

For details about the default values, see *14.4 Default values of the Java HotSpot VM options that can be specified in Cosminexus* in the *uCosminexus Application Server Definition Reference Guide*.

The method for estimating the usage of the Metaspace area is as follows:

- **Method of estimating the usage of the Metaspace area**

  The memory usage in the Metaspace area is almost the total size of the class files loaded in the J2EE server. In case of Application Server, you can perform estimations on the basis of the sum of following class file size:

  1. All the class files under `WEB-INF/classes`
  2. All the class files included in the JAR files under `WEB-INF/lib`
  3. All the class files generated as a result of JSP compilation
  4. All the class files included in EJB-JAR
  5. If container extension library, library JAR, and reference library are being used, then all the class files included in the JAR files that are added
  6. Class files loaded by containers

     Metaspace-area-after-application-starts - *Metaspace-area-before-application-is-registered*. Calculate this value by actually running the J2EE server and then checking the Metaspace area.
  7. Class files provided by the Application Server (system class files)

     The total size of the system class files is approximately 140 MB.
  8. Class files provided by JDK

     The total size of the class files provided by JDK is approximately 110 MB.
  9. Dynamic class generated by reflection processing

     Reflection processing dynamically generates calling classes to increase processing speed.

     Determine the size of this class by measuring how much the size of the Metaspace area increases after the server has been running for a reasonable period of time.

  Specify the memory size of the Metaspace area in the `-XX:MaxMetaspaceSize=`*size* and `-XX:MetaspaceSize=`*size* options. For details about the default values of these options, see *14.4 Default values of the Java HotSpot VM options that can be specified in Cosminexus* in the *uCosminexus Application Server Definition Reference Guide*.

  Note that the usage of the Metaspace area might temporarily increase while importing the application.

Reference note

Resolution method when the Metaspace area is compressed due to software references in the development environment

Due to software references, the release of the Metaspace area might be delayed when an application is un-deployed. Therefore, when you repeatedly deploy and un-deploy applications in the development environment, the Metaspace area might be compressed due to the delay in the release of the Metaspace area. Specify the following option to resolve the compression of the Metaspace area due to software references:

- `-XX:SoftRefLRUPolicyMSPerMB=0`

If you specify `0` in the `-XX:SoftRefLRUPolicyMSPerMB` option, all software references are disabled. A software reference is often used as a cache for performance improvement, so the performance of an application might deteriorate due to the specification of this option. Therefore, specify this option only when the Metaspace area is compressed in the development environment.

7.10 Analyzing the factors of Full GC using the extended verbosegc information

This section explains how to use the extended verbosegc information to analyze the factors that contribute to Full GC being triggered when using serial GC. *The extended* `verbosegc` information is the log information of JavaVM that can be output by specifying the `-XX:+HitachiVerboseGC` option that is a JavaVM option. In addition to the information that is useful for tuning, the information that is also useful for analyzing the cause of errors, is output.

You can check the following information, by referring to the extended verbosegc information when tuning:

- Memory size used by each area before and after GC is executed
- Factors that contribute to GC occurring

You can output further detailed information by combining the `-XX:+HitachiVerboseGC` option with another JavaVM option. For details about the `-XX:+HitachiVerboseGC` option and other JavaVM extended options, see *14.2 Details of JavaVM extension options* in the *uCosminexus Application Server Definition Reference Guide*.

7.10.1 Overview of the output format of extended verbosegc information

The extended verbosegc information is output when copy GC occurs and when Full GC occurs.

When copy GC occurs, `GC` is output as the GC type in the extended verbosegc information. When Full GC occurs, `Full GC` is output as the GC type. Following the type, the extended verbosegc information shows the value of *area-size-before-GC* - *area-size-after-GC* (*allocated-area-size*) for each area.

The following are examples of the extended verbosegc information output when Full GC occurs. The factors responsible for occurrence of the GC and the CPU time of the GC thread are also output to the extended verbosegc information.

For details about the output format of the extended verbosegc information and the respective options, see *14. Options for Invoking JavaVM* in the *uCosminexus Application Server Definition Reference Guide*.

7.10.2 Examples of extended verbosegc information output when Full GC occurs

This subsection explains the output of extended `verbosegc` information when Full GC occurs.

(1) An example of output when the memory size being used in the New area (total of the Eden area and the Survivor area) exceeds the unused memory size for the maximum value of the Tenured area

An example of output of extended verbosegc information is shown below. Parts that are highlighted and **bolded** indicate factors that contribute to Full GC occurring.

```
...
[VGC]<Wed May 11 23:12:05 2005>[GC 27340K->27340K(32704K), 0.0432900 secs][D
efNew::Eden: 3440K->0K(3456K)][DefNew::Survivor: 58K->58K(64K)][Tenured: 238
41K->27282K(29184K)][Metaspace: 3634K(4492K, 4492K)->3634K(4492K, 4492K)][cl
ass space: 356K(388K, 388K)->356K(388K, 388K)][cause:ObjAllocFail][User: 0.0
156250 secs][Sys: 0.0312500 secs]
[VGC]<Wed May 11 23:12:05 2005>[Full GC 30780K->30780K(32704K), 0.2070500 se
cs][DefNew::Eden: 3440K->1602K(3456K)][DefNew::Survivor: 58K->0K(64K)][Tenur
ed: 27282K->29178K(29184K)][Metaspace: 3634K(4492K, 4492K)->3634K(4492K, 449
2K)][class space: 356K(388K, 388K)->356K(388K, 388K)][cause:ObjAllocFail][Us
er: 0.0156250 secs][Sys: 0.0312500 secs]
...
```

The following conclusions are drawn from this output example:

- The memory size being used in the New area (3440 K + 58 K=3498 K) has exceeded the unused memory size for the maximum value of the Tenured area (29184 K - 27282 K = 1902 K).
- The cause of Full GC is failure to allocate the object.

(2) An example of output when copy GC fails to move objects from the New area (the Eden area and the Survivor area) to the Tenured area

The following is an output example of the extended verbosegc information. The part that is highlighted and **bolded** indicates a factor that contributes to Full GC occurring.

```
...
[VGC]<Thu Oct 20 11:04:42 2011>[GC 26418K->26418K (29696K), 0.0000000 secs][
DefNew::Eden:8188K->8188K(8192K)][DefNew::Survivor: 1021K->1021K(1024K)][Ten
ured:17208K->17208K (20480K)][Metaspace: 3634K(4492K, 4492K)->3634K(4492K, 4
492K)][class space: 356K(388K, 388K)->356K(388K, 388K)][cause:ObjAllocFail][
User: 0.0000000 secs][Sys: 0.0000000 secs][IM: 877K, 1104K, 0K][TC: 9][DOE:
0K, 0]
[VGC]<Thu Oct 20 11:04:42 2011>[Full GC 26418K->6450K(29696K), 0.0156250 s
ecs][DefNew::Eden:8188K->0K(8192K)][DefNew::Survivor:1021K->0K(1024K)][Tenur
ed:17208K->6450K(20480K)][Metaspace: 3634K(4492K, 4492K)->3634K(4492K, 4492K
)][class space: 356K(388K, 388K)->356K(388K, 388K)][cause:PromotionFail][Use
r: 0.0156250 secs][Sys: 0.0000000 secs][IM: 925K, 1104K, 0K][TC: 9][DOE: 0K
, 0]
```

The following conclusion is drawn from this output example:

- The cause of Full GC is failure to move objects from the New area to the Tenured area due to copy GC.

(3) An example of output when the memory size to be allocated (size of the Java objects created by new) exceeds the unused memory size of the Tenured area

An example of output of extended verbosegc information is shown below. Parts that are highlighted and **bolded** indicate factors that contribute to Full GC occurring.

```
...
[VGC]<Wed May 11 23:53:18 2005>[GC 28499K->28490K(32704K), 0.0540590 secs][D
efNew::Eden: 808K->0K(3456K)][DefNew::Survivor: 64K->62K(64K)][Tenured: 2762
6K->28428K(29184K)][Metaspace: 3634K(4492K, 4492K)->3634K(4492K, 4492K)][cla
ss space: 356K(388K, 388K)->356K(388K, 388K)][cause:ObjAllocFail][User: 0.01
56250 secs][Sys: 0.0312500 secs]
[VGC]<Wed May 11 23:53:18 2005>[Full GC 28490K->8959K(32704K), 0.1510380 sec
s][DefNew::Eden: 0K->0K(3456K)][DefNew::Survivor: 62K->0K(64K)][Tenured: 284
28K->8959K(29184K)][Metaspace: 3634K(4492K, 4492K)->3634K(4492K, 4492K)][cla
ss space: 356K(388K, 388K)->356K(388K, 388K)][cause:ObjAllocFail][User: 0.01
56250 secs][Sys: 0.0312500 secs]
...
```

The following conclusions are drawn from this example of output:

- An attempt was made to create a Java object whose memory size exceeds the unused memory size of the Tenured area (`29184 K - 28428 K = 756 K`) with `new`.
- The cause of Full GC is a failure to allocate objects.

(4) An example of output when copy GC causes the unused memory size of the Tenured area to fall below 10,000 bytes

The following is an output example of the extended verbosegc information. Parts that are highlighted and **bolded** indicate factors that contribute to Full GC occurring.

```
...
[VGC]<Fri May 25 15:21:33 2007>[GC 15436K->15416K(19840K), 0.0111825 secs][D
efNew::Eden: 4413K->0K(4416K)][DefNew::Survivor: 512K->509K(512K)][Tenured:
10511K->14906K(14912K)][Metaspace: 3634K(4492K, 4492K)->3634K(4492K, 4492K)]
[class space: 356K(388K, 388K)->356K(388K, 388K)][cause:ObjAllocFail][User:
0.0000000 secs][Sys: 0.0000000 secs]
[VGC]<Fri May 25 15:21:33 2007>[Full GC 15416K->8622K(19840K), 0.0284614 sec
s][DefNew::Eden: 0K->0K(4416K)][DefNew::Survivor: 509K->0K(512K)][Tenured: 1
4906K->8622K(14912K)][Metaspace: 3634K(4492K, 4492K)->3634K(4492K, 4492K)][c
lass space: 356K(388K, 388K)->356K(388K, 388K)][cause:ObjAllocFail][User: 0.
0312500 secs][Sys: 0.0000000 secs]
...
```

The following conclusions are drawn from this example of output:

- The used memory size of the Tenured area increased from 10,511 KB to 14,906 KB due to the transfer of objects from the New area to the Tenured area in the first copy GC. As a result, the unused memory size of the allocated Tenured area became `14,912 KB - 14,906 KB = 6 KB` and fell below 10,000 bytes (approximately 10 K bytes).
- The reason for the first copy GC is the object allocation failure. The first copy GC and second Full GC occur consecutively before the control returns to the Java program.

(5) An example of output when the allocated Tenured area is extended due to the transfer of objects to the Tenured area during copy GC

An example of output of extended verbosegc information is shown below. Parts that are highlighted and **bolded** indicate factors that contribute to Full GC occurring.

```
...
[VGC]<Fri May 25 15:42:00 2007>[GC 12745K->10151K(15872K), 0.0048346 secs][D
efNew::Eden: 4416K->0K(4416K)][DefNew::Survivor: 137K->512K(512K)][Tenured:
8192K->9639K(10944K)][Metaspace: 3634K(4492K, 4492K)->3634K(4492K, 4492K)][c
lass space: 356K(388K, 388K)->356K(388K, 388K)][cause:ObjAllocFail][User: 0.
0156250 secs][Sys: 0.0000000 secs]
[VGC]<Fri May 25 15:42:00 2007>[GC 14563K->14536K(19072K), 0.0104957 secs][D
efNew::Eden: 4412K->0K(4416K)][DefNew::Survivor: 512K->510K(512K)][Tenured:
9639K->14026K(14144K)][Metaspace: 3634K(4492K, 4492K)->3634K(4492K, 4492K)][
class space: 356K(388K, 388K)->356K(388K, 388K)][cause:ObjAllocFail][User: 0
.0156250 secs][Sys: 0.0000000 secs]
[VGC]<Fri May 25 15:42:00 2007>[Full GC 14536K->8610K(19072K), 0.0287254 sec
s][DefNew::Eden: 0K->0K(4416K)][DefNew::Survivor: 510K->0K(512K)][Tenured: 1
4026K->8610K(14144K)][Metaspace: 3634K(4492K, 4492K)->3634K(4492K, 4492K)][c
lass space: 356K(388K, 388K)->356K(388K, 388K)][cause:ObjAllocFail][User: 0.
0312500 secs][Sys: 0.0000000 secs]
...
```

The following conclusions are based on this example of output:

- The Tenured area must be at least 14,026 KB or more due to the movement of objects from the New area to the Tenured area in the second copy GC. Therefore, the size of the allocated Tenured area is extended from 10,944 KB to 14,144 KB.
- The reason for the second copy GC is object allocation failure. The second copy GC and third Full GC occur consecutively before the control returns to the Java program.

(6) An example of output when the java.lang.System.gc() method is executed in an application

An example of output of extended verbosegc information is shown below. The part that is highlighted and **bolded** indicates a factor that contributes to Full GC occurring.

```
...
[VGC]<Mon Apr 18 20:36:29 2005>[Full GC 330K->150K(3520K), 0.0387690 secs][
DefNew::Eden: 330K->0K(2048K)][DefNew::Survivor: 0K->0K(64K)][Tenured: 0K->1
50K(1408K)][Metaspace: 3634K(4492K, 4492K)->3634K(4492K, 4492K)][class space
: 356K(388K, 388K)->356K(388K, 388K)][cause:System.gc][User: 0.0156250 secs]
[Sys: 0.0312500 secs]
...
```

The following conclusion is drawn from this example of output:

- The cause of Full GC is calling the `java.lang.System.gc()` method in J2EE applications or batch applications.

(7) An example of output when the memory size to be allocated to the Metaspace area exceeds the unused memory size of the allocated Metaspace area

An example of output of extended verbosegc information is shown below. The part that is highlighted and **bolded** indicates a factor that contributes to Full GC occurring.

```
...
[VGC]<Wed Jan 07 01:56:13 2015>[Full GC 11273K->6037K(15872K), 0. 0060004 se
cs][DefNew::Eden: 442K->0K(4416K)][DefNew::Survivor: 512K->0K(512K)][Tenured
: 10319K->6037K(10944K)][Metaspace: 22811K(24520K, 24576K)->22811K(24520K, 2
4576K)][class space: 10758K(10988K, 11008K)->10758K(10988K, 11008K)][cause:M
etaspaceAllocFail][User: 0.0312002 secs][Sys: 0.0000000 secs]
...
```

The following conclusions are drawn from this example of output:

- The memory size to be allocated to the `Metaspace` area exceeded the unused memory size of the allocated `Metaspace` area (`24576 K - 22811 K = 1765 K`).
- Full GC was caused by a failure to allocate sufficient memory to the Metaspace heap.

(8) An example of output for the case when the javagc command is executed

An example of output of extended verbosegc information is shown below. The part that is highlighted and **bolded** indicates a factor that contributes to Full GC occurring.

```
...
[VGC]<Mon Apr 18 21:46:50 2005>[Full GC 369K->189K(3520K), 0.0403010 secs][
DefNew::Eden: 369K->0K(2048K)][DefNew::Survivor: 0K->0K(64K)][Tenured: 0K->1
89K(1408K)][Metaspace: 3634K(4492K, 4492K)->3634K(4492K, 4492K)][class space
: 356K(388K, 388K)->356K(388K, 388K)][cause:JavaGC Command][User: 0.0156250
secs][Sys: 0.0312500 secs]
...
```

The following conclusion is drawn from this example of output:

- The cause of Full GC is execution of the `javagc` command.

(9) An example of output for the case when the jheapprof command is executed

An example of output of extended verbosegc information is shown below. The part that is highlighted and **bolded** indicates a factor that contributes to Full GC occurring.

```
...
[VGC]<Mon Apr 18 21:46:50 2005>[Full GC 369K->189K(3520K), 0.0403010 secs][
DefNew::Eden: 369K->0K(2048K)][DefNew::Survivor: 0K->0K(64K)][Tenured: 0K->1
89K(1408K)][Metaspace: 3634K(4492K, 4492K)->3634K(4492K, 4492K)][class space
: 356K(388K, 388K)->356K(388K, 388K)][cause:JHeapProf Command][User: 0.01562
50 secs][Sys: 0.0312500 secs]
...
```

The following conclusion is drawn from this example of output:

- The cause of Full GC is execution of the `jheapprof` command.

7.11 Explicit heap tuning

This section describes about the Explicit heap tuning.

7.11.1 How to estimate the memory size of Explicit heap (Estimating memory size used in J2EE server)

The settings to use the Explicit Memory Management functionality are required as a prerequisite for tuning Explicit heap. The Explicit Memory Management functionality is enabled when `-XX:+HitachiUseExplicitMemory` is specified as the start option of JavaVM. In J2EE servers, there are default settings to use the Explicit Memory Management functionality. Also, the objects responsible for increasing the memory size of the Tenured area are set to be allocated in Explicit heap. Therefore, always estimate the memory size of Explicit heap required for the objects allocated by J2EE servers.

If the memory size of Explicit heap is not estimated properly, the Explicit Memory Management functionality is not effective.

In a J2EE server, objects related to the HTTP session that cause the memory size of the Tenured area to increase are deployed in the Explicit heap.

As a result of estimating the memory size of the Explicit heap, if the memory size of the Explicit heap used by the HTTP session-related objects is extremely large, see *Appendix A Efficient Usage of the Explicit Heap Used in an HTTP session* to revise the application design.

7.11.2 Memory size used by the object related to the HTTP session

This section describes about how to estimate the memory size used by the object related to the HTTP session.

Important note

Note that by this procedure you cannot estimate the memory size when using the functionality for reducing Explicit heap memory size utilized by an HTTP session. Follow the procedure described in *7.11.4 How to estimate using statistical information* for the memory size if you are using functionality for reducing Explicit heap memory size utilized by an HTTP session.

The memory size of the Explicit heap used by HTTP session-related objects is estimated with the following formula:

```
Memory-size-of-the-Explicit-heap-used-by-HTTP-session-related-objects
= Memory-size-of-the-Explicit-heap-used-in-the-HTTP-session
 + Memory-size-of-the-Explicit-heap-area-used-in-the-Web-container
```

The memory size of the Explicit heap used by the HTTP session is estimated with the following formula:

```
Memory-size-of-the-Explicit-heap-used-in-the-HTTP-session
= Memory-size-used-in-one-session#1 × Number-of-sessions-required-in-the-syst
em
```

The memory size of the Explicit heap area used in the Web container for the HTTP session is estimated with the following formula:

```
Memory-size-of-the-Explicit-heap-area-used-in-the-Web-container
= Size-of-the-objects-used-for-managing-the-HTTP-sessions#2 × (Number-of-Web
-applications#3 + 2)
```

#1

Memory-size-used-in-one-session is equivalent to the size of one Explicit memory block. To estimate the size of one Explicit memory block, actually run the application and check the usage of the Explicit heap. Note that the minimum size of the Explicit memory block differs depending on the use of the automatic allocation functionality of the Explicit Memory Management functionality. The following table describes the minimum size of the Explicit memory block depending on the use of the automatic allocation functionality of the Explicit Memory Management functionality.

Table 7–4: Minimum size of the Explicit memory block depending on the use of the automatic allocation functionality of the Explicit Memory Management functionality

Item number	Whether the automatic allocation functionality of the Explicit Memory Management functionality is used	Minimum size of Explicit memory block
1	Y	16 kilobytes
2	--	64 kilobytes

Legend:

Y: Automatic allocation functionality of the Explicit Memory Management functionality is used.

--: Automatic allocation functionality of theExplicit Memory Management functionality is not used.

You can extend the Explicit memory block in 64 kilobyte units. Therefore, the size would be "s*ize-of-one-Explicit-memory-block* ≥ *memory-size-used-in-one-session*". Also, estimate the size by adding 16 kilobytes when using the automatic allocation functionality of theExplicit Memory Management functionality.

#2

Size-of-the-objects-used-for-managing-the-HTTP-sessions is the minimum size of the Explicit memory block described in *Table 7-4*.

#3

Number-of-Web-applications shows the number of running Web applications.

(1) Estimation procedure

The estimation procedure is as follows:

1. Start the application that creates the HTTP session, and then execute the processes up to the last process executed immediately before the session annulment (such as logout).
2. Trigger Full GC by executing the `javagc` command.
3. Check the information output to the event log of the Explicit Memory Management functionality after Full GC is executed.
   The event log of the Explicit Memory Management functionality is output to a file or directory specified in the `-XX:HitachiExplicitMemoryJavaLog` option which is the startup option of JavaVM. The default output destination is as follows:

In Windows

Cosminexus-installation-directory`\CC\server\public\ejb\`*J2EE-server-name*`\logs\ehjavalog`*n*`.log`

In UNIX

`/opt/Cosminexus/CC/server/public/ejb/`*J2EE-server-name*`/logs/ehjavalog`*n*`.log`

Check the size used in Explicit heap and the number of Explicit memory blocks required in the series of businesses.

While calculating the memory size, check the following items from among the output items of the event log:

- Allocated size of the Explicit heap
- Number of Explicit memory blocks

For details about the contents of the Explicit Memory Management functionality event logs, see *5.11 Event log of Explicit Memory Management functionality* in the *uCosminexus Application Server Maintenance and Migration Guide*. These items are output when GC occurs and are included in use status of Explicit heap.

4. Based on contents checked in step (3), calculate the size of one Explicit memory block.

   You can calculate the size using the following formula:

   ```
   Size-of-one-Explicit-memory-block
   =allocated-size-of-Explicit-heap/number-of-Explicit-memory-blocks
   ```

   size-of-one-Explicit-memory-block is equivalent to the memory size used in one session.

5. Calculate the total memory size of Explicit heap used in the HTTP session by multiplying the value calculated in step (4) with the number of sessions required in business.

(2) Estimation example

This point shows an estimation example based on the output example of event log of the Explicit Memory Management functionality. An output example of the Explicit Memory Management functionality is as follows:

Output example of event log of the Explicit Memory Management functionality:

```
[ENS]Thu Oct 21 14:55:50 2007[EH: 12672K->12800K(12800K/65536K)][E/F/D: 20
0/0/0][cause:GC][CF: 0]
```

In this example, the allocated size of Explicit heap is 12,800 Kilobytes, and the number of Explicit memory block is 200. If this value is applied according to step (4) of *(1) Estimation procedure*, it is displayed as follows:

Estimation example of Explicit memory block size:

```
Size-of-one-Explicit-memory-block
= allocated-size-of-Explicit-heap-(12,800 Kilobytes)/number-of-Explicit-me
mory-size-(200)
=64 Kilobytes
```

The value multiplied with the number of sessions assumed in the businesses is considered as the total memory size of Explicit heap used in the HTTP session.

7.11.3 Impact of using the Explicit Memory Management functionality when estimating memory size

The use of a function from the Application Server functions has impact on the memory size of the Explicit heap area. The following table describes how the estimations are affected when you use a specific function:

Table 7–5: Impact on estimations according to the use of functions

No.	Function	Difference in the Explicit heap area according to the usage of function	Impact on estimation
1	Automatic release function of Explicit Memory Management (`-XX:+HitachiExplicitMemoryAutoReclaim` option)	When enabled JavaVM uses the "*size-of-the-Survivor-area-of-java-heap* `×2`" areas in the Explicit heap area. When disabled JavaVM does not use the Explicit heap area.	The estimated size of the final Explicit heap area when the function is enabled, is a value where "*size-of-the-Survivor-area-of-Java-heap*`×2`" is added to the estimated size calculated from the statistical information.
2	Automatic allocation functionality of the Explicit Memory Management functionality (`-XX:+HitachiAutoExplicitMemory` option)	When enabled The minimum size of Explicit memory block is 16 kilobytes. When disabled The minimum size of Explicit memory block is 64 kilobytes. Furthermore, how to allocate the Explicit heap area differs depending on the enabling and disabling of function. However, enabling and disabling does not have an impact on the estimations. When enabled The memory size specified in the `-XX:HitachiExplicitHeapMaxSize` option is allocated when starting the process. When disabled Only the required memory size is allocated when acquiring the Explicit memory block.	The information related to Explicit heap size that is output to statistical information differs depending on the enabling and disabling of function.

7.11.4 How to estimate using statistical information

You can use the statistical information to check the actual usage of Explicit heap with a J2EE server after J2EE server starts operating and for implementing the J2EE server tests. This subsection describes the procedure for checking the usage of Explicit heap using statistical information.

For details about the output contents of the statistical information and the settings to output the statistical information, and output destination of the statistics file, see *3.3 Statistics File Output Functionality* in the *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*.

(1) Concept of estimations using statistical information

When estimating using statistical information, the memory size of the Explicit heap area required in the system is as follows:

1. Memory size of the Explicit heap area used in HTTP session
2. Memory size of the Explicit heap area used in container excluding the area mentioned in point 1.
3. Memory size of the Explicit heap area used in applications and JavaVM
4. Memory size (*size-of-the-Survivor-area-of-Java-heap* ×2) of the Explicit heap area used to manage Explicit memory block by JavaVM

You can confirm the memory size of points 1. to 3. from statistical information. In point 4., you can use the memory size of *size-of-the-Survivor-area-of-Java-heap* ×2 when the automatic release function of Explicit Memory Management is enabled.

The following table describes the examples using the Explicit heap area mentioned in the points 1 to 3. The points 1 to 3 correspond to the item numbers 1 to 3 in the table:

Table 7–6: Concrete example of items using the Explicit heap area

No.	Explicit heap area	Concrete example of using the Explicit heap area
1	The Explicit heap area used in a HTTP session	HTTP session
2	The Explicit heap area used in a container	Objects used for managing HTTP sessions
3	The Explicit heap area used in applications and JavaVM	• Application • JavaVM

(2) Precautions when acquiring statistical information used in estimation

Acquire the statistical information used for estimations in the actual environment or an environment same as the actual environment.

When the following items differ from the actual environment, you cannot estimate the appropriate memory size using the statistical information:

- Properties set up in each definition file and values specified in options
- Number of Web applications registered on a server
- Size of the data processed by business applications
- Data processed at regular intervals

Furthermore, specify an option to set a maximum value for the size of the Explicit heap area to avoid completely used status of the Explicit heap area when acquiring the statistical information for estimation.

When you acquire statistical information in a status where maximum size of the Explicit heap area is insufficient, the Explicit heap area would be in a completely used state. You cannot estimate properly if the statistical information is acquired in a state whereby the Explicit area is completed used. You can confirm whether the Explicit heap area is in a completed used status by checking that the value of `EHeapSize.HighWaterMark` of statistical information is same as the value of maximum size of the Explicit heap area. The Explicit heap area is in completed used status when the value of `EHeapSize.HighWaterMark` of statistical information is same as the value of maximum size of the Explicit heap area.

(3) How to estimate

The following points describe how to estimate based on the statistical information:

(a) Memory size of the Explicit heap area used by HTTP session

You can acquire the memory size used by the HTTP sessions from the formulas for the memory size of the Explicit heap used by the HTTP sessions that are described in the *7.11.2 Memory size used by the object related to the HTTP session*. Here, you can confirm the "Memory size used on one session" included in formula from the statistical information.

Memory size used in one session corresponds to the "Maximum size of Explicit memory block" output to statistical information. In the "Maximum size of Explicit memory block", the size of maximum items used is output from the Explicit memory block released in the statistical information collection interval. Therefore, round-out in unit of 64 kilobytes and estimate the Explicit heap. Additionally, when using the automatic allocation functionality of the Explicit Memory Management functionality, add 16 kilobytes, and then estimate the Explicit heap.

When estimating, please see the following values. Also, the number of sessions required in the system correspond to "Number of Explicit memory blocks":

- Maximum size of Explicit memory blocks acquired in HTTP session (Value of `HTTPSessionEMemoryBlockMaxSize.HighWaterMark`)
- Number of Explicit memory blocks acquired in HTTP session (Value of `HTTPSessionEMemoryBlockCount.HighWaterMark`)

(b) Memory size of the Explicit heap area used in container

The memory size of the Explicit heap area used in the container corresponds to "Explicit heap size used in container" of statistical information. Use the maximum value (value of `ContainerEHeapSize.HighWaterMark`) from the acquired value for the statistical information used for estimation.

(c) Memory size of the Explicit heap area used in applications and JavaVM

The memory size of the Explicit heap area used in applications and JavaVM corresponds to "Explicit heap size used in applications" value of statistical information. Use the maximum value (`ApplicationEHeapSize.HighWaterMark`) from the acquired value for the statistical information used for estimation.

(4) Procedure to confirm the statistical information

This point describes the procedure to confirm the statistical information. This point also describes the confirmation procedures with estimation formula of statistical information in (3) as an example. For details about the output contents of the statistical information file, see *3.3 Statistics File Output Functionality* in the *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*.

Estimation formula

```
memory-size-of-the-required-Explicit-heap-area
=(value-where-HTTPSessionEMemoryBlockMaxSize.HighWaterMark-is-rounded-out-in
-64-kilobytes-unit
×HTTPSessionEMemoryBlockCount.HighWaterMark)
+ ContainerEHeapSize.HighWaterMark
+ ApplicationEHeapSize.HighWaterMark
+ size-of-the-Survivor-area-of-Java-heap
×2(only-when-explicit-memory-management-automatic-release-function-is-enable
d)
```

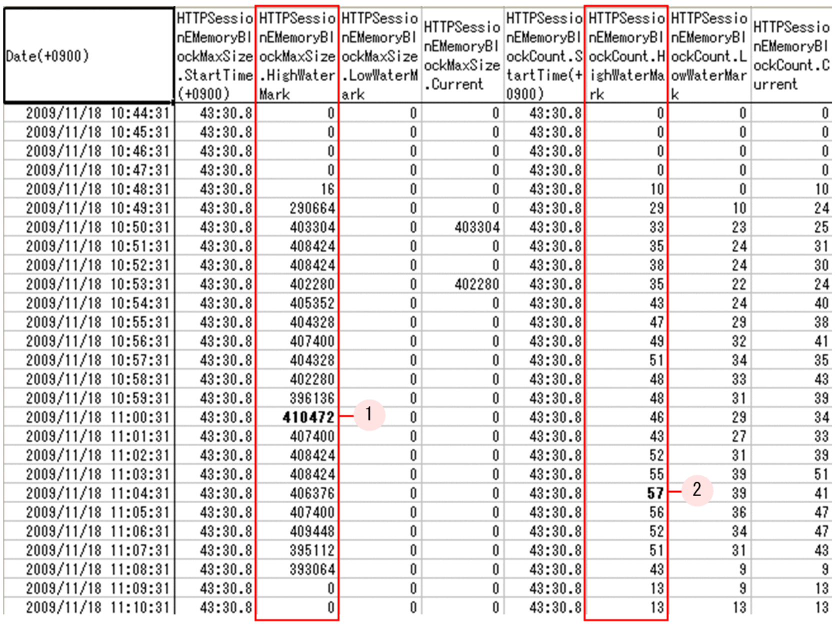
The following points describe how to confirm the respective values:

(a) Memory size of the Explicit heap area used by HTTP session

Confirm the memory size of the Explicit heap area used by
HTTP session from the `HTTPSessionEMemoryBlockMaxSize.HighWaterMark` and `HTTPSessionEMemoryBlockCount.HighWaterMark` values output to statistical information file of JavaVM.

The following figure shows an example to output statistical information of memory size of the Explicit heap area used by HTTP sessions:

Figure 7–13: Example to output statistical information of memory size of the Explicit heap area used by HTTP sessions



Date(+0900)	HTTPSessionEMemoryBlockMaxSize.StartTime(+0900)	HTTPSessionEMemoryBlockMaxSize.HighWaterMark	HTTPSessionEMemoryBlockMaxSize.LowWaterMark	HTTPSessionEMemoryBlockMaxSize.Current	HTTPSessionEMemoryBlockCount.StartTime(+0900)	HTTPSessionEMemoryBlockCount.HighWaterMark	HTTPSessionEMemoryBlockCount.LowWaterMark	HTTPSessionEMemoryBlockCount.Current
2009/11/18 10:44:31	43:30.8	0	0	0	43:30.8	0	0	0
2009/11/18 10:45:31	43:30.8	0	0	0	43:30.8	0	0	0
2009/11/18 10:46:31	43:30.8	0	0	0	43:30.8	0	0	0
2009/11/18 10:47:31	43:30.8	0	0	0	43:30.8	0	0	0
2009/11/18 10:48:31	43:30.8	16	0	0	43:30.8	10	0	10
2009/11/18 10:49:31	43:30.8	290664	0	0	43:30.8	29	10	24
2009/11/18 10:50:31	43:30.8	403304	0	403304	43:30.8	33	23	25
2009/11/18 10:51:31	43:30.8	408424	0	0	43:30.8	35	24	31
2009/11/18 10:52:31	43:30.8	408424	0	0	43:30.8	38	24	30
2009/11/18 10:53:31	43:30.8	402280	0	402280	43:30.8	35	22	24
2009/11/18 10:54:31	43:30.8	405352	0	0	43:30.8	43	24	40
2009/11/18 10:55:31	43:30.8	404328	0	0	43:30.8	47	29	38
2009/11/18 10:56:31	43:30.8	407400	0	0	43:30.8	49	32	41
2009/11/18 10:57:31	43:30.8	404328	0	0	43:30.8	51	34	35
2009/11/18 10:58:31	43:30.8	402280	0	0	43:30.8	48	33	43
2009/11/18 10:59:31	43:30.8	396136	0	0	43:30.8	48	31	39
2009/11/18 11:00:31	43:30.8	**410472**	0	0	43:30.8	46	29	34
2009/11/18 11:01:31	43:30.8	407400	0	0	43:30.8	43	27	33
2009/11/18 11:02:31	43:30.8	408424	0	0	43:30.8	52	31	39
2009/11/18 11:03:31	43:30.8	408424	0	0	43:30.8	55	39	51
2009/11/18 11:04:31	43:30.8	406376	0	0	43:30.8	**57**	39	41
2009/11/18 11:05:31	43:30.8	407400	0	0	43:30.8	56	36	47
2009/11/18 11:06:31	43:30.8	409448	0	0	43:30.8	52	34	47
2009/11/18 11:07:31	43:30.8	395112	0	0	43:30.8	51	31	43
2009/11/18 11:08:31	43:30.8	393064	0	0	43:30.8	43	9	9
2009/11/18 11:09:31	43:30.8	0	0	0	43:30.8	13	9	13
2009/11/18 11:10:31	43:30.8	0	0	0	43:30.8	13	13	13

The maximum value of `HTTPSessionEMemoryBlockMaxSize.HighWaterMark` is 410472 bytes (400.85 kilobytes) acquired at 11:00:31 in the 1. in above figure.

If you round-out this value in unit of 64 kilobytes, the value would be 448 kilobytes. The maximum value of `HTTPSessionEMemoryBlockCount.HighWaterMark` is 57 acquired at 11:04:31 in the 2. in above figure.

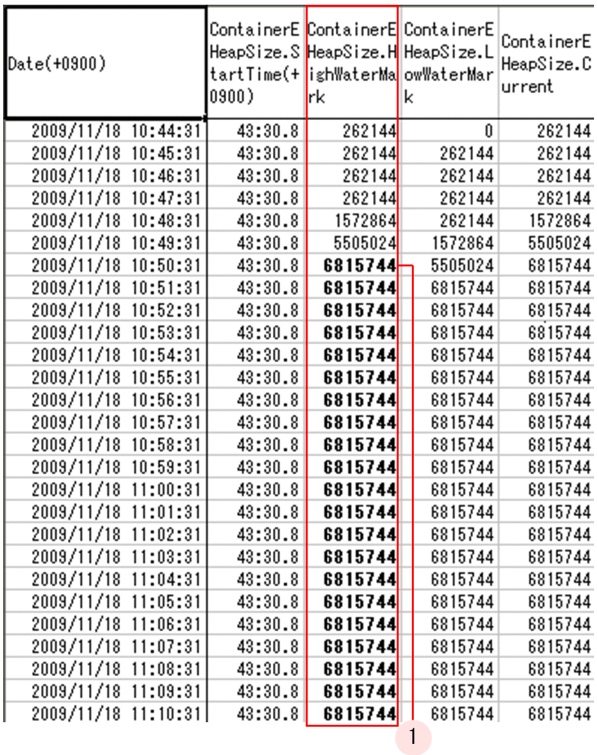
The value acquired by multiplying these two values would be the memory size of the Explicit heap area used by HTTP session.

(b) Memory size of the Explicit heap area used in container

Confirm the memory size of the Explicit heap area used by the container from `ContainerEHeapSize.HighWaterMark` value output to statistical information file of JavaVM.

The following figure shows an example to output statistical information of memory size of the Explicit heap area used in container:

Figure 7–14: Example to output statistical information of memory size of the Explicit heap area used in container



Date(+0900)	ContainerEHeapSize.StartTime(+0900)	ContainerEHeapSize.HighWaterMark	ContainerEHeapSize.LowWaterMark	ContainerEHeapSize.Current
2009/11/18 10:44:31	43:30.8	262144	0	262144
2009/11/18 10:45:31	43:30.8	262144	262144	262144
2009/11/18 10:46:31	43:30.8	262144	262144	262144
2009/11/18 10:47:31	43:30.8	262144	262144	262144
2009/11/18 10:48:31	43:30.8	1572864	262144	1572864
2009/11/18 10:49:31	43:30.8	5505024	1572864	5505024
2009/11/18 10:50:31	43:30.8	**6815744**	5505024	6815744
2009/11/18 10:51:31	43:30.8	**6815744**	6815744	6815744
2009/11/18 10:52:31	43:30.8	**6815744**	6815744	6815744
2009/11/18 10:53:31	43:30.8	**6815744**	6815744	6815744
2009/11/18 10:54:31	43:30.8	**6815744**	6815744	6815744
2009/11/18 10:55:31	43:30.8	**6815744**	6815744	6815744
2009/11/18 10:56:31	43:30.8	**6815744**	6815744	6815744
2009/11/18 10:57:31	43:30.8	**6815744**	6815744	6815744
2009/11/18 10:58:31	43:30.8	**6815744**	6815744	6815744
2009/11/18 10:59:31	43:30.8	**6815744**	6815744	6815744
2009/11/18 11:00:31	43:30.8	**6815744**	6815744	6815744
2009/11/18 11:01:31	43:30.8	**6815744**	6815744	6815744
2009/11/18 11:02:31	43:30.8	**6815744**	6815744	6815744
2009/11/18 11:03:31	43:30.8	**6815744**	6815744	6815744
2009/11/18 11:04:31	43:30.8	**6815744**	6815744	6815744
2009/11/18 11:05:31	43:30.8	**6815744**	6815744	6815744
2009/11/18 11:06:31	43:30.8	**6815744**	6815744	6815744
2009/11/18 11:07:31	43:30.8	**6815744**	6815744	6815744
2009/11/18 11:08:31	43:30.8	**6815744**	6815744	6815744
2009/11/18 11:09:31	43:30.8	**6815744**	6815744	6815744
2009/11/18 11:10:31	43:30.8	**6815744**	6815744	6815744

The maximum value of `ContainerEHeapSize.HighWaterMark` is 6815744 bytes (6656 kilobytes) acquired after 10:50:31 in the 1. in above figure.

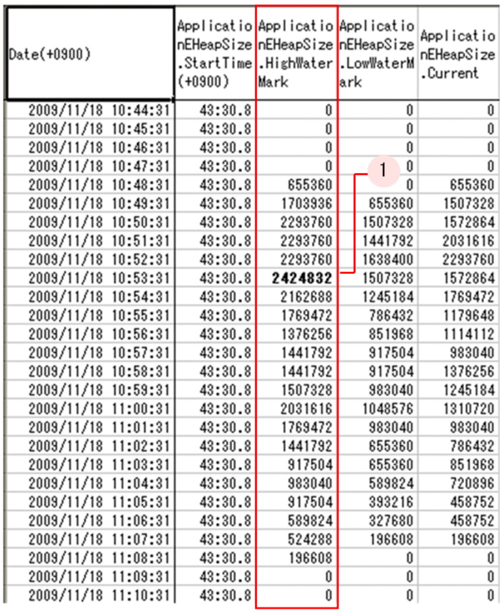
This is the memory size of the Explicit heap area used in container.

(c) Memory size of the Explicit heap area used by applications and JavaVM

Confirm the memory size of the Explicit heap area used by applications and JavaVM from `ApplicationEHeapSize.HighWaterMark` value output to statistical information file of JavaVM.

The following figure shows an example to output statistical information of memory size of the Explicit heap area used by applications and JavaVM:

Figure 7–15: Example to output statistical information of memory size of the Explicit heap area used by applications and JavaVM



Date(+0900)	ApplicationEHeapSize.StartTime(+0900)	ApplicationEHeapSize.HighWaterMark	ApplicationEHeapSize.LowWaterMark	ApplicationEHeapSize.Current
2009/11/18 10:44:31	43:30.8	0	0	0
2009/11/18 10:45:31	43:30.8	0	0	0
2009/11/18 10:46:31	43:30.8	0	0	0
2009/11/18 10:47:31	43:30.8	0	0	0
2009/11/18 10:48:31	43:30.8	655360	0	655360
2009/11/18 10:49:31	43:30.8	1703936	655360	1507328
2009/11/18 10:50:31	43:30.8	2293760	1507328	1572864
2009/11/18 10:51:31	43:30.8	2293760	1441792	2031616
2009/11/18 10:52:31	43:30.8	2293760	1638400	2293760
2009/11/18 10:53:31	43:30.8	**2424832**	1507328	1572864
2009/11/18 10:54:31	43:30.8	2162688	1245184	1769472
2009/11/18 10:55:31	43:30.8	1769472	786432	1179648
2009/11/18 10:56:31	43:30.8	1376256	851968	1114112
2009/11/18 10:57:31	43:30.8	1441792	917504	983040
2009/11/18 10:58:31	43:30.8	1441792	917504	1376256
2009/11/18 10:59:31	43:30.8	1507328	983040	1245184
2009/11/18 11:00:31	43:30.8	2031616	1048576	1310720
2009/11/18 11:01:31	43:30.8	1769472	983040	983040
2009/11/18 11:02:31	43:30.8	1441792	655360	786432
2009/11/18 11:03:31	43:30.8	917504	655360	851968
2009/11/18 11:04:31	43:30.8	983040	589824	720896
2009/11/18 11:05:31	43:30.8	917504	393216	458752
2009/11/18 11:06:31	43:30.8	589824	327680	458752
2009/11/18 11:07:31	43:30.8	524288	196608	196608
2009/11/18 11:08:31	43:30.8	196608	0	0
2009/11/18 11:09:31	43:30.8	0	0	0
2009/11/18 11:10:31	43:30.8	0	0	0

The maximum value of `ApplicationEHeapSize.HighWaterMark` is 2424832 bytes (2368 kilobytes) acquired at 10:53:31 in the 1. in above figure.

(d) Memory size of the required Explicit heap area

The following is the memory size of the required Explicit heap area acquired from the statistical information described in (a) to (c):

```
448 (kilobytes)× 57 + 6656 (kilobytes) + 2368 (kilobytes)
=34560 (kilobytes) ≒  34 megabytes
```

When the auto-release functionality of Explicit Memory Management is enabled, a value with "*Survivor-area-size-of-Java-heap* × 2" added becomes the final estimated size of the Explicit heap area.

7.12 Estimating the memory size when using the Explicit Memory Management functionality in the application

When there is an object that causes an increase in the memory size of the Tenured area in the application created by user, the corresponding object can be deployed in the Explicit heap. This section describes about how to estimate the memory size when using the Explicit Memory Management functionality in the application.

Tip

The description of this section corresponds to all the Java applications that run on JavaVM including the J2EE server. However, when you want to use only the Explicit heap used in the J2EE server, you are not required to read this section. However, you can see this section as and when required.

7.12.1 Determine whether to use the Explicit Memory Management functionality in the application

Full GC might occur frequently despite tuning the Java heap and following the procedure in *7.11.1 How to estimate the memory size of Explicit heap (Estimating memory size used in J2EE server)*. In this case, consider using the Explicit Memory Management functionality in the application.

First, identify which object is causing Full GC to occur. If Full GC can be controlled by deploying a specific object in the Explicit heap, then use the Explicit Memory Management functionality.

However, you must know the life-cycle of the object to be deployed in the Explicit heap. Determine the applicability when the timing to create the object and the timing when the object is no longer required is clear in the Java program.

To apply the Explicit Memory Management functionality, use the automatic allocation setup file and Explicit Memory Management functionality API. For details about using the Explicit Memory Management functionality to reduce the frequency of Full GC, see *7. Suppression of Full GC by Using the Explicit Memory Management Functionality* in the *uCosminexus Application Server Expansion Guide*.

7.12.2 Estimation concept

The memory size of the Explicit heap used in an application is estimated before you operate an application. Actually the memory size is estimated by checking an event log of the Explicit Memory Management functionality after the application is operated. For details about how to estimate, see *7.12.3 Memory size used in the application*.

When investigating the usage of the Explicit heap after the application is operated or is being developed, use the JavaVM log file and Java APIs. For details about the procedures for investigating after application development or operations, see *7.14 Errors that occur during the application of the Explicit Memory Management functionality and the solutions*.

Reference note

The information that you can use for estimating memory size is also output to the statistical information. For details about the tuning of memory size using statistical information, see *7.11.4 How to estimate using statistical information*.

This section also describes the correspondence between items output to statistical information and items output to thread dump.

7.12.3 Memory size used in the application

Estimate the memory size used in the application and memory size setup in the -XX:HitachiExplicitHeapMaxSize option before starting the operations.

The memory size is estimated by executing the test after running the application that has implemented the Explicit Memory Management functionality, and then confirming the output log. The estimated value is setup in the -XX:HitachiExplicitHeapMaxSize option of execution environment used in the actual operating environment.

This subsection describes the two types of estimation methods according to the environment used for executing the test.

Following are the prerequisites when the test is executed using any of the method:

Prerequisites for estimation

- Set up enough size for the maximum size of the Explicit heap and then execute the test.
- In -XX:HitachiExplicitMemoryLogLevel option, do not set none.

For details about the output item of the event log used for estimation, see *5.11 Event log of Explicit Memory Management functionality* in the *uCosminexus Application Server Maintenance and Migration Guide*.

(1) When you can execute the test in an environment equivalent to the actual operating environment

When the test can be executed in an environment equivalent to the actual operating environment, the maximum value of allocated memory size of Explicit heap output to the event log, is considered as the memory size of Explicit heap.

The procedure for checking is as follows:

1. Execute the application in usual manner in the test environment.
   If GC occurs while the application is running, information is output to the event log of the Explicit Memory Management functionality.
2. Check whether the value of allocated memory size (<EH_TOTAL>) of Explicit heap is maximum among all the records (row) of the event log output.

Consider this value as the memory size of the Explicit memory heap. However, when the auto-release functionality of Explicit Memory Management is enabled, use the value with "*Survivor-area-size* × 2" added, as the Explicit heap memory size.

(2) When the test is executed in the small scale environment as compared to the actual operating environment

When the test is executed in the small scale environment as compared to the actual operating environment, estimate the memory size required in the actual operating environment using the following formula:

```
Explicit-memory-size-in-the-actual-operating-environment
= (maximum-size-of-the-Explicit-heap)/ (number-of-Explicit-memory-blocks) ×
number-of-Explicit-memory-blocks-in-the-actual-operating-environment
+ Survivor-area-size×2#
```

#

Add "*Survivor-area-size* × 2" when the auto-release functionality of Explicit Memory Management is enabled.

The procedure for checking is as follows:

1. Execute the application in usual manner in the test environment.

   If GC occurs while the application is running, information is output to the event log of the Explicit Memory Management functionality.

2. Check whether the value of allocated memory size (<EH_TOTAL>) of Explicit heap is maximum among all the records (row) of the event log output. Also, check the number of valid Explicit memory blocks (Total of <AC_NUM> and <DA_NUM>) output to the same record.

3. Divide the value of <EH_TOTAL> checked in step (2) with (<AC_NUM>+<DA_NUM>).

   You can calculate a rough size for one Explicit memory block.

4. Multiply the maximum number of Explicit memory block calculated in the actual operating environment by the value calculated in step (3).

#

This method can be applied when the size of each Explicit memory block in development environment and actual environment is same and only the number is different.

Moreover, when Explicit heap is used for multiple purpose, you must check the Explicit heap for each purpose (the state where explicit memory block is not used other than the purpose of estimation).

Tip

The following table describes the correspondence of items output to thread dump and items output to statistical information:

Table 7–7: Correspondence of items output to thread dump and statistical information

Item output to thread dump	Item output to statistical information	Output contents
<EH_TOTAL>	EHeapSize	Allocated memory size of Explicit heap
<AC_NUM>+<DA_NUM>	EMemoryBlockCount	Number of valid Explicit memory blocks output to same records

7.13 Determining the usage of the Explicit heap using the automatic allocation functionality of the Explicit Memory Management functionality

This section describes how to determine the usage of the Explicit heap using the automatic allocation functionality of the Explicit Memory Management functionality.

You can easily use the Explicit Memory Management functionality, if you use the automatic allocation function for the Explicit heap area that uses the Explicit Memory Management functionality. Hitachi recommends that you use the automatic allocation function in the following cases:

7.13.1 When there are objects that cause increase in the Tenured area in the application

When there are objects that cause increase in the Tenured area in the application, Hitachi recommends that you use the automatic allocation setup file and allocate the objects to Explicit heap. A good example of Java program, with which you can determine the object allocation, is as follows:

```
01:package abcd.efg;
02:import java.util.HashMap;
03:// KVStorage-instance-continues-to-exist-for-a-long time
04:class KVStorage {
05: HashMap _map = new HashMap();
06:
07: public void store(MyKey k,MyData d) {
08: // ...before-processing...
09: _map.put(k,d);
10: // ...after processing...
11: }
12:
13: public MyData load(MyKey k) {
14: // ...before-processing...
15: MyData d = map.get(k);
16: // ...after processing...
17: return d;
18: }
19:}
```

In this setup example, the `HashMap` class, where `KVStorage` is stored to instance field, is the object with a long lifespan that causes increase in the memory size of the Tenured area. To change the generation destination of this object to Explicit heap, specify the automatic allocation setup file as described in the following example:

```
#Described for generation position (method name or class name), class name t
o be generated.
abcd.efg.KVStorage.<init>, java.util.HashMap
```

Alike this example, you can change the position for generating `HashMap` instance `(_map)` on 5th line in Java program example from Java heap to Explicit heap. You can also move the `MyKey` instance and `MyData` instance saved in `_map` by the `store` method sequentially to Explicit heap. These instances are automatically released by JavaVM when they are not required.

7.13.2 When the cause of increase in the used size of the Tenured area is not known

Even after implementing the Survivor area tuning, there is an increase in the used size of the Tenured area. As a result, when the occurrence interval of Full GC does not fulfill the system requirements, the objects that cause increase in the used size of the Tenured area are generated in Explicit heap. Set the automatic allocation setup file of Explicit Memory Management to generate objects in Explicit heap.

The following point describes how to investigate the objects that cause increase in the used size of the Tenured area and how to setup the automatic allocation setup file:

(1) Investigating increase in the used size of the Tenured area

Specify the `-garbage` option in the `jheapprof` command for the applications that are running, and execute the unused objects statistical function in the Tenured area.

In earlier versions than 08-70, to use the `-garbage` option concurrently with the automatic allocation function (`-XX:+HitachiAutoExplicitMemory`) of Explicit memory block, you must start the Application Server in a status where the `-XX:-HitachiExplicitMemoryPartialTenuredAreaCollection` option is specified beforehand.

The following is an output example of unused object statistical function in the Tenured area. This function outputs the list of class names of objects (basic objects that cause increase in the Tenured area) that cause increase in the used size of the Tenured area to Thread dump log file:

```
Garbage Profile Root Object Information
-----------------------------------
*, java.util.HashMap # 35234568
*, java.util.WeakHashMap # 4321000
```

In this output example, it is understood that the `java.util.HashMap` object is of 35,234,568 bytes and the `java.util.WeakHashMap` object is of 4,321,000 bytes that is the cause for increase in the used size of the Tenured area. For details about the unused object statistical function in the Tenured area, see *9.8 Unused objects statistical functionality in the Tenured area* in the *uCosminexus Application Server Maintenance and Migration Guide*.

(2) Description of the automatic allocation setup file

Enter some part of output examples list (last 2 lines) of the unused object statistical function in the Tenured area to automatic allocation setup file. An example for setting automatic allocation setup file is as follows:

```
*, java.util.HashMap # 35234568
*, java.util.WeakHashMap # 4321000
```

In such cases, all the `java.util.HashMap` objects and `java.util.WeakHashMap` objects in the program are generated in Explicit heap.

All the objects saved in above objects are moved sequentially to Explicit heap. The runtime overhead when generating objects in Explicit heap is higher as compared to generating objects in Java heap. Therefore, when generating objects in Explicit heap, you can reduce the runtime overhead by setting up the objects generating.

In the automatic allocation setup file, `*` indicates *all-the-classes-running-in-JavaVM*. In this setup example, the position for generating `java.util.HashMap` and `java.util.WeakHashMap` objects in all the classes is Explicit heap.

As a result, the position for generating objects that do not cause increase in the used size of the Tenured area is Explicit heap, and this might cause an increase in Explicit heap.

By specifying `*`, when the throughput of applications does not fulfill the requirements, determine the methods to sort positions for generating objects that cause increase in the used size of the Tenured area.

When the system administrator and application developer are two different people, the request for investigations must be sent to application developer. Even though it is difficult to perform detail investigation of the application, you can specify positions for generating objects in 4 stages such as "All classes", "Specific packages", "Specific classes", and "Specific methods" in the automatic allocation setup file. You can implement sorting on the range where investigations can be performed and can improve the throughput by specifying the automatic allocation setup file.

For example, when the position for generating object is under the `com.abc.defg` package, by changing the example for setting the automatic allocation setup file as follows you can sort from "All classes of all packages" to "All classes of `com.abc.defg` package and sub packages":

```
com.abc.defg.*, java.util.HashMap # 35234568
com.abc.defg.*, java.util.WeakHashMap # 4321000
```

For details about how to specify the automatic allocation configuration file, see *7.13.2 Using the Explicit Memory Management functionality by using the automatic placement configuration file* in the *uCosminexus Application Server Expansion Guide*.

(3) Investigating applications by the unused object statistical function in the Tenured area

To investigate the application based on contents of the automatic allocation setup file, use the unused object statistical function in the Tenured area.

Specify the `-garbage` option in the `jheapprof` command and execute the unused object statistical function in the Tenured area to output the list of basic objects that cause increase in the Tenured area and the statistical information of unused objects in the Tenured area to the extended thread dump. The following is an output example of extended thread dump:

```
Garbage Profile
---------------
________________Size__Instances__Class________________
            35234568     10648 java.util.HashMap
             5678900     10668 [Ljava.util.HashMap$Entry;
             4456788      7436 java.util.HashMap$Entry
             4321000       200 java.util.WeakHashMap
             1234568       190 [Ljava.util.WeakHashMap$Entry
1456788      9524 java.lang.String
1256788      6424 com.abc.defg.MyData;
:
```

There are objects that are saved to `java.util.HashMap` or `java.util.WeakHashMap` and not output to list of basic objects that cause increase in the Tenured area. These objects are output by the unused object statistical function in the Tenured area. The number of instances of each object are also output.

You can acquire this log multiple times and provide the log as an input file for the class-wise statistical information analysis function (`jheapprofanalyzer` command) to investigate the changes in each object size and time of the number of instances.

Investigate the applications based on the above information and sort the positions for generating objects. For details about the unused object statistical function in the Tenured area, see *9.8 Unused objects statistical functionality in the Tenured area* in the *uCosminexus Application Server Maintenance and Migration Guide*. For details about the class-wise statistical information analysis function, see *9.10 Class-wise statistical information analysis functionality* in the *uCosminexus Application Server Maintenance and Migration Guide*.

7.14 Errors that occur during the application of the Explicit Memory Management functionality and the solutions

The following table describes the events that might occur due to the memory size of the Explicit heap, how the Explicit Memory Management functionality is used, how the Java application is designed, and the execution environment settings.

Table 7–8: Events that occur during the application of the Explicit Memory Management functionality and the reference locations for the actions

No.	Event	Reference location for the action
1	The Explicit heap is full and an attempt to generate an object in the Explicit memory block fails.	*7.14.3*
2	An attempt to generate an Explicit memory block fails.	*7.14.4*
3	Even if the Explicit Memory Management functionality is used, a full garbage collection occurs frequently.	*7.14.5*
4	The automatic release processing of the Explicit memory block takes a long time.	*7.14.6*

You can check whether these events are occurring from the event log of the Explicit Memory Management functionality. For details on the verification methods, see *7.14.1 Investigating the usage (snapshot) of Explicit heap at a certain point*, and *7.14.2 Investigating the transition of usage status*. For details on the event log contents of the Explicit Memory Management functionality, see *5.11 Event log of Explicit Memory Management functionality* in the *uCosminexus Application Server Maintenance and Migration Guide*. Note that for ready visibility, \ has been entered at the linefeed locations in the examples of output described in this section. There is no linefeed in the actually output content.

7.14.1 Investigating the usage (snapshot) of Explicit heap at a certain point

The method of investigating the usage (snapshot) of Explicit heap at a certain point includes the method for checking the thread dump and the method for acquiring information using Java API.

- **How to check thread dump**

  You can execute the `cjdumpsv` command to output a thread dump at any given time. In the thread dump, usage of Explicit heap and each memory block is output.

  An output example is as follows:

```
Explicit Heap Status
--------------------
 max 65536K, total 21376K, used 20480K, garbage 1234K (31.2% used/max, 95.
8% used/total, 6.0% garbage/used), 1 spaces exist
 Explicit Memories(0x12345678)
 "EJBMgrData" eid=1(0x02f25610)/R, total 21376K, used 20480K, garbage 1234
K (95.8% used/total, 6.0% garbage/used, 0 blocks) Enable
```

  Parts that are highlighted and **bolded** indicate the usage status of the Explicit heap and the individual Explicit memory block.

  In this example, the maximum size of Explicit heap is 65,536 Kilobytes and the allocated Explicit heap size is 21,376 Kilobytes. Moreover, the memory allocated size of Explicit memory block called *EJBMgrData* is 21,376 Kilobytes and the used size is 20,480 Kilobytes.

- **How to acquire the information with Java API**
  You can acquire the usage of the Explicit heap and the Explicit memory block using the Java API of JavaVM. By implementing the application using the following API, you can acquire the information at any given time of processing.

  Method used to acquire the use status of Explicit heap:

  ```
  JP.co.Hitachi.soft.jvm.MemoryArea.ExplicitMemory.getMemoryUsage()
  ```

  Method used to acquire the use status of Explicit memory block:

  ```
  JP.co.Hitachi.soft.jvm.MemoryArea.ExplicitMemory.freeMemory()
  JP.co.Hitachi.soft.jvm.MemoryArea.ExplicitMemory.totalMemory()
  JP.co.Hitachi.soft.jvm.MemoryArea.ExplicitMemory.usedMemory()
  ```

7.14.2 Investigating the transition of usage status

The method for investigating the transition of use status of Explicit heap includes the method for investigating the event log of the Explicit Memory Management functionality and method for acquiring the information using JavaAPI. For details about how to use the Java API, see *7.14.1 Investigating the usage (snapshot) of Explicit heap at a certain point*.

This point describes how to investigate using the event log of the Explicit Memory Management functionality.

(1) Transition of usage status of Explicit heap

Set `normal` in the `-XX:HitachiExplicitMemoryLogLevel` option of JavaVM. As a result, the use status of Explicit heap is output at the following timings:

- When GC occurs (regular)
- During the explicit release processing of the Explicit memory block
- During the automatic release processing of the Explicit memory block

Moreover, when `verbose` is specified in the `-XX:HitachiExplicitMemoryLogLevel` option of JavaVM, the use status of the Explicit heap will be output at the following timings in addition to the output timing in case of `normal`.

- When object of Explicit memory block is created by the methods such as `ExplicitMemory.newInstance` method

An output example is as follows:

```
[ENS]<Thu Oct 21 14:55:50 2007>[EH: 12672K->12800K(12800K/65536K)][E/F/D: 20
0/0/0][cause:GC][CF: 0]
[ENS]<Thu Oct 21 14:55:50 2007>[EH: 12800K->12800K(12800K/65536K), 0.112462
6 secs][E/F/D: 200/0/0]\
[DefNew::Eden: 0K->0K(243600K)][DefNew::Survivor: 0K->0K(17400K)][Tenured: 1
03400K->103400K(556800K)]\
[target:584K/0K/584K][cause:Migrate]
```

The highlighted and **bolded** parts beginning with `EH:` indicate the usage status of the Explicit heap. The line that indicates the usage status of the Explicit heap always contains information equivalent to the highlighted and **bolded** information. You can see how usage changes over time by such means as plotting this value on a graph.

In this example, as shown by the part starting with `cause:`, the first line that starts with `[ENS]` is the log entry output when copy GC (GC) occurred. The second line that starts with `[ENS]` is the log entry output during automatic release processing, the processing by which the Explicit memory block is automatically released. The log in the first line indicates that the used size of the Explicit heap before the copy GC occurs is 12,672 KB, and the used size after the copy garbage collection occurs is 12,800 KB. The log in the second line indicates that the used size of the Explicit heap is 12,800 KB and that the automatic release processing of the Explicit memory block took 0.1124626 seconds.

Note that the log indicating the use status starts with `ENS` or `EVS`. When the event log is filtered in this character string, it becomes easier to check the use status.

(2) Transition of usage status for each Explicit memory block

Specify `verbose` in the `-XX:HitachiExplicitMemoryLogLevel` option of JavaVM. As a result, the use status of Explicit memory block with the changed size will be output at the following timings:

- When migrating the object to the Explicit memory block
- When creating the object of the Explicit memory block

An output example is as follows:

```
[ENS]Thu Oct 21 14:55:50 2007[EH: 11422K->12800K(12800K/65536K)][E/F/D: 200/
0/0][cause:GC][CF: 0]
[EVS]["REM2" eid=2/R: 0K->88K(128K)]["REM3" eid=3/R: 30K->230K(256K)]["REM6
" eid=6/R: 30K->200K(256K)]\
["Session1" eid=8/R: 30K->250K(256K)]["Session2" eid=10/R: 30K->250K(256K)]
[EVS]["Session3" eid=12/R: 30K->510K(512K)]
```

The part that is highlighted and **bolded** indicates the usage status of the Explicit memory block named `"REM2"`.

Moreover, when `verbose` is specified in the `-XX:HitachiExplicitMemoryLogLevel` option, if the Explicit memory block is released, the information of the released Explicit memory block is output. An output example is as follows:

```
[ENS]Tue Jul 24 01:23:51 2007[EH: 12800K->11776K(11776K/65536K), 0.1129602 s
ecs][E/F/D: 523/0/0]\
[DefNew::Eden: 0K->0K(243600K)][DefNew::Survivor: 12K->0K(17400K)][Tenured:
103400K->103400K(556800K)][cause:Reclaim]
[EVS]["REM2" eid=2/R: 320K]["BEM3" eid=5/B: 320K]["BEM1" eid=7/B: 384K]
```

The part that is highlighted and **bolded** indicates information about the released Explicit memory block named `"REM2"`. "320K" is the released memory size (Size of the allocated Explicit memory block).

You can check the transition of the use status for each Explicit memory block by coding and plotting these values in the graph. Note that the size of individual Explicit memory block increases uniformly until released.

7.14.3 Checking and measures when there is an overflow from the Explicit heap

This point describes the measures to be taken and how to check when there is overflow of Explicit heap.

An *overflow of Explicit heap* indicates the following state:

- State where Explicit heap was used up to the maximum size
- State where OS failed to allocate memory when the Explicit memory block is extended

When there is an overflow of the Explicit heap, the sub-state of Explicit memory block, whose area is tried to be extended, changes from `Enable` to `Disable`. You cannot deploy the object in Explicit memory block which is in the `Disable` state.

You can check whether there is an overflow of Explicit heap by checking the event log of the Explicit Memory Management functionality or by checking the contents of thread dump. Moreover, you can also check through the information acquired by JavaAPI.

When there is an overflow of Explicit heap, take the following measures:

Measures to be taken when Explicit heap overflows

- Increase the maximum size of Explicit heap.
  Change the specifications of the `-XX:HitachiExplicitHeapMaxSize` option.
- If there is an overflow of Explicit heap when the maximum size of Explicit heap is not achieved, increase the memory size that can be allocated from OS
  Increase the memory size that can be used by the Application Server.
- Eliminate the cause responsible for consuming large amount of Explicit heap.

These points describe about how to check an overflow from Explicit heap.

(1) Investigating the event log of the Explicit Memory Management functionality

To investigate with the event log of the Explicit Memory Management functionality, you must specify `normal` in the `-XX:HitachiExplicitMemoryLogLevel` option of JavaVM in advance. As a result, whenever the GC occurs, the use status of Explicit block will be output to the event log of the Explicit Memory Management functionality.

An output example is as follows:

```
[ENS]Thu Oct 21 14:55:50 2007[EH: 12672K->12800K(12800K/65536K)][E/F/D: 200/
0/0][cause:GC][CF: 0]
```

The part that is highlighted and **bolded** indicates the number of Explicit memory blocks. `E` and `D` indicate `Enable` and `Disable` which are the sub-states of Explicit memory block. When the Explicit memory block is in the `Disable` state, there is an overflow of Explicit heap. In this example, it is concluded that there are 200 Explicit memory blocks in `Enable` state and nil in `Disable` state. Furthermore, when the Explicit memory block is `Disable`, check for relation with the Explicit heap maximum size. When there is extra Explicit heap maximum size, it can be concluded that the OS fails to allocate the memory.

Moreover, when `verbose` is specified in the `-XX:HitachiExplicitMemoryLogLevel` option of JavaVM, the factors responsible for changing the sub-state of Explicit memory block to `Disable` state, is also output.

An output example is as follows:

```
[EVO]Tue Jul 24 01:23:51 2007[alloc failed(Disable)][EH: 32760K(0K)/32768K/6
5536K][E/F/D: 321/0/1][cause:GC]\
["BasicExplicitMemory-3" eid=3/B: 128K(0K)/128K][Thread: 0x00035a60]
[EVO][Thread: 0x00035a60] at ExplicitMemory.newInstance0(Native Method)
```

```
[EVO][Thread: 0x00035a60] at BasicExplicitMemory.newInstance(Unknown Source)
[EVO][Thread: 0x00035a60] at AllocTest.test(AllocTest.java:64)
[EVO][Thread: 0x00035a60] at java.lang.Thread.run(Thread.java:2312)
```

Above is the example showing the overflow of Explicit heap.

Of the parts that are highlighted and **bolded**, `[alloc failed(Disable)]` indicates the reason the Explicit memory block entered the *Disable* sub-status. `"BasicExplicitMemory-3" eid=3/B: 128K(0K)/128K` indicates the information of Explicit memory block that is in `Disable` state. Moreover, the rows starting with `[EVO][Thread: 0x00035a60]` indicates the stack trace when the event has occurred. However, when there is an overflow of Explicit heap due to the transition of object by GC, the stack trace is not output.

(2) Investigating from the log file output by the thread dump

You can output the sub-state of each Explicit memory block by the output of the thread dump using the `cjdumpsv` command.

The output example is as follows:

```
Explicit Heap Status
--------------------
 max 65536K, total 21888K, used 20992K, garbage 1288K (32.0% used/max, 95.9
% used/total, 6.1% garbage/used), 2 spaces exist

 Explicit Memories(0x12345678)

 "EJBMgrData" eid=1(0x02f25610)/R, total 21376K, used 20480K, garbage 1234K
(95.8% used/total, 6.0% garbage/used, 0 blocks) Enable

 "ExplicitMemory-4" eid=4(0x02f45800)/B, total 512K, used 512K, garbage 54K
(100.0% used/total, 10.5% garbage/used, 0 blocks) Disable
```

Parts that are highlighted and **bolded** indicate the sub-status of each Explicit memory block.

(3) Investigation from the API of Java

You can investigate the sub-state of the Explicit memory block using the following methods:

- `JP.co.Hitachi.soft.jvm.MemoryArea.ExplicitMemory.isActive()`
- `JP.co.Hitachi.soft.jvm.MemoryArea.ExplicitMemory.isReclaimed()`

If the return value of both methods is `false`, the sub-state of the Explicit memory block can be determined as `Disable`.

7.14.4 Checking and measures when the initialization of the Explicit memory block fails

This point describes about the measures to be taken and how to check when the initialization of Explicit memory block fails.

When number of Explicit memory blocks reaches the maximum limit, the Explicit memory block cannot be initialized beyond that limit.

In such cases, reduce the number of Explicit memory blocks.

Here, how to check whether the initialization of Explicit memory block has failed, is described.

(1) Investigating from the event log of the Explicit Memory Management functionality

To investigate with the event log of explicit management functionality, you must specify `normal` in the `-XX:HitachiExplicitMemoryLogLevel` option of JavaVM in advance. As a result, whenever the GC occurs, the frequency at which the initialization of Explicit memory block fails, will be output to the Explicit Memory Management functionality event log.

An output example is as follows:

```
[ENS]Thu Oct 21 14:55:50 2007[EH: 12672K->12800K(12800K/65536K)][E/F/D: 200/
0/0][cause:GC][CF: 0]
```

The part that is highlighted and **bolded** indicates the number of times initialization of the Explicit memory block failed since the last time this information was output. In this example, frequency is "0". When there is no failure in the initialization, it is concluded that there is no problem.

Moreover, when `verbose` is specified in the `-XX:HitachiExplicitMemoryLogLevel` option of JavaVM, the information about event of failure in the initialization of Explicit memory block will be output.

An output example is as follows:

```
[EVO]Tue Jul 24 01:23:51 2007[Creation failed][EH: 32760K(0K)/32768K/65536K]
[E/F/D: 65535/0/0][Thread: 0x00035a60]
[EVO][Thread: 0x00035a60] at ExplicitMemory.registerExplicitMemory(Native Me
thod)
[EVO][Thread: 0x00035a60] at BasicExplicitMemory.<init>(Unknown Source)
[EVO][Thread: 0x00035a60] at AllocTest.test(AllocTest.java:64)
[EVO][Thread: 0x00035a60] at java.lang.Thread.run(Thread.java:2312)
```

The part that is highlighted and **bolded** indicates that initialization of the Explicit memory block has failed. Moreover, the rows starting with `[EVO][Thread: 0x00035a60]` indicates the stack trace when event occurs.

Also, when `debug` is specified in the `-XX:HitachiExplicitMemoryLogLevel` option of JavaVM, the detailed information about the initialization events of the Explicit memory block other than the event of failure in the initialization will be output. The initialization fails when the number of Explicit memory blocks exceed the constant number. Therefore, the information of initialization event prior to failure in initialization is useful for investigations.

An output example is as follows:

```
[EVO]Tue Jul 24 01:23:51 2007[Created]["BasicExplicitMemory-2" eid=2(0x12345
68)/B][Thread: 0x00035a60]
[EDO][Thread: 0x00035a60] at ExplicitMemory.registerExplicitMemory(Native Me
thod)
[EDO][Thread: 0x00035a60] at BasicExplicitMemory.<init>(Unknown Source)
[EDO][Thread: 0x00035a60] at AllocTest.test(AllocTest.java:64)
[EVO][Thread: 0x00035a60] at java.lang.Thread.run(Thread.java:2312)
```

The part that is highlighted and **bolded** indicates that the information relates to initialization of the Explicit memory block. Moreover, the rows starting with `[EDO][Thread: 0x00035a60]` indicates the stack trace when the event has occurred.

(2) Investigating from the log file output by the thread dump

Though you cannot check the direct cause responsible for failure in initialization of the Explicit memory block from the information output by thread dump, you can find out the number of Explicit memory blocks.

An output example is as follows:

```
Explicit Heap Status
--------------------
 max 65536K, total 21888K, used 20992K, garbage 1288K (32.0% used/max, 95.9
% used/total, 6.1% garbage/used), 2 spaces exist

 Explicit Memories(0x12345678)

 "EJBMgrData" eid=1(0x02f25610)/R, total 21376K, used 20480K, garbage 1234K
(95.8% used/total, 6.0% garbage/used, 0 blocks) Enable

 "ExplicitMemory-4" eid=4(0x02f45800)/B, total 512K, used 512K, garbage 54K
(100.0% used/total, 10.5% garbage/used, 0 blocks) Disable
```

The part that is highlighted and **bolded** indicates the number of Explicit memory blocks.

(3) Investigating from the API of Java

You can investigate the number of Explicit memory blocks using the following methods:

- `JP.co.Hitachi.soft.jvm.ExplicitMemory.countExplicitMemories()`

However, you cannot check the direct cause responsible for failure in initialization of the Explicit memory block in this API.

7.14.5 Checks and measures when an object is transited to the Java heap during the explicit release processing of the Explicit memory block

When there is a reference available for the objects within the Explicit heap that is to be released during the explicit release processing of the Explicit memory block, the referenced object and the objects referenced either directly or indirectly from that object, are transited to the Java heap. The object is transited to the Tenured area on priority. Therefore, when the object is transited frequently, the used size of the Tenured area increases resulting in Full GC.

You can investigate whether the object is transited to the Java heap through the extended `verbosegc` information of the JavaVM log file or the explicit management heap event log.

(1) Checking by using the extended `verbosegc` information

When the Explicit Memory Management functionality is not used, the used size of the Tenured area increases only when the copy GC occurs. Therefore, the used size of the Tenured area after the Nth copy GC completion matches the used size of the Tenured area before the *N*+1th copy GC starts.

As a result, when the objects are transited from Explicit heap to Java heap, the used size of the Tenured area increases during the Explicit heap release. From this difference, it is concluded that the objects start transiting during the explicit release processing of the Explicit memory block.

You can calculate the size of the object transited to the Java heap during the explicit release processing of the Explicit memory block after completion of the *N*th copy GC by the following formula:

```
Size-of-object-transited-from-Explicit-heap-to-Java-heap
=tenured-area-used-size-before-N + 1th-Copy-GC
 -Tenured-area-used-size-after-Nth-Copy-GC
```

(2) Checking by using the event log of the explicit management heap

When `none` is not specified in the `-XX:HitachiExplicitMemoryLogLevel` option of JavaVM, a log related to the explicit release processing of the Explicit memory block is output. In this log, you can directly check the increase in the size of the used memory of the Tenured area during the explicit release processing of the Explicit memory block.

An output example is as follows:

```
[ENS]Tue Jul 24 01:23:51 2007[EH: 12800K->11776K(11776K/65536K), 0.1129602 s
ecs][E/F/D: 523/0/0]\
[DefNew::Eden: 0K->0K(243600K)][DefNew::Survivor: 0K->0K(17400K)][Tenured: 1
03400K->103464K(556800K)][cause:Reclaim]
```

Of the parts that are highlighted and **bolded**, `[cause:Reclaim]` indicates that the information was output when the Explicit memory block was explicitly released. Furthermore, `[DefNew::Eden: 0K->0K(243600K)] [DefNew::Survivor: 0K->0K(17400K)][Tenured: 103400K->103464K(556800K)]` indicates the change in the Java heap during the explicit release processing of the Explicit memory block. In this example, the memory size of the Tenured area is increased from 103,400 Kilobytes to 103,464 Kilobytes. That is, the memory size is increased by 64 Kilobytes. From this, it is concluded that an object of 64 Kilobytes is transited from the Java heap during the explicit release processing of the Explicit memory block.

Moreover, when `verbose` is specified in the `-XX:HitachiExplicitMemoryLogLevel` option of JavaVM, information related to released Explicit memory block is also output. As a result, you can check by which Explicit memory block release the used size of the Tenured area has increased.

An output example is as follows:

```
[ENS]Tue Jul 24 01:23:51 2007[EH: 12800K->11776K(11776K/65536K), 0.1129602 s
ecs][E/F/D: 523/0/0]\
[DefNew::Eden: 0K->0K(243600K)][DefNew::Survivor: 0K->0K(17400K)][Tenured: 1
03400K->103464K(556800K)][cause:Reclaim]
[EVS]["REM2" eid=2/R: 320K]["BEM3" eid=5/B: 320K]["BEM1" eid=7/B: 384K]
```

Parts that are highlighted and **bolded** indicate the released Explicit memory block. From the output contents, it is understood that the object of 64 Kilobytes transited to Java heap, is transited from the Explicit memory block of either (REM2) (BEM3) (BEM1).

Also, when `debug` is specified in the `-XX:HitachiExplicitMemoryLogLevel` option of JavaVM, you can check the object referenced as the object within the Explicit heap to be released during the explicit release processing.

An output example is as follows:

```
[EDO][eid=3: Reference to ClassZ(0x1234680), total 64K]
[EDO] ClassU(0x1233468)(Tenured)
```

Following can be understood from `[eid=3: Reference to ClassZ(0x1234680), total 64K]`:

- An object transited to Java heap, is the instance of "ClassZ"
- The total size of the object transited to Java heap is 64 Kilobytes by referencing to "ClassZ" instance.

Moreover, from (ClassU(0x1233468)(Tenured)), it can be concluded that the object that refers to the instance of "Class Z" is the instance of "Class U".

Based on this information, modify the Java program in such a way so that there is no reference to the object in the Explicit memory block during the explicit release processing of that Explicit memory block.

7.14.6 Checks and measures when the automatic release processing of the Explicit memory block takes a long time

This subsection describes the checks and measures when the automatic release processing of the Explicit memory block takes a long time.

The automatic release processing of the Explicit memory block occurs concurrently with the GC, and the processing of the J2EE server stops during the automatic release processing. Therefore, if the automatic release processing takes a long time, problems might occur in the system.

If an Explicit memory block with a large size (called *large block* hereafter) is generated, the automatic release processing takes a long time. A large block might be generated when objects that are not collected by Full GC, such as objects used the entire time an application runs, are allocated to the Explicit heap. Therefore, you must prevent the generation of a large block by allocating the objects that must not be allocated to the Explicit heap, to the Java heap. For details on the event in which a large block is generated and the automatic release processing takes time, see *7.10.2 Mechanism of reducing time required for automatic release processing*, and *Appendix B.1 Effect on the automatic release processing of the Explicit memory block* in the *uCosminexus Application Server Expansion Guide*.

From the content of the thread dump, you can check whether a large block is being generated.

If a large block is generated, follow the below procedure to specify settings to allocate the objects responsible for the large block to the Java heap.

Action

1. Applying the functionality for controlling the transfer of objects to the Explicit memory block
2. Applying the functionality for specifying the classes to be excluded from the Explicit Memory Management functionality
3. Identifying the objects responsible for a large block and preventing the transfer of those objects to the Explicit heap
4. Re-tuning the Java heap area and Explicit heap area

This subsection describes how to check whether a large block is being generated and the action to be taken.

(1) Checking whether a large block is being generated

Execute the `eheapprof` command to check the Explicit heap information output to the thread dump.

An example of output is as follows:

```
 "NULL" eid=1(0x1000000000123456)/B, total 112K, used 55K, garbage 0K (49.2
% used/total, 0.0% garbage/used, 0 blocks) Enable
 "NULL" eid=2(0x1000000000223456)/A, total 153744K, used 144766K, garbage 0
K (94.2% used/total, 0.0% garbage/used, 0 blocks) Enable
 "ReferenceExplicitMemory-2" eid=3(0x1000000000323456)/R, total 112K, used 5
5K, garbage 0K (49.3% used/total, 0.0% garbage/used, 0 blocks) Enable
```

Note:

> `NULL` in the name of the Explicit memory block indicates that the automatic release processing was implemented once, for this Explicit memory block.

The highlighted and **bolded** parts labeled `total` indicate the memory size allocated to the Explicit heap.

In this example, the memory size described in `total` for each Explicit memory block indicates that the Explicit memory block `eid=2` is 153,744 KB, and is extremely large compared to the 112-KB Explicit memory blocks `eid=1` and `eid=3`. This indicates that a large block has been generated and that this block is the Explicit memory block `eid=2`.

(2) Applying the functionality for controlling the transfer of objects to the Explicit memory block

Check the transition in the usage status of the Explicit heap using the event log of the Explicit Memory Management functionality. For details on how to check the transition in the usage status of the Explicit heap, see *7.14.2 Investigating the transition of usage status*.

An example of output is as follows:

```
[ENS]<Thu Oct 21 14:55:50 2007>[EH: 12672K->172032K(172032K/196608K)][E/F/D
: 200/0/0][cause:Full GC][CF: 0]
[ENS]<Thu Oct 21 14:55:50 2007>[EH: 172032K->172032K(172032K/196608K), 0.112
4626 secs][E/F/D: 200/0/0]\
[DefNew::Eden: 0K->0K(243600K)][DefNew::Survivor: 0K->0K(17400K)][Tenured: 1
03400K->103400K(556800K)]\
[target:584K/0K/584K][cause:Migrate]
```

The highlighted and **bolded** parts beginning with `EH:` indicate the usage status of the Explicit heap.

In this example, as shown by the part starting with `cause:`, the first line is log data output when Full GC occurred, and the line immediately after is log data output during automatic release processing, the processing by which the Explicit memory block is automatically released. This indicates that the automatic release processing occurred immediately after the Full GC. The part beginning with `EH:` in the log on the first line indicates that the used size of the Explicit heap before the Full GC occurred is 12,672 KB, the used size after the Full GC occurs is 172,032 KB, and that the used size of the Explicit heap has increased considerably. Note that if such an event (great increase in the used size of the Explicit heap) does not occur, proceed to step (3).

As shown by this example, if the used size of the Explicit heap has increased considerably after Full GC, apply the functionality for controlling the transfer of objects to the Explicit memory block by specifying `1` in the `-XX:ExplicitMemoryFullGCPolicy` option. This functionality prevents objects from being transferred to the Explicit memory block based on reference relationships when Full GC occurs. For details on the functionality for controlling the transfer of objects to the Explicit memory block, see *7.10 Reducing time required for automatic release processing of Explicit memory blocks* in the *uCosminexus Application Server Expansion Guide*.

Proceed to step (5) if you can prevent the generation of a large block by applying this functionality, or proceed to step (3) if a large block is still being generated.

(3) Applying the functionality for specifying the classes to be excluded from the Explicit Memory Management functionality

If the following events occur, specify the `-XX:+ExplicitMemoryUseExcludeClass` option to apply the functionality for specifying the classes to be excluded from the Explicit Memory Management functionality.

- A considerable increase in the Explicit heap was not observed in the usage status after Full GC implemented immediately before automatic release processing
- A large block is being generated even after applying the functionality for controlling the transfer of objects to the Explicit memory block (step (2) is implemented)

If you apply the functionality for specifying the classes to be excluded from the Explicit Memory Management functionality, the objects of a specific class are no longer transferred to the Explicit heap. The objects of a specific class indicate the objects of the classes coded in the exclusion configuration file for the Explicit Memory Management functionality provided with the system (`sysexmemexcludeclass.cfg`). For details on the functionality for specifying the classes to be excluded from the Explicit Memory Management functionality, see *7.10 Reducing time required for automatic release processing of Explicit memory blocks* in the *uCosminexus Application Server Expansion Guide*.

Proceed to step (5) if you can prevent the generation of a large block by applying this functionality, or proceed to step (4) if a large block is still being generated.

(4) Identifying the objects responsible for a large block and preventing the transfer of the objects to the Explicit heap

If a large block is being generated even after applying the functionality for specifying the classes to be excluded from the Explicit Memory Management functionality in step (3), the large block might be generated by the transfer of the following objects to the Explicit heap. From among these objects, the objects that continue to survive are not called back with the automatic release processing of the Explicit memory block.

- Objects of the classes created by the user in the Java application
- Objects created automatically by the framework being used

Therefore, the Explicit Memory Management functionality is not effective even if the objects that are not called back within a fixed period are allocated to the Explicit heap, so allocating the objects to the Java heap (Tenured area) is more appropriate. From among the objects, identify the objects responsible for a large block, and make sure these objects do not move to the Explicit heap.

To identify the objects responsible for a large block, check the object release ratio. The object release ratio is the percentage of objects released with the automatic release processing of the Explicit memory block. Execute the `eheapprof` command by specifying the `-freeratio` option to output the object release ratio in the Explicit heap information output in the thread dump.

An example of output is as follows:

```
  "NULL" eid=1(0x1000000000123456)/B, total 112K, used 55K, garbage 0K (49.2
% used/total, 0.0% garbage/used, 0 blocks) Enable
    deployed objects
    ________________Size__Instances__FreeRatio__Class________________
```

```
                   49256         10         0 [B
                    3680          4        20 package1.session.StandardManag
er
                   52936         14 total


  "NULL" eid=2(0x1000000000223456)/A, total 153744K, used 144766K, garbage 0
K (94.2% used/total, 0.0% garbage/used, 0 blocks) Enable
    deployed objects
    ________________Size__Instances__FreeRatio__Class________________
                77862918     433523        10 [C
                52622946     441714        10 java.lang.String
                12838192      39462        35 [B
                    3680          4        20 package1.session.StandardManag
er
                     104          4         0 framework.ut.impl.performList
               143327840     914707 total

  "ReferenceExplicitMemory-2" eid=3(0x1000000000323456)/R, total 112K, used
55K, garbage 0K (49.3% used/total, 0.0% garbage/used, 0 blocks) Enable
    deployed objects
    ________________Size__Instances__FreeRatio__Class________________
                   49256          4         - [B
                    3416         10         - package3.ajp.RequestHandler
                      64          2         - java.lang.StringBuffer
                      64          1         - java.net.SocketInputStream
                      48          1         - [I
                      24          1         - [C
                   52872         19 total
```

#1

NULL in the name of the Explicit memory block indicates that the automatic release processing was implemented once for this Explicit memory block.

#2

[B in the class name indicates the array type of the Byte class, [C indicates the array type of the Char class, and [I indicates the array type of the Integer class.

The highlighted and **bolded** parts labeled total indicate the memory size allocated to the Explicit heap. The parts labeled FreeRatio indicate information about the ratio of released objects.

In this example, there are three Explicit memory blocks, eid=1, eid=2, and eid=3. The object release ratio indicated in FreeRatio of each Explicit memory block shows that the object release ratio of the Explicit memory block eid=3 is "-", and that the automatic release processing has not been executed.

First, see the memory size indicated in total for each Explicit memory block. The Explicit memory block eid=2 is 153,744 KB, and is extremely large compared to the 112-KB Explicit memory blocks eid=1 and eid=3. This indicates that the Explicit memory block eid=2 is a large block.

Next, see the object release ratio and the Explicit memory block size for each class of the objects in the large block (eid=2). In this example, there are five objects and the classes of these objects also include the classes provided in Java SE ([B, [C, and java.lang.String). In many cases, the classes provided in Java SE are transferred to the Explicit memory block referenced from the objects responsible for the large block. Therefore, if you specify settings so that the objects responsible for the large block are not transferred to the Explicit memory block, you can also prevent the transfer of the objects of the classes provided in Java SE. Also, if you set the classes provided in Java SE as the target of the functionality for specifying the classes to be excluded from the Explicit Memory Management functionality, the extent of impact is wide and also results in restrictions on the objects allocated to an ideal Explicit memory block. Therefore, do not set the classes provided in Java SE as the target of the functionality for specifying the classes to be excluded from the Explicit Memory Management functionality (however, excluding the cases in which all the objects of the classes provided in Java SE might be allocated to the Java heap).

The classes other than those provided with Java SE include `package1.session.StandardManager` and `framework.ut.impl.performList`. The object of `package1.session.StandardManager` is also used in the Explicit memory block `eid=1`, but `eid=1` is not a large block. This indicates that the objects of this class are not responsible for the large block. From this, you can identify that the object of `framework.ut.impl.performList` is responsible for the large block.

After identifying the object responsible for the large block, code the class of that object in the configuration file used with the functionality for specifying the classes to be excluded from the Explicit Memory Management functionality. For details on how to specify the code in the configuration file, see *7.13.3 Controlling application target of the Explicit Memory Management functionality by using a configuration file* in the *uCosminexus Application Server Expansion Guide*.

(5) Re-tuning the Java heap area and Explicit heap area

If you apply the object transfer control functionality and the functionality for specifying the classes to be excluded from the Explicit Memory Management functionality, the allocated area of the object changes and the memory size of the Java heap and Explicit heap also increases and decreases. Therefore, see the methods described at the following locations, and then re-tune the memory size for each area:

- *7.4 Java heap tuning*
- *7.5 Estimating the memory size of the Tenured area in Java heap*
- *7.6 Estimating the memory size of the New area in Java heap*
- *7.7 Determining the handling of objects that exist for a fixed time period in Java heap*
- *7.8 Deciding the maximum size or the initial size of Java heap*
- *7.9 Estimating the memory size of the Metaspace area in Java heap*
- *7.10 Analyzing the factors of Full GC using the extended verbosegc information*
- *7.11 Explicit heap tuning*
- *7.12 Estimating the memory size when using the Explicit Memory Management functionality in the application*
- *7.13 Determining the usage of the Explicit heap using the automatic allocation functionality of the Explicit Memory Management functionality*

7.15 Mechanism of G1 GC

This section explains the mechanism of G1 GC.

7.15.1 Overview of G1 GC

GC is a technique that automatically reclaims memory that a program has finished using, making that memory available to other programs.

Program processing stops while GC is occurring. For this reason, appropriate implementation of GC significantly impacts the processing performance of the system.

Java objects created by a program using the `new` operator are stored in the memory areas managed by the JavaVM. The time period from when the Java object is created until it is no longer needed is called the *lifespan* of the Java object.

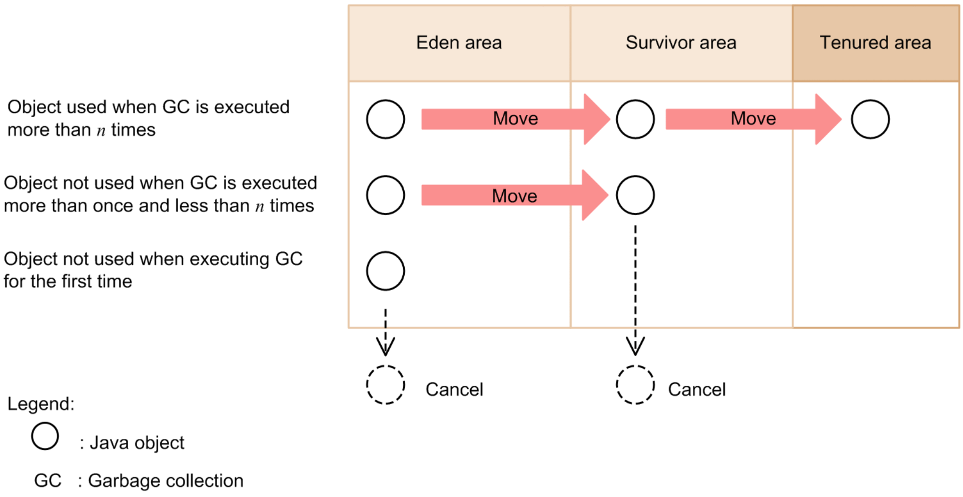
There are two types of Java objects, those with a short lifespan and those with a long lifespan. A Java application running on the server will create a large number of Java objects by request and response, transaction management, and other services. These Java objects have a short lifespan, as they are no longer needed when the processing ends. In contrast, the Java objects that are used continually while the application is running are considered to have a long lifespan.

To implement GC effectively, you need to reclaim memory in an efficient way by targeting objects with a short lifespan. Another key to maintaining system performance is to avoid unnecessary garbage collection for repeatedly used objects with long lifespans. An approach that achieves both these objectives is *generational GC*.

Generational GC manages Java objects in a *New* area which stores objects with a short lifespan, and a *Tenured* area which stores objects with a long lifespan. The New area is further divided into an *Eden* area for objects that were just created by the `new` operator, and a *Survivor* area for objects that have survived GC at least once. Java objects that have undergone GC in the New area a certain number of times are determined to have long-term utility and transferred to the Tenured area.

The following figure shows an overview of the memory spaces managed by generational GC and how Java objects move among them.

Figure 7–16: Overview of memory spaces managed by generational GC and Java objects


Eden area
Survivor area
Tenured area
Object used when GC is executed more than *n* times
Move
Move
Object not used when GC is executed more than once and less than *n* times
Move
Object not used when executing GC for the first time
Cancel
Cancel
Legend:
: Java object
GC : Garbage collection

There are three types of GC executed by generational GC under the G1 GC technique.

- Young GC

GC that applies to the Eden area and the Survivor area. Young GC is triggered when enough Java objects are created that the Eden area becomes full.

Note:

> On the Application Server, you can select serial GC or G1 GC (`UseG1GC`). You cannot use parallel GC, concurrent GC, or incremental GC.

- Mixed GC

  GC that applies to the Eden area, Survivor area, and part of the Tenured area. The occurrence of mixed GC is tied to object analysis processing called *Concurrent Marking*. Mixed GC is triggered when enough Java objects are created that the Eden area becomes full.

- Full GC

  GC that applies to all areas in the JavaVM including the Tenured area. Full GC is triggered when there is no more space available in the Tenured area, Metaspace area, or Humongous area.

Typically, young GC and mixed GC take less time to perform than Full GC.

The following explains GC processing using the example of a specific Java object (Object A).

Processing executed in the Eden area

> After creating an object A, the object A is destroyed if it is not used by the time young GC or mixed GC is first executed.
>
> If object A is used at the time young GC or mixed GC is first executed, it is moved from the Eden area to the Survivor area.

Processing executed in the Survivor area

> After young GC or mixed GC have been executed several times, object A that was moved to the Survivor area is moved from the Survivor area to the Tenured area. The threshold value used to determine whether the object is moved depends on the JavaVM options and the usage status of the Java heap. In Figure 7-16, the threshold is *n* times.
>
> After moving to the Survivor area, if object A is not used when young GC or mixed GC occurs for the *n*th time, then object A is discarded as part of the *n*th young GC or mixed GC.

Processing executed in the Tenured area

> If object A is moved to the Tenured area, it will not be discarded by subsequent young GC. This is because young GC targets the Eden area and Survivor area only. Mixed GC will cause some of the objects in the Tenured area to be discarded.

If the number of objects moved to the Tenured area is greater than the number of objects discarded by the mixed GC, the size of the Tenured area will increase. If mixed GC is unable to secure more space for the Tenured area, then Full GC will occur.

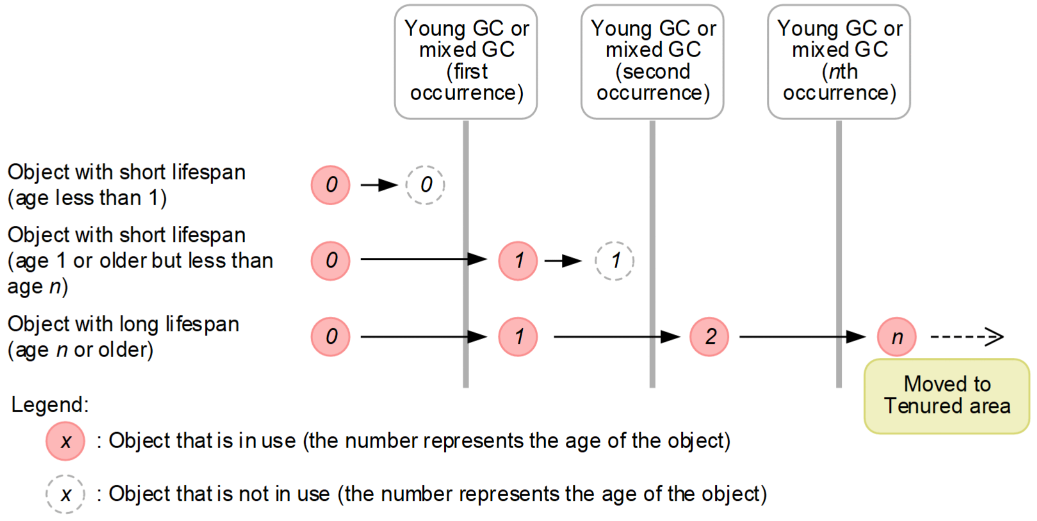
When tuning the JavaVM, you can prevent unnecessary objects from being moved to the Tenured area by setting the appropriate size and proportion of each memory space in the JavaVM options. This reduces the frequency with which Full GC occurs.

7.15.2 Relationship between the lifespan of an object and its age

The number of times young GC or mixed GC is executed for an object is called the *age* of the object.

The following figure shows the relation between the lifespan and the age of an object.

Figure 7–17: Relation between the lifespan and the age of an object



Young GC or mixed GC (first occurrence)
Young GC or mixed GC (second occurrence)
Young GC or mixed GC (nth occurrence)
Object with short lifespan (age less than 1)
Object with short lifespan (age 1 or older but less than age n)
Object with long lifespan (age n or older)
0
1
2
n
Moved to Tenured area
Legend:
x : Object that is in use (the number represents the age of the object)
x : Object that is not in use (the number represents the age of the object)

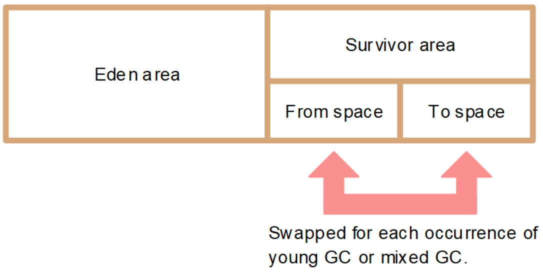
After the application has started, the initialization process has completed, and the young GC or mixed GC has been executed a certain number of times, objects with a long lifespan that will be needed for a long time move to the Tenured area. As a result, the Java heap stabilizes shortly after the application starts, and most of the Java objects that are created are objects that have a short lifespan. If the New area has been tuned appropriately, then the majority of objects created after the Java heap stabilizes are objects with a short lifespan that are collected during the first young GC or mixed GC.

7.15.3 GC mechanism that targets the New area

The JavaVM divides the memory space of the New area that is subject to young GC or mixed GC into the Eden area and the Survivor area for management purposes. The survivor area is further divided into a From space and a To space. The From space and the To space have the same memory size.

The following figure shows the configuration of the New area.

Figure 7–18: Configuraton of New area



The Eden area is the area in which objects created by the `new` operator are first stored. Memory is allocated from the Eden area when a program executes the `new` operator.

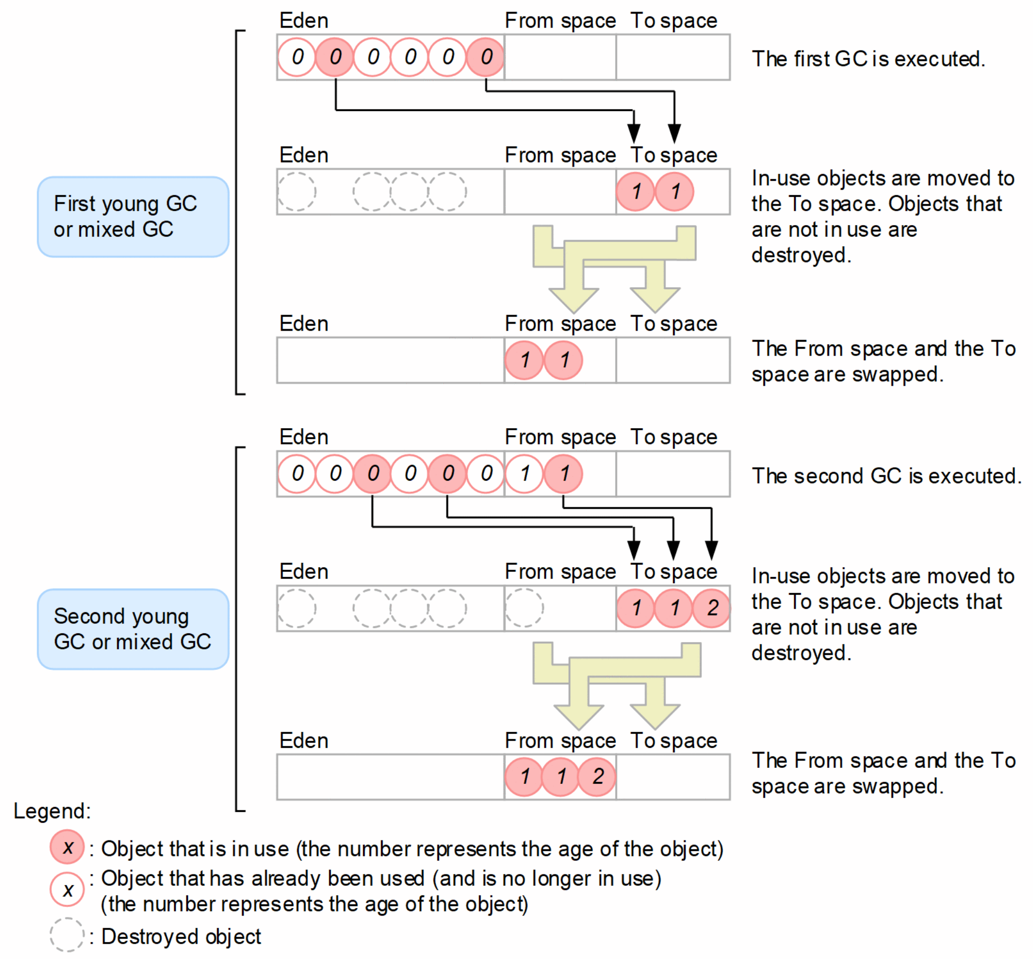
When the Eden area becomes full, young GC or mixed GC is executed and the following processing takes place:

1. The in-use Java objects in the Eden area and in the From space of the Survivor area are copied to the To space of the Survivor area. The Java objects that are not in use are destroyed.
2. The designations of the To space and From space of the Survivor area are swapped.

As a result, the Eden area and the To space become empty, and all in-use objects are now in the From space.

The following figure shows the movement of objects when young GC or mixed GC occurs.

Figure 7–19: Movement of objects when copy GC is executed

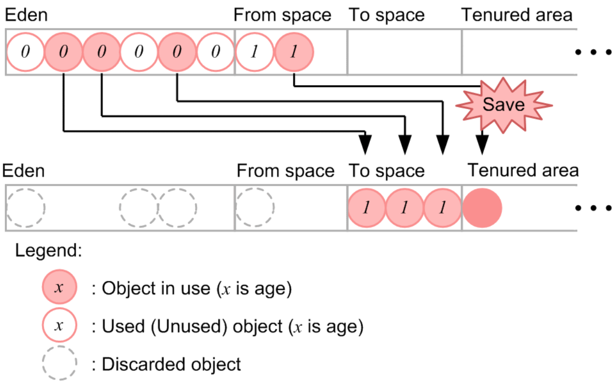

Eden
From space
To space
0 0 0 0 0 0
The first GC is executed.
First young GC or mixed GC
1 1
In-use objects are moved to the To space. Objects that are not in use are destroyed.
1 1
The From space and the To space are swapped.
0 0 0 0 0 0 1 1
The second GC is executed.
Second young GC or mixed GC
1 1 2
In-use objects are moved to the To space. Objects that are not in use are destroyed.
1 1 2
The From space and the To space are swapped.
Legend:
x : Object that is in use (the number represents the age of the object)
x : Object that has already been used (and is no longer in use) (the number represents the age of the object)
: Destroyed object

In this manner, objects that are in use move back and forth between the From space and the To space each time young GC or mixed GC occurs. However, the processing associated with moving objects with a long lifespan back and forth comes with several issues, including the load it imposes on the system. To prevent these issues, you can set a threshold value that governs how many times a Java object can be moved between the From and To spaces, and move objects whose age reaches this threshold to the Tenured area.

7.15.4 Saving objects

The process of moving a Java object whose age has not reached the threshold to the Tenured area is called *saving* the object. Saving of objects takes place when there is a large number of in-use objects in the Eden area and the From space when young GC or mixed GC is executed, and there is not enough memory space in the To space to accommodate all those objects. In this case, objects that could not be moved to the To space are moved to the Tenured area.

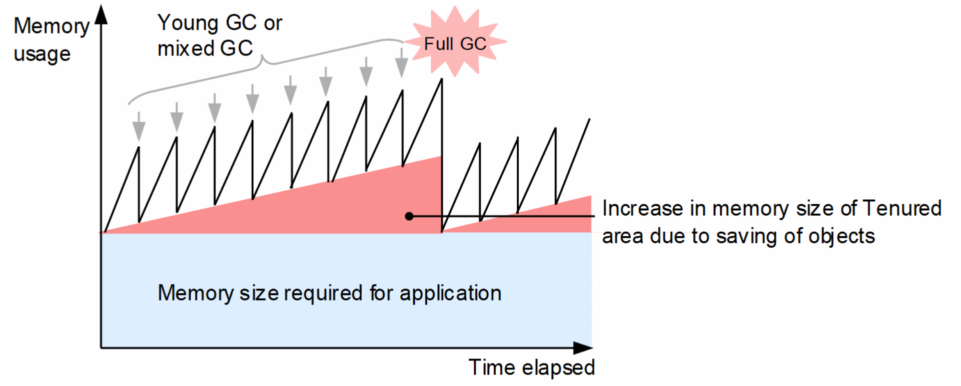
Figure 7–20: Saving objects



The saving of objects results in objects with short lifespans being stored in the Tenured area that are not meant to be stored in the Tenured area. If this process repeats, objects that were meant to be collected by young GC or mixed GC remain in the memory space. This increases the memory usage of the Java heap, ultimately triggering Full GC.

The following figure shows how memory usage changes when objects are saved.

Figure 7–21: Change in memory usage when objects are saved



When Full GC is executed, the application might pause for several seconds to several tens of seconds.

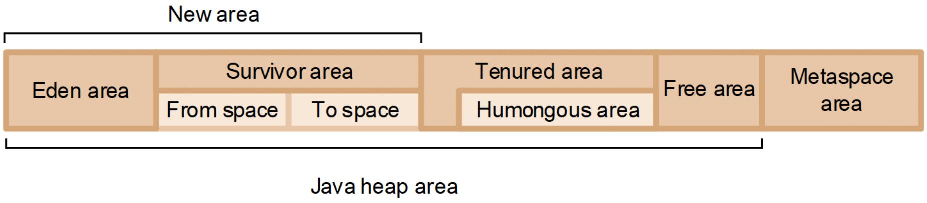
For this reason, when determining the configuration of the memory space and memory size, you must achieve a balance between the Eden area and the Survivor area to prevent objects from being saved.

7.15.5 Relationship between memory space and region

G1 GC divides the Java heap area into a *New* area which stores objects with a short lifespan, and a *Tenured* area which stores objects with a long lifespan. The New area is further divided into an *Eden* area for objects that were just created by the `new` operator, and a *Survivor* area for objects that have survived GC at least once. Objects that have undergone GC in the New area a certain number of times are determined to have a long lifespan and transferred to the Tenured area.

The following figure shows an overview of the memory spaces managed by G1 GC while an application is running. The Eden area, Survivor area, Tenured area, and Free area are collectively called the Java heap area.

Figure 7–22: Configuration of memory spaces managed by G1 GC during application execution



G1 GC does not allocate a Tenured area from the start. When the time comes to move an object to the Tenured area, G1 GC moves the object after allocating a region of the Free area to the Tenured area. G1 GC also expands or shrinks the New area after GC (hereinafter called *resizing*). When resizing entails expanding the New area, G1 GC allocates a region of the Free area to the New area. When resizing entails shrinking the New area, G1 GC recovers a region of the New area and allocates it to the Free area. When an object is created that is large in size, that object is stored in the Humongous area.

The following explains how each area is used:

- New area

  An area that stores objects with a short lifespan. The New area consists of the Eden are and the Survivor area. The New area is resized after GC to prepare for the next time GC occurs. For details, see *7.15.8 Young GC*.

- Eden area

  An area in which objects are stored first when created.

- Survivor area

  An area that stores the objects in the New area that were not collected when GC was executed. The Survivor area consists of a From space and a To space. The From space and the To space are the same size.

- Tenured area

  An area that stores objects with a long lifespan. Objects that have survived GC in the Survivor area a certain number of times are transferred to the Tenured area.

- Humongous area

  A part of the Tenured area that stores large objects. Objects in the Humongous are collected by Full GC or Concurrent Cleanup. For details about Concurrent Cleanup, see *7.15.9 Concurrent Marking (CM)*.

- Free area

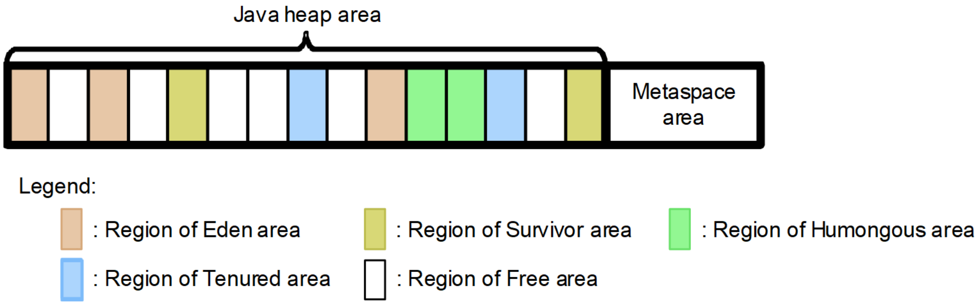
  An area that is not allocated to the New area or the Tenured area. The Free area is allocated to the New area or the Tenured area when the region of the Tenured area is too small and when resizing the New area after GC.

- Metaspace area

  An area that stores the loaded class information. Note that the Metaspace area is not managed at the region level. Objects in the Metaspace area are collected by Full GC only.

Figure 7-22 shows the areas of the Java heap area to be contiguous, but the areas are not necessarily contiguous in practice. G1 GC manages the Java heap area as blocks called *regions*, which are allocated in a distributed way as shown in the following figure.

Figure 7–23: Structure of memory space in practice



Each region is represented by the name of the area followed by the word *region*. For example, a region of the Eden area is called an *Eden region*, and a region of the New area is called a *New region*.

7.15.6 How regions are used

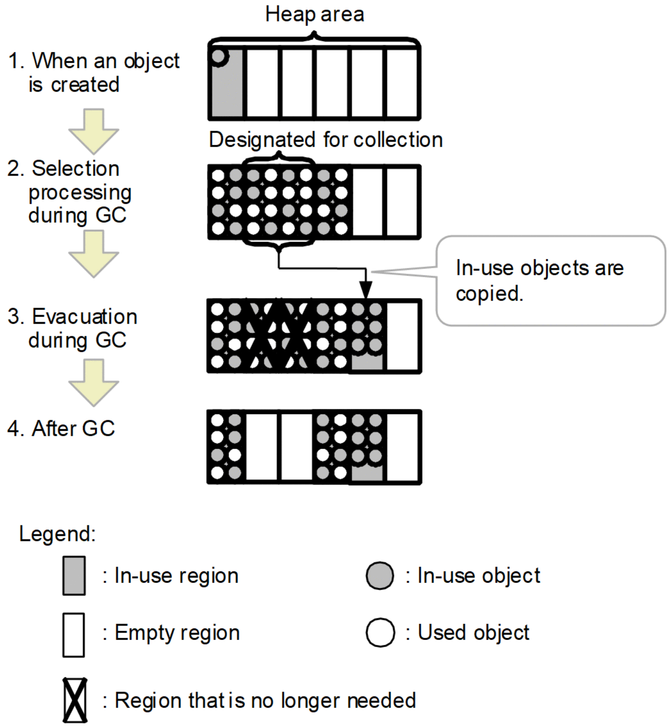
When an object is created, the JavaVM initially stores it in an Eden region. If the size of the object exceeds half the region size, it is considered a large object[#] and is allocated to multiple contiguous regions. These multiple contiguous regions are taken from the Free area and allocated to the Humongous area. Because only Full GC and Concurrent Cleanup collect objects from the Humongous area, G1 GC is poorly suited to applications that generate many large objects. For details, see *7.16.3(1) Concept of suppressing Full GC*.

The size of a single region is determined as a proportion of the initial size of the Java heap area at startup, and is a fixed size once started. The minimum size of one region is 1 MB, and the maximum size is 32 MB.

[#] Here, a *large object* is an object with a large number of instance fields or an object with a long array. When the region size is 1 MB, a large object is any object larger than 512 KB. Examples of that will qualify as large objects include an object with approximately 130,000 `int` instance fields, and an object containing a byte array with approximately 500,000 elements.

The following shows how regions are used in terms of the lifecycle from usage to collection.

Figure 7–24: How regions are used


Heap area
1. When an object is created
Designated for collection
2. Selection processing during GC
In-use objects are copied.
3. Evacuation during GC
4. After GC
Legend:
: In-use region
: In-use object
: Empty region
: Used object
: Region that is no longer needed

The preceding figure shows how regions are used. The following explains the processing illustrated by steps 1. to 4. in the figure in detail.

1. When an object is created

   When an object is created, it is initially stored in the Eden region. When creating a large object, the Free region is allocated to the Humongous region and the object is stored in the Humongous region. If there is no free space in the Eden region, the object is stored in a new Eden region. If there is no free space in the Humongous region, an additional region is allocated from the Free region to the Humongous region and the object is stored.

2. Selection processing during GC

   When the requirements for GC are met, the JavaVM selects the regions to subject to GC based on the criteria of the GC method being used.

3. Evacuation during GC

   In-use objects in regions subject to collection are copied to other regions in a process called *evacuation*. Because regions that have been evacuated will only contain used objects and objects that have been copied and are no longer needed, collection takes place at the region level.

4. After GC

   The collected regions are designated as Free regions and re-used.

7.15.7 GC implemented by G1 GC

There are three types of GC executed by generational GC under the G1 GC technique.

1. Young GC

   GC that applies to the New area. The types of young GC are *young GC (normal)*, and *young GC (initial-mark)* in which marking is performed as part of young GC. For details, see *7.15.8 Young GC*. If the term young GC is used along, the matter being described relates to both young GC (normal), and young GC (initial-mark). The suffixes (normal) and (initial-mark) are used when there is a need to distinguish between young GC (normal) and young GC (initial-mark). For young GC (normal), log data is output to the log file with `YoungGC` specified as the GC type. Young GC occurs when the creation of a Java object causes the Eden area to run out of space.

2. Mixed GC

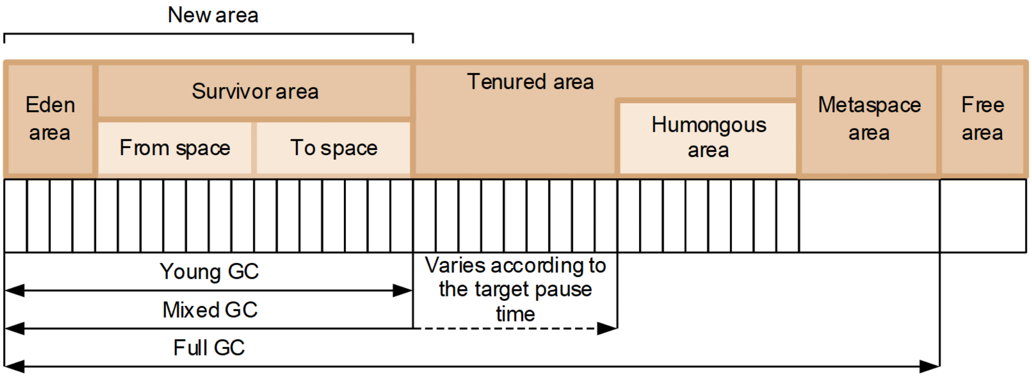
   GC that applies to the New area and the Tenured area. Mixed GC applies selectively to Tenured regions depending on the target pause time. For details, see *7.15.10 Mixed GC*. Like young GC, mixed GC occurs when the creation of a Java object causes the Eden area to run out of space, but can also be triggered by the results of analysis called *Concurrent Marking* that determines whether objects are in use. For details, see *7.15.9 Concurrent Marking (CM)*. If analysis is lacking or the analysis results predict that mixed GC would not be very effective, young GC (normal) is triggered instead.

3. Full GC

   GC that applies to all areas in the JavaVM including the Tenured area, Metaspace area, and Humongous area. For details, see *7.15.11 Full GC*. Full GC is triggered when there are no more Free regions available to be allocated to the Tenured area, Metaspace area, or Humongous area.

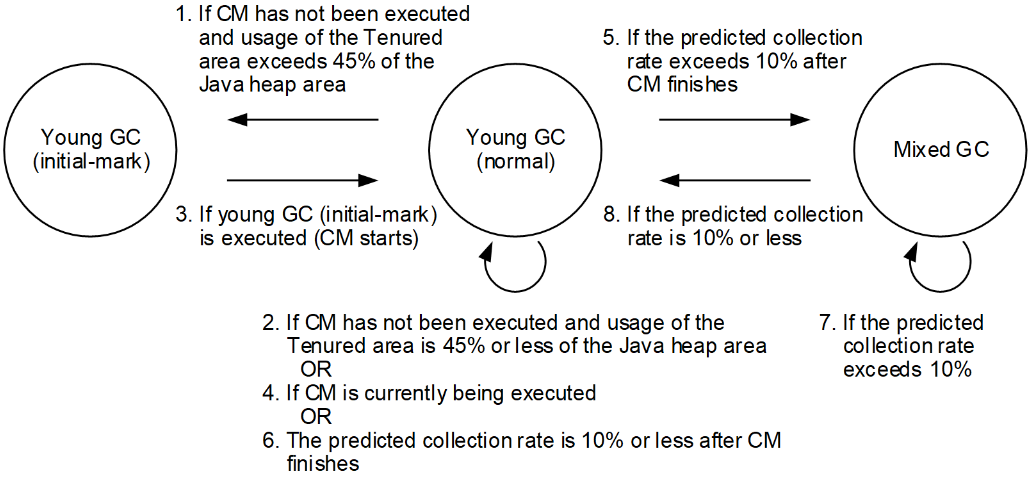
The following figure summarizes the areas targeted by each GC method.

Figure 7–25: Areas targeted by each GC method



The following figure shows the status transitions undergone by each GC method.

Figure 7–26: Status transitions of each GC method



1. If CM has not been executed and usage of the Tenured area exceeds 45% of the Java heap area
5. If the predicted collection rate exceeds 10% after CM finishes
Young GC (initial-mark)
Young GC (normal)
Mixed GC
3. If young GC (initial-mark) is executed (CM starts)
8. If the predicted collection rate is 10% or less
2. If CM has not been executed and usage of the Tenured area is 45% or less of the Java heap area
OR
4. If CM is currently being executed
OR
6. The predicted collection rate is 10% or less after CM finishes
7. If the predicted collection rate exceeds 10%

1. After young GC (normal) is executed, if Concurrent Marking (CM) has not been executed and usage of the Tenured area exceeds 45% of the Java heap area, the status transitions to one in which young GC (initial-mark) is executed the next time GC occurs.

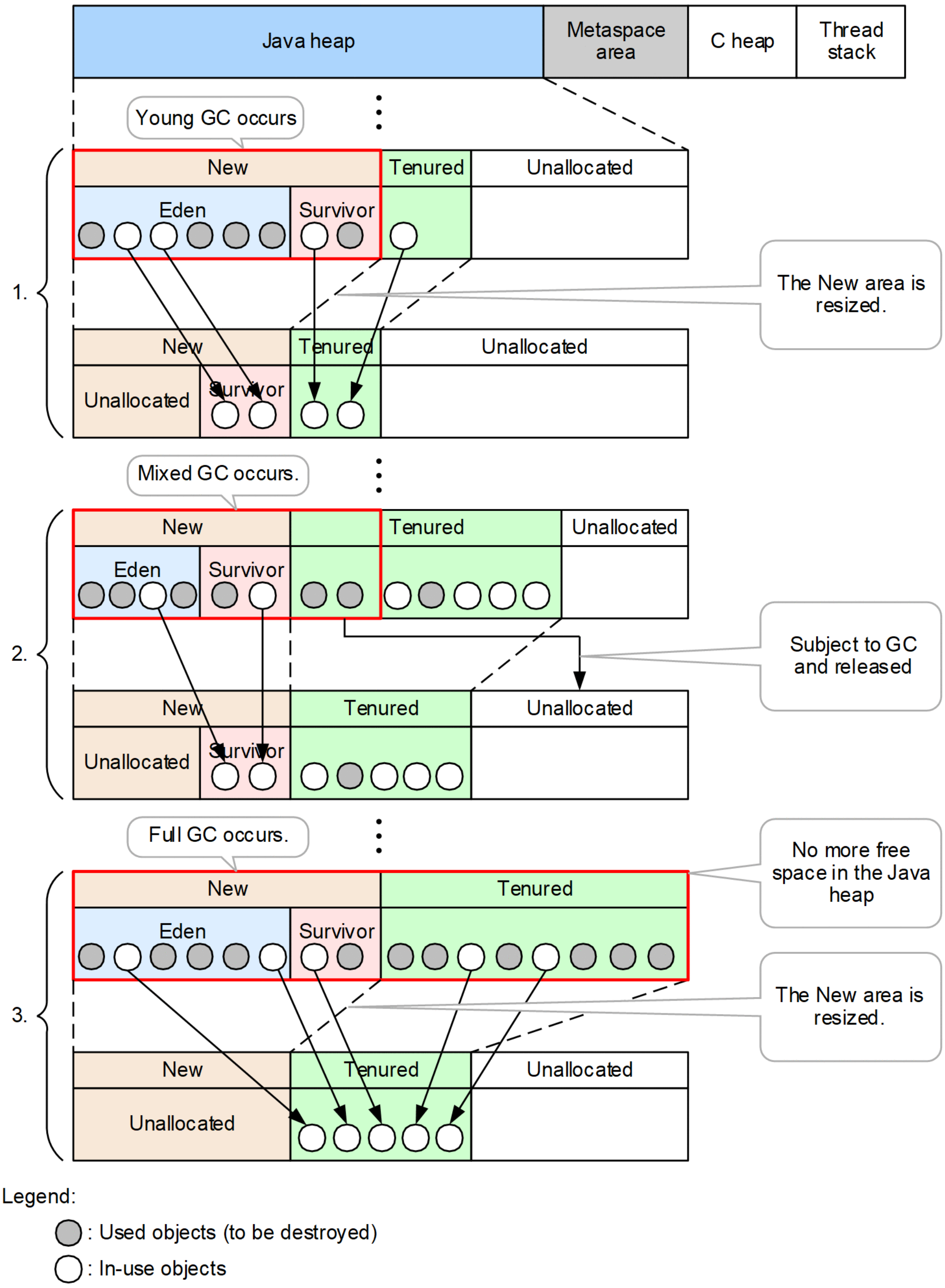
2. After young GC (normal) is executed, if Concurrent Marking (CM) has not been executed and usage of the Tenured area does not exceed 45% of the Java heap area, the status remains unchanged and young GC (normal) is executed the next time GC occurs.
3. After young GC (initial-mark) is executed, the status transitions to one in which young GC (normal) and CM are executed in parallel the next time GC occurs.
4. While CM is being executed, the status remains one in which young GC (normal) will be executed.
5. If the predicted collection rate of young GC (normal) immediately after CM finishes exceeds 10%, the status transitions to one in which mixed GC is executed the next time GC occurs.
6. If the predicted collection rate is 10% or less, the status remains unchanged and young GC (normal) is executed the next time GC occurs
7. After mixed GC is executed, if the predicted collection rate exceeds 10%, the status remains unchanged and mixed GC is executed the next time GC occurs.
8. After mixed GC is executed, if the predicted collection rate is 10% or less, the status changes and young GC (normal) is executed the next time GC occurs.

Note that in the following circumstances, the applicable GC method will be implemented when the conditions are met regardless of the GC status.

- If young GC or mixed GC is executed but cannot secure enough space, then Full GC is executed. The status then changes to one in which young GC (normal) is executed the next time GC occurs.
- When securing the space for a large object, if CM has not been executed and the total of the used size of the Tenured area and the size of the object exceeds 45% of the Java heap area, then young GC (initial-mark) is executed.

The following explains G1 GC processing using the example of a Java object.

Figure 7–27: Flow of G1 GC



1. Young GC

   As shown in the preceding figure, young GC is triggered when there are no longer any empty regions allocated to the New area. Young GC moves in-use objects to the regions allocated to the Survivor area, and releases used objects at the region level. Like the copy GC method of serial GC, objects that are in use when young GC occurs move back and forth between the regions allocated to the Survivor area. After moving a certain number of times, these objects are moved to a region allocated to the Tenured area. After young GC, the system estimates the time required for the next GC based on how long GC took, and resizes the New area accordingly. In the example in the preceding figure, the New area is reduced in size because GC took longer than the estimated time.

2. Mixed GC

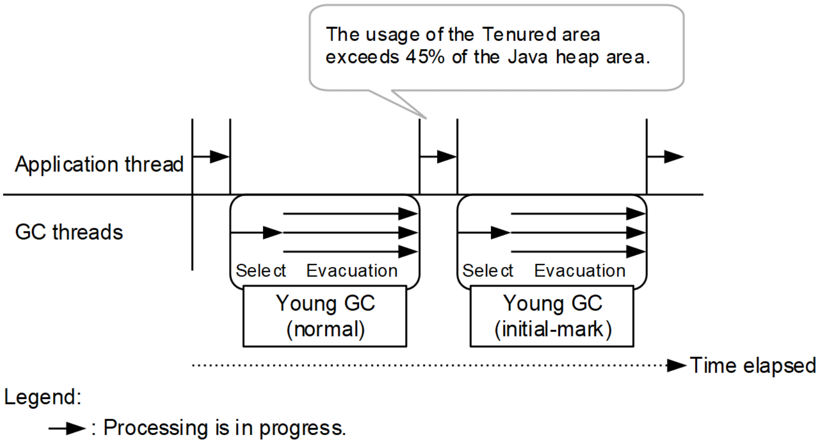
   Mixed GC is triggered when the usage of the Tenured area increases. In addition to the regions allocated to the New area, mixed GC targets some of the regions allocated to the Tenured area as the target pause time permits. GC prioritizes those regions allocated to the Tenured area that would free the most space based on analysis of which objects are in use conducted in parallel with the application. If analysis has not been performed to a sufficient level or the analysis results predict that mixed GC would not be very effective, then mixed GC is not triggered.

3. Full GC

   If there are no more empty regions in the Java heap and mixed GC is not triggered, then Full GC occurs for the entire Java heap.

7.15.8 Young GC

Figure 7–28: Flow of young GC



(1) Triggers

1. Young GC (normal)

   Young GC (normal) occurs when there is not enough space for an object in the Eden area. It also occurs when the Humongous area does not have enough space to accommodate a large object.

2. Young GC (initial-mark)

   If the used size of the Tenured area exceeds 45% of the Java heap area when young GC (normal) finishes, young GC (initial-mark) is executed when the trigger in 1. is satisfied. When securing the space for a large object, if the total of the used size of the Tenured area and the size of the object exceeds 45% of the Java heap area, then young GC (initial-mark) is executed.

(2) Target

New area

(3) Processing

- When young GC occurs, a single thread performs region selection, followed by multi-threaded evacuation.
- Evacuation in young GC entails moving in-use objects from the Eden area and the From space to the To space or the Tenured area, and reclaiming the Eden area and the From space. The mechanisms for moving and recovering objects are the same as for copy GC. For details about copy GC, see *7.2.3 Mechanism of copy GC*.
- Young GC estimates the GC pause time based on information about GC executed in the past, and resizes the New area so that the predicted pause time of the next GC falls within the target pause time.
- The New area has a minimum size and a maximum size, and is resized within these bounds. Because all GC methods target the New area, the GC pause time cannot be shorter than the GC pause time if the New area is the minimum size.

- When young GC occurs after CM has completed, if the predicted collected size exceeds 10% of the Java heap area, the system determines whether to perform mixed GC for the next GC. If mixed GC is selected for the next GC, then the JavaVM predicts that GC will mainly target Tenured regions and resizes the New area accordingly.
- The evacuation processing of young GC (initial-mark) marks objects that are directly referenced by local variables and in-use objects during evacuation. The Concurrent Marking processing uses the results of this marking. For details about Concurrent Marking, see *7.15.9 Concurrent Marking (CM)*.

(4) Processing results

Eden area: The objects are collected and the area is left empty. The Eden area is resized after GC has completed.

Survivor area: The objects in the From space are collected and the space is left empty. The Survivor area is resized after GC has completed.

Tenured area: Objects determined to have long-term utility are moved to the Tenured area.

Humongous area: Unchanged.

Metaspace area: Unchanged.

Free area: The Free area increases or decreases in size as a result of the resizing that occurs after GC.

(5) Pausing of applications

Applications are paused.

(6) Relationship with other GC methods

CM: Not executed during young GC.

Mixed GC: Not executed during young GC.

Full GC: If the requirements for Full GC are met while young GC is in progress, young GC is canceled and Full GC is executed.

(7) Supplementary notes

- Related options

  You can use the `-XX:ParallelGCThreads` option to change the number of threads involved in evacuation. By increasing the number of threads, you can reduce the time young GC takes to complete. If you do not specify this option, GC uses the default value of the `-XX:ParallelGCThreads` option. For details about the `-XX:ParallelGCThreads` option, see *14.5 Java HotSpot VM options that can be specified in Cosminexus* in the *uCosminexus Application Server Definition Reference Guide*.
- Confirmation method

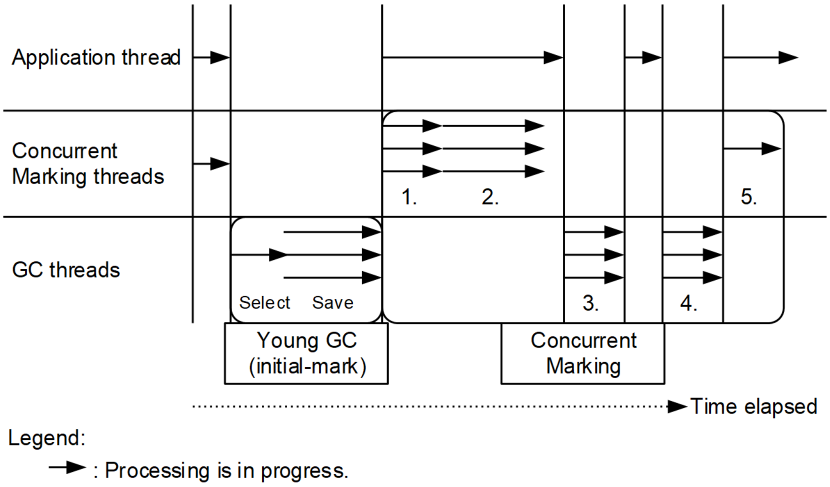
  You can confirm that young GC has occurred when `YoungGC` or `YoungGC(initial-mark)` appears as the GC type in the log file. You can also confirm the resizing of the New area based on how the sizes of the Eden area and the survivor area have changed.

```
[VGl]<Wed Jun 12 11:21:10 2013>[Young GC 899070K/899072K(1048576K)->50175
5K/501760K(1048576K), 0.0931560 secs][Status:-][G1GC::Eden: 389120K(389120
K)->0K(397312K)][G1GC::Survivor: 41984K->41984K][G1GC::Tenured: 459776K->4
59776K][G1GC::Humongous: 2048K->2048K][G1GC::Free: 609536K->607232K][Metas
pace: 3634K(4492K, 4492K)->3634K(4492K, 4492K)][class space: 356K(388K, 3
88K)->356K(388K, 388K)][cause:G1EvacuationPause][RegionSize: 1024K][Target
: 0.2000000 secs][Predicted: 0.2495800 secs][TargetTenured: 0K][Reclaimabl
e: 0K(0.00%)][User: 0.0156250 secs][Sys: 0.0312500 secs][IM: 729K, 928K, 0
K][TC: 509][DOE: 16K, 171][CCI: 2301K, 49152K, 2304K]
```

In the preceding log file entry, the size of the New area before GC is 389120K + 41984K = 431104K, and the size of the New area after GC is 397312K + 41984K = 439296K. This shows that the New area has been extended. For details about how to interpret this information, see *-XX:[+|-]HitachiVerboseGC (Option for extended verbosegc information output)* in the *uCosminexus Application Server Definition Reference Guide*.

7.15.9 Concurrent Marking (CM)

Figure 7–29: Flow of Concurrent Marking



(1) Triggers

Concurrent Marking (CM) broadly consists of the following five types of processing, the trigger for each of which is explained. Note that the labels 1. to 5. correspond to 1. to 5. in the preceding figure.

1. Concurrent Root Region Scan:
   Executed after young GC (initial-mark) is completed.
2. Concurrent Mark:
   Executed immediately after 1. has completed.
3. Remark:
   Executed when there is an opportunity to pause the application after 2. has completed.
4. Cleanup:
   Executed when there is an opportunity to pause the application after 3. has completed.
5. Concurrent Cleanup:
   Executed immediately after 4. has completed.

(2) Target

New area, Tenured area, and Humongous area

(3) Processing

CM broadly consists of the following five types of processing. The following describes each type of processing in detail.

1. Concurrent Root Region Scan:

   Processing that involves marking objects in the Survivor area that are directly referenced by local variables and in-use objects. This processing uses multiple threads.

2. Concurrent Mark:

   Processing that involves marking objects that are referenced by objects marked by young GC (initial-mark) and by the marking in 1. This processing uses multiple threads.

3. Remark:

   Processing that re-marks objects whose referential relationships changed during the marking in 2. This processing uses multiple threads.

4. Cleanup:

   Processing that determines the total size of in-use objects in each region. Cleanup processing also initializes markings in preparation for the next CM phase. This processing uses multiple threads.

5. Concurrent Cleanup:

   Processing that reclaims regions that do not contain any in-use objects. This processing is single-threaded.

(4) Processing results

Eden area:

Areas corresponding to in-use objects are marked.

If Eden regions are reclaimed by the processing in 5., the size of the area is reduced.

Survivor area:

Areas corresponding to in-use objects are marked.

If Survivor regions are reclaimed by the processing in 5., the size of the area is reduced.

Tenured area:

Areas corresponding to in-use objects are marked.

If Tenured regions are reclaimed by the processing in 5., the size of the area is reduced.

Humongous area:

Areas corresponding to in-use objects are marked.

If the Humongous region is reclaimed by the processing in 5., the size of the area is reduced.

Metaspace area:

Unchanged.

Free area:

If regions that do not contain any in-use objects are reclaimed, the size of the Free area increases.

(5) Pausing of applications

1. Concurrent Root Region Scan:

   Executed in a CM thread without pausing the application.

2. Concurrent Mark:

   Executed in a CM thread without pausing the application.

3. Remark:

Executed in a VM thread with the application paused.

4. Cleanup:

   Executed in a VM thread with the application paused.

5. Concurrent Cleanup:

   Executed in a CM thread without pausing the application.

Although the application is not paused, the processing associated with young GC (normal), Full GC, or securing the new region will be paused if Full GC occurs or a new region is secured between the conclusion of the processing in 4. and the conclusion of the processing in 5. This pause continues until the conclusion of the processing in 5.

(6) Relationship with other GC methods

Note: Young GC (initial-mark) will not occur during CM processing.

1. Concurrent Root Region Scan:
   - Young GC (normal): Not executed.
   - Mixed GC: Not executed.
   - Full GC: Not executed.
2. Concurrent Mark:
   - Young GC (normal): If triggered during the processing in 2., the processing in 2. is paused. This processing resumes when young GC (normal) has completed.
   - Mixed GC: Not executed.
   - Full GC: If triggered during the processing in 2., the processing in 2. is paused. The results of Concurrent Mark to that point are discarded.
3. Remark:
   - Young GC (normal): Not executed.
   - Mixed GC: Not executed.
   - Full GC: Not executed.
4. Cleanup:
   - Young GC (normal): Not executed.
   - Mixed GC: Not executed.
   - Full GC: Not executed.
5. Concurrent Cleanup:
   - Young GC (normal): If triggered during the processing in 5., young GC (normal) waits until the processing in 5. has completed.
   - Mixed GC: Not executed.
   - Full GC: If triggered during the processing in 5., Full GC waits until the processing in 5. has completed.

(7) Supplementary notes

- Related options

  You can use the `-XX:ConcGCThreads` option to specify the number of threads to use for CM processing. Increasing the number of threads used for CM reduces the throughput of the application. If you do not specify this option, CM uses the default value of the `-XX:ConcGCThreads` option. For details about the

`-XX:ConcGCThreads` option, see *14.5 Java HotSpot VM options that can be specified in Cosminexus* in the *uCosminexus Application Server Definition Reference Guide*.

- Confirmation method

  You can confirm that Concurrent Root Region Scan, Concurrent Mark, and Concurrent Cleanup have occurred by looking for log entries that start with the `VCM` identifier. The following shows an example of log entries related to Concurrent Mark processing:

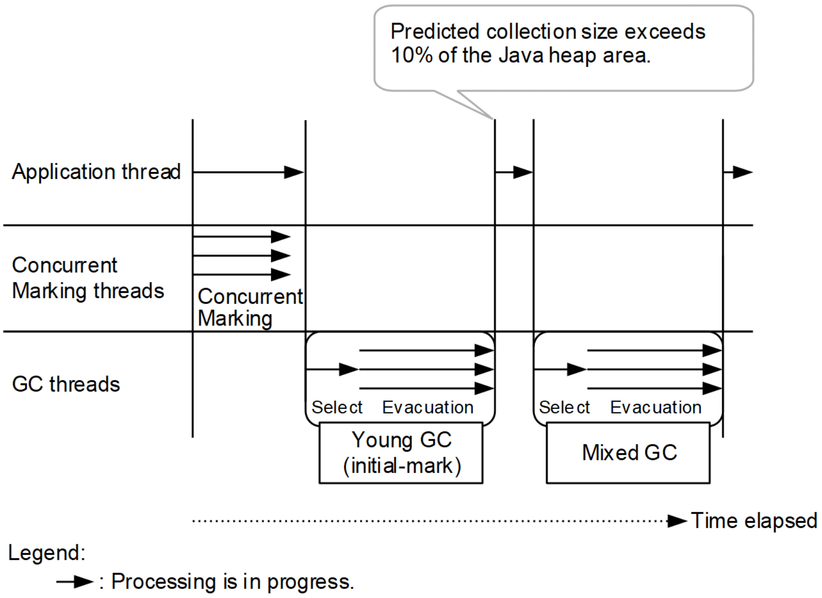
```
[VCM]<Wed Jul 31 11:45:23 2013>[Concurrent Mark Start][User: 0.0000000 sec
s][Sys: 0.0000000 secs]
[VCM]<Wed Jul 31 11:45:31 2013>[Concurrent Mark End][User: 0.0321549 secs]
[Sys: 0.0129454 secs]
```

Because `VCM` log entries span multiple lines, `Start` is output when CM starts and `End` when it ends. The `End` log entry outputs the CPU usage time from the start to the end of CM processing, with the `Start` log entry showing `0`. For details about how to interpret this information, see *-XX:[+|-]HitachiVerboseGC (Option for extended verbosegc information output)* in the *uCosminexus Application Server Definition Reference Guide*.

Because the application is paused during Remark and Cleanup processing, this processing is logged by a `VG1` log entry. You can identify Remark and Cleanup processing by the labels `CM Remark` and `CM Cleanup` representing the GC type in the GC log entry.

7.15.10 Mixed GC

Figure 7–30: Flow of mixed GC



(1) Trigger

If the predicted collection size exceeds 10% of the Java heap area, mixed GC is scheduled as the method of the next GC. This judgment is made at the conclusion of young GC (normal) immediately following CM, or at the conclusion of mixed GC.

(2) Target

Part of the New area and the Tenured area.

(3) Processing

- When mixed GC occurs, single-threaded region selection processing is performed together with multi-threaded Evacuation processing.
- Mixed GC targets the New area, and some of the Tenured area as long as the estimated pause time is within the target pause time.
- The Evacuation processing of mixed GC is the same as the Evacuation processing of young GC with respect to the New area. For details, see *7.15.8 Young GC*.
- When the Tenured area is added as a GC target, mixed GC re-allocates the in-use objects in the Tenured region to another Tenured region.
- If the predicted collection size still exceeds 10% of the Java heap area after mixed GC, mixed GC is selected again as the next GC, with young GC executed as usual if the requirements for mixed GC are not met.

(4) Processing results

Eden area:

The objects are collected and the area is left empty. The Eden area is resized after GC has completed.

Survivor area:

The objects in the From space are collected and the space is left empty. The Survivor area is resized after GC has completed.

Tenured area:

Objects determined to have long-term utility are moved to the Tenured area.

Objects in the area added as a GC target are collected.

Humongous area:

Unchanged.

Metaspace area:

Unchanged.

Free area:

The Free area increases or decreases in size as a result of the resizing that occurs after GC.

(5) Pausing of applications

Applications are paused.

(6) Relationship with other GC methods

CM: Not executed during mixed GC.

Young GC: Not executed during mixed GC.

Full GC: If the requirements for Full GC are met while mixed GC is in progress, mixed GC is canceled and Full GC is executed.

(7) Supplementary notes

- Related options

  You can use the `-XX:ParallelGCThreads` option to change that number of threads involved in evacuation. By increasing the number of threads, you can reduce the time mixed GC takes to complete. If you do not specify this option, the number of threads is based on the number of CPUs recognized by the OS. For details about the `-XX:ParallelGCThreads` option, see *14.5 Java HotSpot VM options that can be specified in Cosminexus* in the *uCosminexus Application Server Definition Reference Guide*.

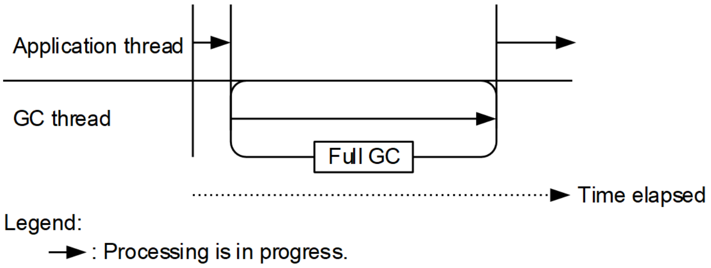
- Confirmation method

  You can confirm that mixed GC has occurred when `Mixed GC` appears as the GC type in the log file. The `TargetTenured` item shows the size of the Tenured area selected by mixed GC. The `Reclaimable` item shows the estimated size of collectable objects. For details about how to interpret this information, see *-XX:[+|-]HitachiVerboseGC (Option for extended verbosegc information output)* in the *uCosminexus Application Server Definition Reference Guide*.

```
[VG1]<Wed Jun 12 11:21:10 2013>[Mixed GC 899070K/899072K(1048576K)->501742
K/501760K(1048576K), 0.0931560 secs][Status:-][G1GC::Eden: 389120K(389120K
)->0K(397312K)][G1GC::Survivor: 41984K->41984K][G1GC::Tenured: 459776K->45
9776K][G1GC::Humongous: 2048K->2048K][G1GC::Free: 609536K->607232K][Metasp
ace: 3634K(4492K, 4492K)->3634K(4492K, 4492K)][class space: 356K(388K, 388
K)->356K(388K, 388K)][cause:G1EvacuationPause][RegionSize: 1024K][Target:
0.2000000 secs][Predicted: 0.2495800 secs][TargetTenured: 2048K][Reclaimab
le: 17703K(1.69%)][User: 0.0156250 secs][Sys: 0.0312500 secs][IM: 729K, 92
8K, 0K][TC: 509][DOE: 16K, 171][CCI: 2301K, 49152K, 2304K]
```

7.15.11 Full GC

Figure 7–31: Flow of Full GC



(1) Triggers

Full GC is executed when there is no more Free area remaining and Evacuation processing cannot secure an area to which to copy evacuated objects, or when there is not enough free space to copy an object to the Humongous area. It is also executed when there is no more free space in the Metaspace area, and when `System.gc()` is called.

(2) Target

New area, Tenured area, Humongous area, and Metaspace area

(3) Processing

- Full GC is executed using a single thread after pausing the application thread.

- When Full GC is triggered, the application pauses until GC is completed without regard to the target pause time.
- Full GC processing is the same as the Full GC processing performed as part of serial GC. If Full GC is unable to secure the necessary free space, then an `OutOfMemory` exception occurs. For details, see *7.2 Mechanism of serial GC*.
- Typically, young GC and mixed GC take less time to complete than Full GC.

(4) Processing results

Eden area: The objects are collected and the area is left empty.

Survivor area: In-use objects are moved to the Tenured area and the area is left empty.

Tenured area: In-use objects are reorganized.

Humongous area: In-use objects are reorganized.

Metaspace area: In-use objects are reorganized.

Free area: Increases in size due to reclaimed regions.

(5) Pausing of applications

Applications are paused.

(6) Relationship with other GC methods

CM: Not executed during Full GC.

Young GC: Not executed during Full GC.

Mixed GC: Not executed during Full GC.

(7) Supplementary notes

- Confirmation method

  You can confirm that Full GC has occurred when `Full GC` appears as the GC type in the log file. Because Full GC makes no predictions regarding subsequent GC, `0` appears as the predicted pause time and predicted collection size.

```
[VG1]<Wed Jan 15 12:51:32 2014>[Full GC 130443K/131072K(131072K)->55462K/
56320K(131072K), 2.3265610 secs][Status:-][G1GC::Eden: 0K(43008K)->0K(4300
8K)][G1GC::Survivor: 0K->0K][G1GC::Tenured: 131072K->56320K][G1GC::Humongo
us: 0K->0K][G1GC::Free: 0K->74752K][Metaspace: 3634K(4492K, 4492K)->3634K(
4492K, 4492K)][class space: 356K(388K, 388K)->356K(388K, 388K)][cause:ObjA
llocFail][RegionSize: 1024K][Target: 0.2000000 secs][Predicted: 0.0000000
secs][TargetTenured: 0K][Reclaimable: 0K(0.00%)][User: 2.1700000 secs][Sys
: 0.0000000 secs][IM: 277185K, 261856K, 44544K][TC: 1168][DOE: 0K, 0][CCI
: 5808K, 49152K, 5952K]
```

7.16 G1 GC tuning

This section explains how to tune the system for G1 GC. Perform the tuning described in this section when creating the system, and apply it to the live environment only after verifying that the system requirements are met.

The following table lists the options related to the size of the Java heap area. The numbers 1 to 8 in the table correspond to their counterparts in the figure.

Table 7–9: Options that specify the size of the areas in the Java heap area and the size of the Metaspace area

No.	Option name	Description
1	`-Xms`*size*	Sets the initial size of the Java heap#1.
2	`-Xmx`*size*	Sets the maximum size of the Java heap#1.
3	`-XX:NewSize=`*value*	Sets the minimum size of the New area#2.
4	`-XX:MaxNewSize=`*value*	Sets the maximum size of the New area#2.
5	`-XX:NewRatio=`*value*	Sets the ratio of the New area to the Java heap area#2. When *value* is `2`, the ratio of the New area to the Tenured area is 1:2.
6	`-XX:SurvivorRatio=`*value*	Sets the maximum ratio of the Survivor area to the New area. The maximum size of the Survivor area relative to the New area is determined by the following formula: *size-of-Survivor-area*#3*=size-of-New-area / survivor-ratio*
7	`-XX:MetaspaceSize=`*size*	Sets the initial size of the Metaspace area#4.
8	`-XX:MaxMetaspaceSize=`*size*	Sets the maximum size of the Metaspace area#4.

#1

We recommend that you specify the same value for the `-Xms` and `-Xmx` options.

#2

Specifying options 3 to 5 limits the extent to which the New area can be resized, negating the advantage of G1 GC by which the GC process can be completed within the target pause time. For this reason, we recommend that you do not specify these options when using G1 GC. If you omit options 3 to 5, the initial size of the New area will be 0.2 times the initial size of the Java heap. The initial size of the Eden area is based on the default value of the `SurvivorRatio` option, which is `8`.

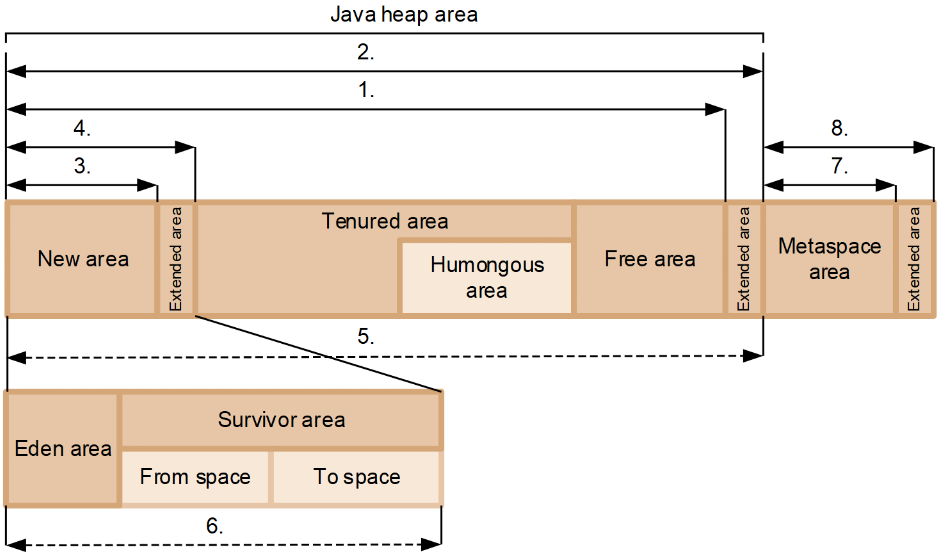
#3

The "size of the Survivor area" in the description in the table refers to the size of either the To space or the From space. It is not the total size of the To space and the From space.

#4

We recommend that you specify the same value for the `-XX:MetaspaceSize` option and the `-XX:MaxMetaspaceSize` option.

Figure 7–32: Scope of options that specify the size of the areas in the Java heap area and the size of the Metaspace area



Legend:

X.: Numbers correspond to their counterparts in the table.

←→ : Scope of options that specify size

←---→ : Scope of options that specify ratio

The following table shows the options used for G1 GC tuning. For details about each option, see *14.5 Java HotSpot VM options that can be specified in Cosminexus* in the *uCosminexus Application Server Definition Reference Guide*.

Table 7–10: G1 GC tuning options

No.	Option name	Default value	Description
1	`-XX:MaxGCPauseMillis=`*value*	200	Target pause time [ms]
2	`-XX:ParallelGCThreads=`*value*	Number of CPUs# (The number of CPUs recognized by the OS)	The number of threads used for Evacuation processing
3	`-XX:ConcGCThreads=`*value*	(`ParallelGCThreads` + 2) / 4	The number of threads used for CM processing

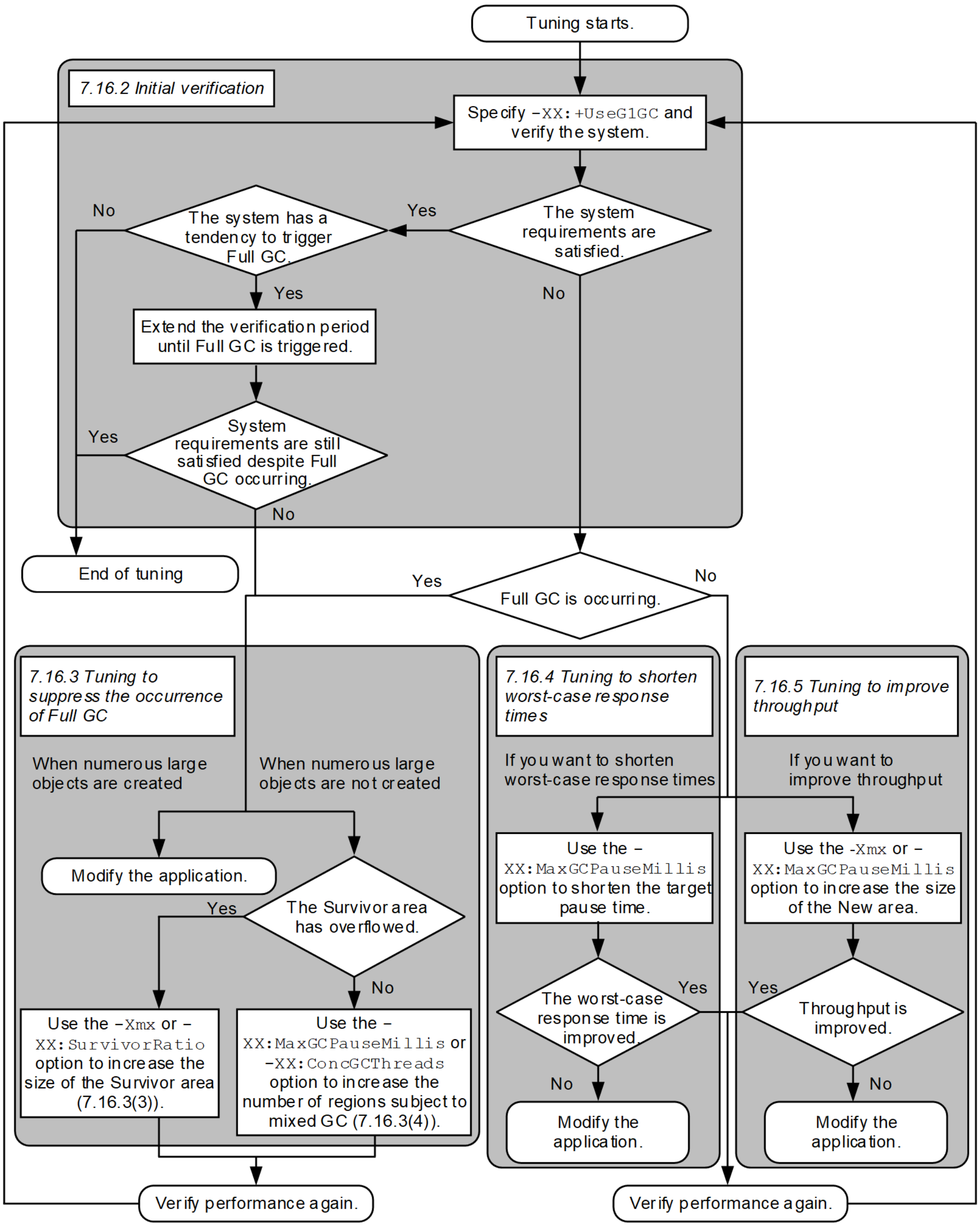
#

The formula depends on the number of CPUs in the execution environment.

7.16.1 Tuning procedure

The following figure shows the overall flow of the tuning process.

Figure 7–33: Overall flow of tuning process


Tuning starts.
7.16.2 Initial verification
Specify –XX:+UseG1GC and verify the system.
The system requirements are satisfied.
Yes
No
The system has a tendency to trigger Full GC.
No
Yes
Extend the verification period until Full GC is triggered.
System requirements are still satisfied despite Full GC occurring.
Yes
No
End of tuning
Full GC is occurring.
Yes
No
7.16.3 Tuning to suppress the occurrence of Full GC
When numerous large objects are created
When numerous large objects are not created
Modify the application.
The Survivor area has overflowed.
Yes
No
Use the –Xmx or –XX:SurvivorRatio option to increase the size of the Survivor area (7.16.3(3)).
Use the –XX:MaxGCPauseMillis or –XX:ConcGCThreads option to increase the number of regions subject to mixed GC (7.16.3(4)).
Verify performance again.
7.16.4 Tuning to shorten worst-case response times
If you want to shorten worst-case response times
Use the –XX:MaxGCPauseMillis option to shorten the target pause time.
The worst-case response time is improved.
Yes
No
Modify the application.
7.16.5 Tuning to improve throughput
If you want to improve throughput
Use the -Xmx or –XX:MaxGCPauseMillis option to increase the size of the New area.
Throughput is improved.
Yes
No
Modify the application.
Verify performance again.

The following explains each part of the tuning process in detail.

7.16.2 Initial verification

Figure 7–34: Flow of initial verification

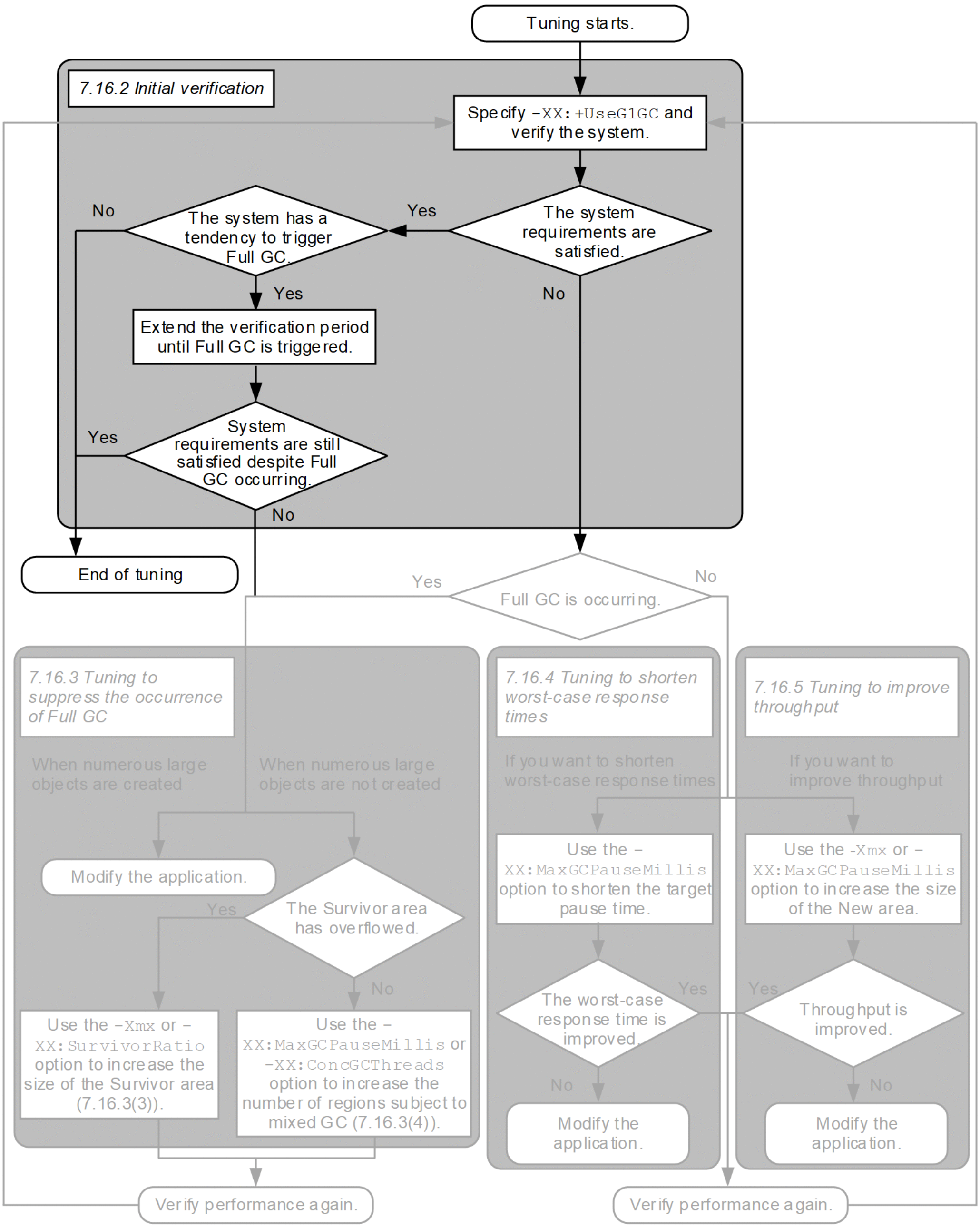

Tuning starts.
7.16.2 Initial verification
Specify -XX:+UseG1GC and verify the system.
The system requirements are satisfied.
Yes
No
The system has a tendency to trigger Full GC.
No
Yes
Extend the verification period until Full GC is triggered.
System requirements are still satisfied despite Full GC occurring.
Yes
No
End of tuning
Full GC is occurring.
Yes
No
7.16.3 Tuning to suppress the occurrence of Full GC
When numerous large objects are created
When numerous large objects are not created
Modify the application.
The Survivor area has overflowed.
Yes
No
Use the -Xmx or -XX:SurvivorRatio option to increase the size of the Survivor area (7.16.3(3)).
Use the -XX:MaxGCPauseMillis or -XX:ConcGCThreads option to increase the number of regions subject to mixed GC (7.16.3(4)).
7.16.4 Tuning to shorten worst-case response times
If you want to shorten worst-case response times
Use the -XX:MaxGCPauseMillis option to shorten the target pause time.
The worst-case response time is improved.
Yes
No
Modify the application.
7.16.5 Tuning to improve throughput
If you want to improve throughput
Use the -Xmx or -XX:MaxGCPauseMillis option to increase the size of the New area.
Throughput is improved.
Yes
No
Modify the application.
Verify performance again.
Verify performance again.

Figure 7-34 charts the flow of initial verification.

1. Enable G1 GC and verify the system. Even if the results of this verification show that the system requirements are satisfied, check whether the system has a tendency to trigger Full GC. If the system requirements are not satisfied, go to *7.16.3 Tuning to suppress the occurrence of Full GC*.
2. If Full GC does not occur during verification, this is no guarantee that Full GC will not occur in the future when the system is run for an extended period of time. For this reason, you must make sure that the overall tendency of the system is not to trigger Full GC.

Figure 7–35: How a tendency to trigger Full GC manifests

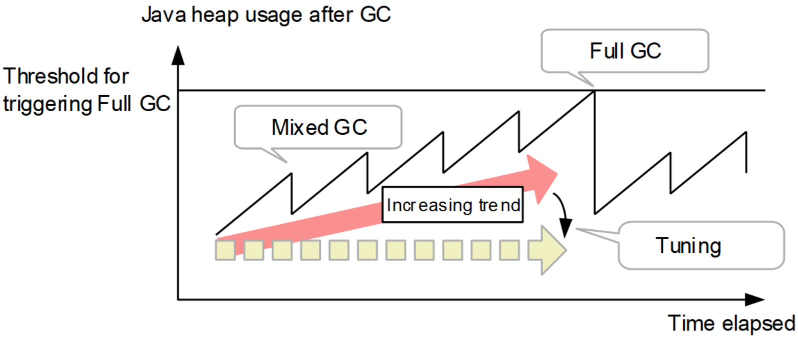


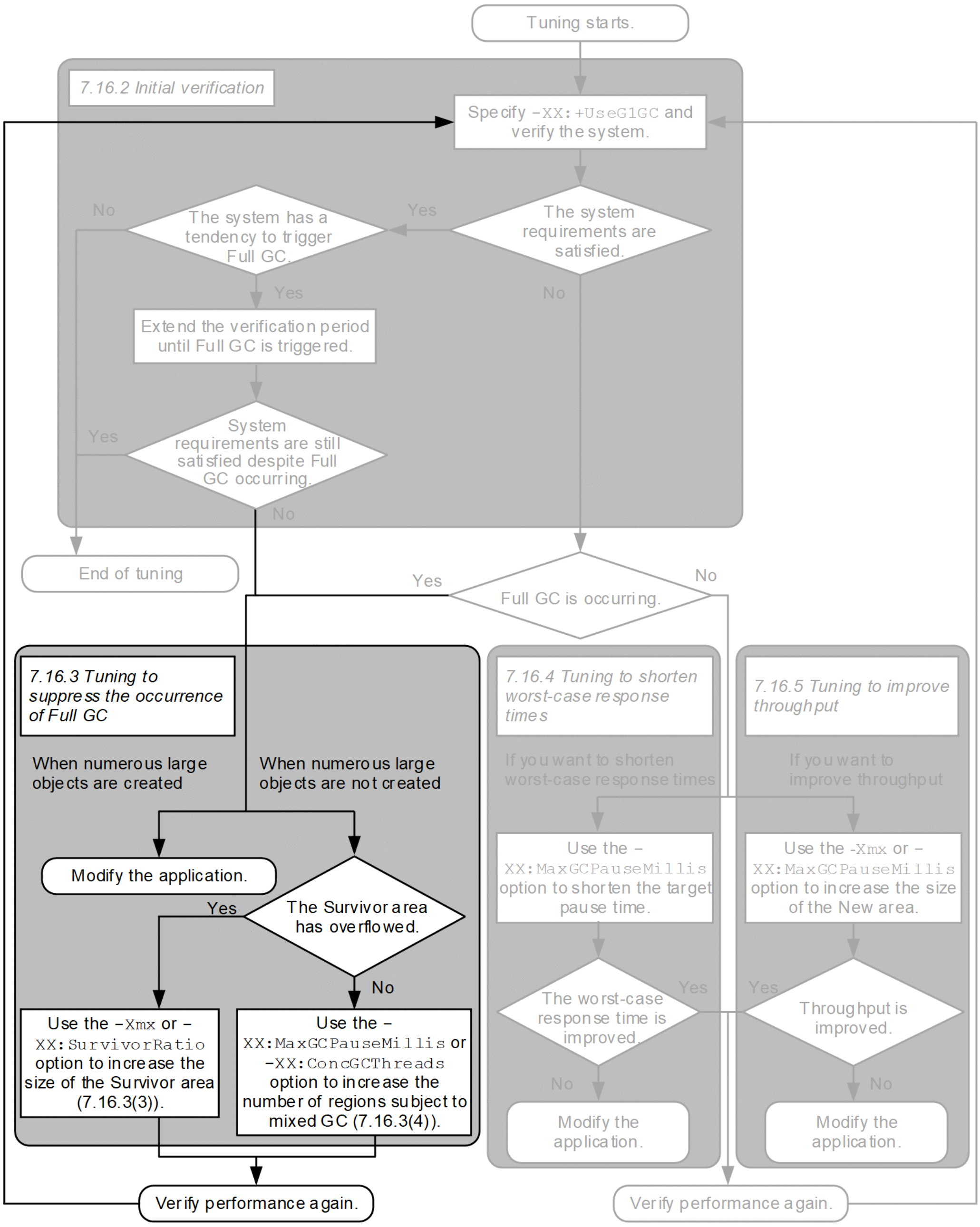
Figure 7-35 graphs the usage of the Java heap after each occurrence of GC during the verification process. A system is considered to have a tendency to trigger Full GC if, as shown in Figure 7-35, there is an overall increasing trend in Java heap usage despite mixed GC occurring.

3. If the system requirements are met but the system exhibits a tendency to trigger Full GC, extend the verification period until Full GC is triggered and then check whether the system requirements are still satisfied. If the system requirements are no longer satisfied after Full GC is triggered, perform tuning to suppress the occurrence of Full GC. If you cannot extend the verification time, you can perform tuning to eliminate the tendency to trigger Full GC even if Full GC did not occur during the verification process. For details about how to tune the Java heap to suppress Full GC, see *7.16.3 Tuning to suppress the occurrence of Full GC*.

7.16.3 Tuning to suppress the occurrence of Full GC

(1) Concept of suppressing Full GC

Figure 7–36: Flow of tuning to suppress occurrence of Full GC

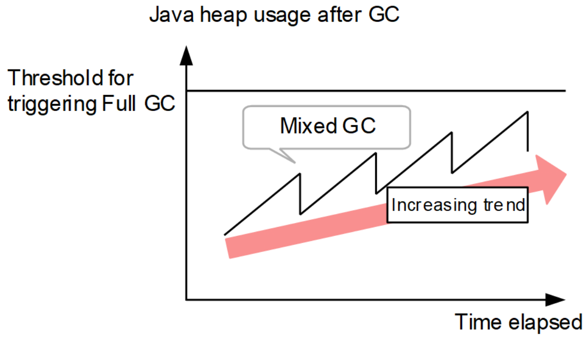


Tuning starts.
7.16.2 Initial verification
Specify –XX:+UseG1GC and verify the system.
The system requirements are satisfied.
Yes
No
The system has a tendency to trigger Full GC.
No
Yes
Extend the verification period until Full GC is triggered.
System requirements are still satisfied despite Full GC occurring.
Yes
No
End of tuning
Full GC is occurring.
Yes
No
7.16.3 Tuning to suppress the occurrence of Full GC
When numerous large objects are created
When numerous large objects are not created
Modify the application.
The Survivor area has overflowed.
Yes
No
Use the –Xmx or –XX:SurvivorRatio option to increase the size of the Survivor area (7.16.3(3)).
Use the –XX:MaxGCPauseMillis or –XX:ConcGCThreads option to increase the number of regions subject to mixed GC (7.16.3(4)).
Verify performance again.
7.16.4 Tuning to shorten worst-case response times
If you want to shorten worst-case response times
Use the –XX:MaxGCPauseMillis option to shorten the target pause time.
The worst-case response time is improved.
Yes
No
Modify the application.
7.16.5 Tuning to improve throughput
If you want to improve throughput
Use the -Xmx or –XX:MaxGCPauseMillis option to increase the size of the New area.
Throughput is improved.
Yes
No
Modify the application.
Verify performance again.

Full GC occurs when the used size of the Tenured area increases to the point where no more Free regions are available.

The following figure illustrates tuning to suppress occurrence of Full GC.

Figure 7–37: Tuning to suppress occurrence of Full GC



The key to suppressing the occurrence of Full GC is to control the increase in the used size of the Tenured area. We can therefore consider the following two approaches to suppressing the occurrence of Full GC:

1. Reduce the number of objects that are moved to the Tenured area
2. Collect objects in the Tenured area that are no longer needed

If the Survivor area is full, then use the first approach to tuning. If the Survivor area is not full, then use the second approach. For details about the first approach, see *7.16.3(3) Tuning the size of the Survivor area*. For details about the second approach, see *7.16.3(4) Tuning to increase the regions selected by mixed GC*. After finishing the tuning process, repeat system verification to confirm whether the system requirements are now satisfied.

(2) When an application creates numerous large objects

An application creating numerous large objects can make it impossible to secure enough contiguous regions and trigger Full GC. Because G1 GC manages memory at the region level, it is not suited to systems that create numerous large objects. If your system creates numerous objects and this triggers Full GC, you need to modify your applications to prevent this from happening.

You can identify large objects by looking for the `Humongous` label in the log file.

```
[VG1]<Wed Jan 15 12:51:32 2014>[Full GC 130443K/131072K(131072K)->55462K/56
320K(131072K), 2.3265610 secs][Status:-][G1GC::Eden: 0K(43008K)->0K(43008K)]
[G1GC::Survivor: 0K->0K][G1GC::Tenured: 131072K->56320K][G1GC::Humongous: 10
24K->0K][G1GC::Free: 0K->74752K][Metaspace: 3634K(4492K, 4492K)->3634K(4492K
, 4492K)][class space: 356K(388K, 388K)->356K(388K, 388K)][cause:ObjAllocFai
l][RegionSize: 1024K][Target: 0.2000000 secs][Predicted: 0.0000000 secs][Ta
rgetTenured: 0K][Reclaimable: 0K(0.00%)][User: 2.1700000 secs][Sys: 0.000000
0 secs][IM: 277185K, 261856K, 44544K][TC: 1168][DOE: 0K, 0][CCI: 5808K, 4915
2K, 5952K]
```

(3) Tuning the size of the Survivor area

The approach that reduces the number of objects that are moved to the Tenured area works by collecting objects while they are still in the Survivor area. When there is no more free space in the Survivor area, the overflow triggers a save process that promotes objects with short lifespans to the Tenured area. By increasing the size of the Survivor area, you can collect objects with short lifespans while they are still in the Survivor area, reducing the number of objects moved to the Tenured area.

When `to exhausted` appears in the log entry, this indicates that the Survivor area has overflowed. In this case, use the `-Xmx` option or the `-XX:SurvivorRatio` option to increase the size of the Survivor area.

```
[VG1]<Wed Jun 12 11:21:10 2013>[Young GC 899070K/899072K(1048576K)->501755K/
501760K(1048576K), 0.0931560 secs][Status:to exhausted][G1GC::Eden: 389120K(
389120K)->0K(397312K)][G1GC::Survivor: 41984K->41984K][G1GC::Tenured: 459776
K->459776K][G1GC::Humongous: 2048K->2048K][G1GC::Free: 609536K->607232K][Met
aspace: 3634K(4492K, 4492K)->3634K(4492K, 4492K)][class space: 356K(388K, 38
8K)->356K(388K, 388K)][cause:G1EvacuationPause][RegionSize: 1024K][Target: 0
.2000000 secs][Predicted: 0.2495800 secs][TargetTenured: 0K][Reclaimable: 0K
(0.00%)][User: 0.0156250 secs][Sys: 0.0312500 secs][IM: 729K, 928K, 0K][TC:
509][DOE: 16K, 171][CCI: 2301K, 49152K, 2304K]
```

The following explains how to use each option to tune the size of the Survivor area.

- `-Xmx` option

  You can increase the size of the Java heap area by specifying a larger value in this option. By increasing the overall size of the Java heap area, you also increase the size of Survivor area. However, increasing the maximum size of the Java heap area also increases the minimum and maximum values that govern the extent to which the New area can be resized. Because this increases the minimum pause time, after completing the tuning process, you must verify again whether the system meets the system requirements.

- `-XX:SurvivorRatio` option

  By specifying a smaller value in this option, you can increase the proportion of the reserved New area that is allocated to the Survivor area. However, because this approach results in a smaller Eden area, it has the secondary effect of making young GC and mixed GC more likely to occur. Therefore, after completing the tuning process, you must verify again whether the system meets the system requirements.

Note:

> Another approach to increasing the size of the Survivor area is using the `-XX:NewRatio`, `-XX:NewSize`, or `-XX:MaxNewSize` option to increase the size of the New area. However, because this limits the extent to which the New area can be resized after GC, we do not recommend that you take this approach.

(4) Tuning to increase the regions selected by mixed GC

To increase the regions targeted by mixed GC, you can have each instance of mixed GC target more regions, or increase the total number of target regions by triggering mixed GC more often. The first approach lowers responsiveness, and the second approach reduces throughput. Select the appropriate tuning method according to the system requirements.

- `-XX:MaxGCPauseMillis` option

  You can increase the target pause time by specifying a larger value in this option. Mixed GC targets the New area and part of the Tenured area. GC targets the Tenured area when there is sufficient leeway in the predicted pause time relative to the target pause time after targeting the New area. By increasing the target pause time, you can increase the number of Tenured regions targeted during a single instance of mixed GC. However, this approach extends the GC pause time, reducing the responsiveness of the system.

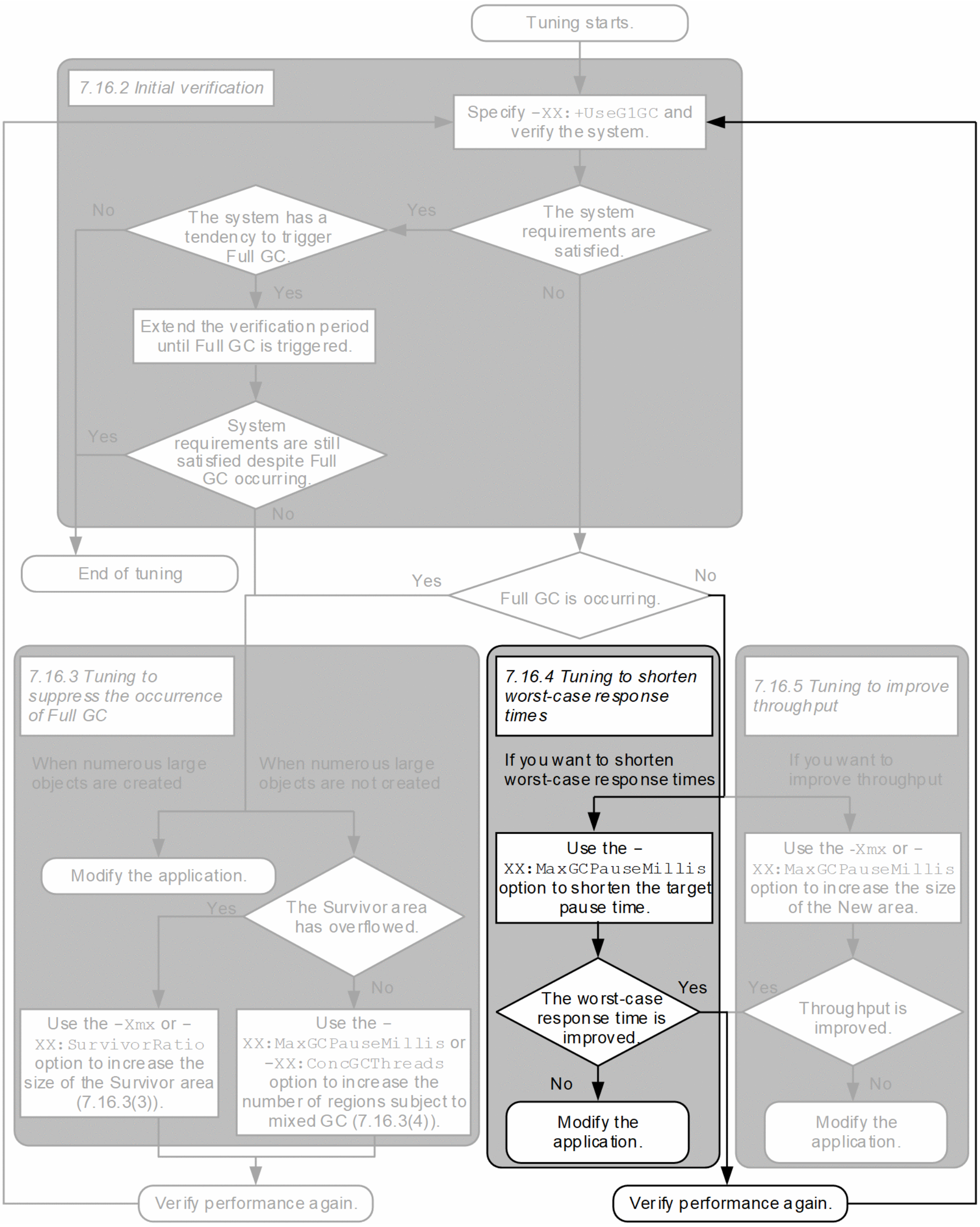
- `-XX:ConcGCThreads` option

  Increasing the value specified for this option increases the number of threads used for CM. CM must have completed before mixed GC can occur, and you can complete CM more quickly by increasing the number of threads. This increases the frequency with which mixed GC occurs, which has the effect of increasing the total number of targeted regions. However, because this method increases the number of processing threads working in parallel with the application threads, the throughput of the system is reduced.

  You cannot specify a value for the number of CM threads that is larger than the number of threads used for evacuation processing. Therefore, when you increase the value of the `-XX:ConcGCThreads` option, you must also increase the value of the `-XX:ParallelGCThreads` option.

7.16.4 Tuning to shorten worst-case response times

Figure 7–38: Flow of tuning to shorten worst-case response times



Tuning starts.
7.16.2 Initial verification
Specify -XX:+UseG1GC and verify the system.
The system requirements are satisfied.
Yes
The system has a tendency to trigger Full GC.
No
Yes
No
Extend the verification period until Full GC is triggered.
System requirements are still satisfied despite Full GC occurring.
Yes
No
End of tuning
Yes
Full GC is occurring.
No
7.16.3 Tuning to suppress the occurrence of Full GC
When numerous large objects are created
When numerous large objects are not created
Modify the application.
The Survivor area has overflowed.
Yes
No
Use the -Xmx or -XX:SurvivorRatio option to increase the size of the Survivor area (7.16.3(3)).
Use the -XX:MaxGCPauseMillis or -XX:ConcGCThreads option to increase the number of regions subject to mixed GC (7.16.3(4)).
Verify performance again.
7.16.4 Tuning to shorten worst-case response times
If you want to shorten worst-case response times
Use the -XX:MaxGCPauseMillis option to shorten the target pause time.
The worst-case response time is improved.
Yes
No
Modify the application.
7.16.5 Tuning to improve throughput
If you want to improve throughput
Use the -Xmx or -XX:MaxGCPauseMillis option to increase the size of the New area.
Throughput is improved.
Yes
No
Modify the application.
Verify performance again.

This approach to tuning shortens the worst-case response time by reducing the length of time an application pauses for GC to take place. This approach is effective when application pause time due to GC accounts for a large percentage of response time. You can identify application pause time from the `gc_time` item in `VG1` log entries. For details about `VG1` log entries, see *-XX:[+|-]HitachiVerboseGC (Option for extended verbosegc information output)* in the *uCosminexus Application Server Definition Reference Guide*.

To shorten worst-case response times, you must reduce the application pause time that is caused by GC. Because G1 GC allows you to specify the pause time, you can reduce the worst-case response time by specifying a smaller value.

- `-XX:MaxGCPauseMillis` option

  You can reduce the target pause time and shorten the worst-case response time by specifying a smaller value in this option. However, this causes GC to occur more often which reduces throughput. If you perform tuning to shorten the worst-case response time, you must repeat system verification to ensure that the system meets the system requirements. Because every type of GC executed by G1 GC targets the New area, you cannot reduce the time taken by GC to less than that taken to reclaim the New area. The pause time beyond which no further reductions are possible is called the *minimum pause time*. If reducing the target pause time does not shorten the worst-case response time, then check the GC pause time in the log data.

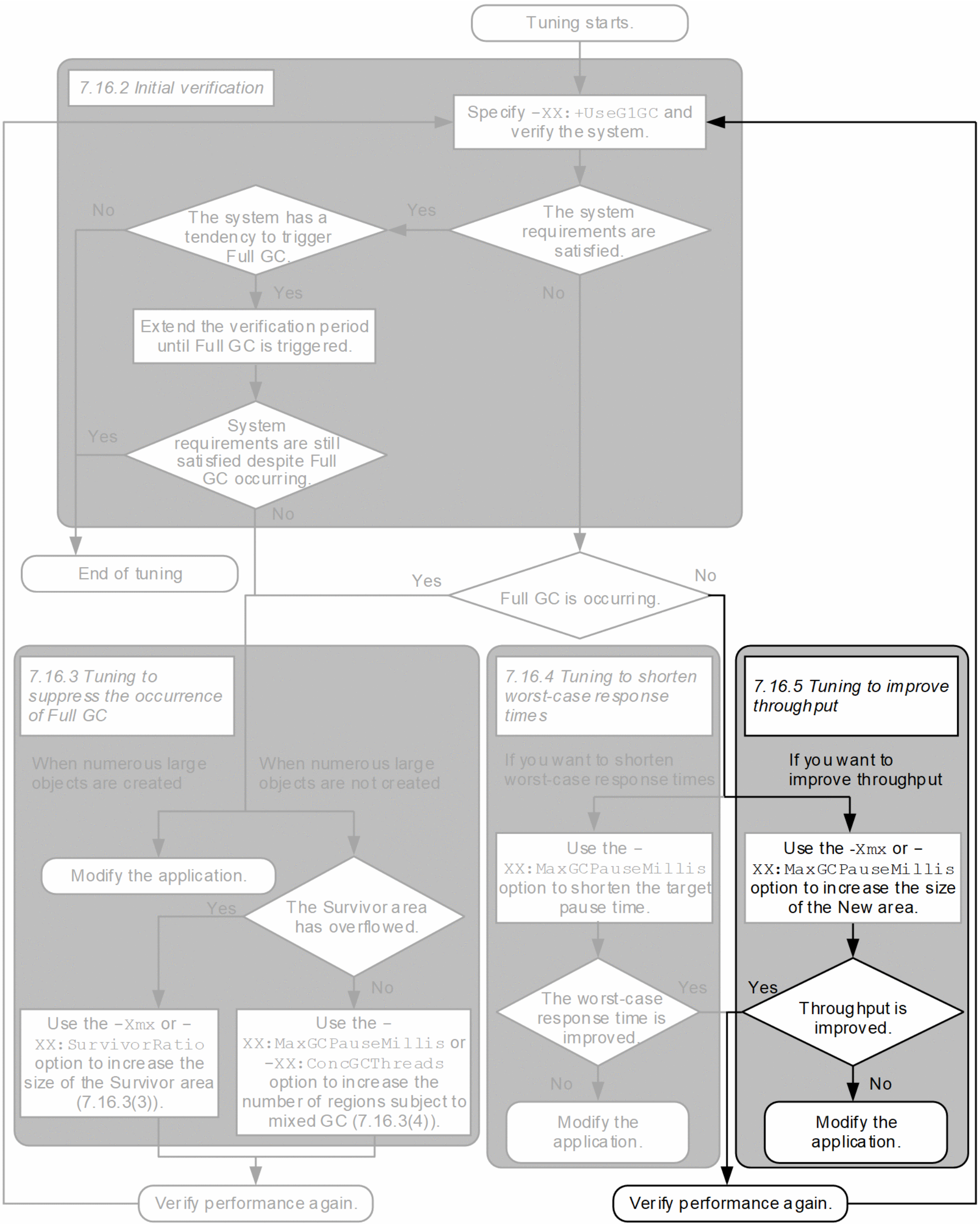
The GC pause time is the highlighted and **bolded** part in the following log entry.

```
[VG1]<Wed Jun 12 11:21:10 2013>[Young GC 899070K/899072K(1048576K)->50175
5K/501760K(1048576K), 0.0931560 secs][Status:-][G1GC::Eden: 389120K(389120
K)->0K(397312K)][G1GC::Survivor: 41984K->41984K][G1GC::Tenured: 459776K->4
59776K][G1GC::Humongous: 2048K->2048K][G1GC::Free: 609536K->607232K][Metas
pace: 3634K(4492K, 4492K)->3634K(4492K, 4492K)][class space: 356K(388K, 3
88K)->356K(388K, 388K)][cause:G1EvacuationPause][RegionSize: 1024K][Target
: 0.2000000 secs][Predicted: 0.2495800 secs][TargetTenured: 0K][Reclaimabl
e: 0K(0.00%)][User: 0.0156250 secs][Sys: 0.0312500 secs][IM: 729K, 928K, 0
K][TC: 509][DOE: 16K, 171][CCI: 2301K, 49152K, 2304K]
```

If reducing the target pause time does not shorten the worst-case response time, this means that the minimum pause time has been reached. A target time that is too close to the minimum pause time means that when the target pause time is exceeded, it is exceeded by a wider margin. This can extend the worst-case response time. If the system still does not meet the system requirements after reducing the target pause time, then tuning on its own is not enough. You will need to modify the application in order to meet the system requirements.

7.16.5 Tuning to improve throughput

Figure 7–39: Flow of tuning to improve throughput


Tuning starts.
7.16.2 Initial verification
Specify -XX:+UseG1GC and verify the system.
The system requirements are satisfied.
Yes
No
The system has a tendency to trigger Full GC.
No
Yes
Extend the verification period until Full GC is triggered.
System requirements are still satisfied despite Full GC occurring.
Yes
No
End of tuning
Full GC is occurring.
Yes
No
7.16.3 Tuning to suppress the occurrence of Full GC
When numerous large objects are created
When numerous large objects are not created
Modify the application.
The Survivor area has overflowed.
Yes
No
Use the -Xmx or -XX:SurvivorRatio option to increase the size of the Survivor area (7.16.3(3)).
Use the -XX:MaxGCPauseMillis or -XX:ConcGCThreads option to increase the number of regions subject to mixed GC (7.16.3(4)).
Verify performance again.
7.16.4 Tuning to shorten worst-case response times
If you want to shorten worst-case response times
Use the -XX:MaxGCPauseMillis option to shorten the target pause time.
The worst-case response time is improved.
Yes
No
Modify the application.
7.16.5 Tuning to improve throughput
If you want to improve throughput
Use the -Xmx or -XX:MaxGCPauseMillis option to increase the size of the New area.
Throughput is improved.
Yes
No
Modify the application.
Verify performance again.

To improve throughput, you must reduce the frequency with which GC occurs. Young GC and mixed GC are triggered when the Eden area becomes full. By increasing the size of the New area, you also increase the size of the Eden area which reduces the frequency of GC. You can perform tuning to increase the size of the New area by using the `-Xmx` option or the `-XX:GCMaxPauseMillis` option.

- `-Xmx` option

  You can increase the maximum size of the Java heap by specifying a larger value in this option. Increasing the maximum size of the Java heap area also increases the size of the New area.

- `-XX:MaxGCPauseMillis` option

  You can increase the target pause time by specifying a larger value in this option. Because young GC reclaims as much of the New area as the target pause time allows, extending the target pause time increases the size of the New area.

If you complete the tuning to improve throughput but throughput does not improve, it is possible that the size of the New area is at the maximum end of the range in which it can be resized. Check whether the New area was resized based on the changes in the values for the Eden area and the Survivor area in the log data. For details about how to perform this check, see *7.15.8 Young GC*. Note that the tuning to improve throughput can sometimes reduce the responsiveness of the application. For this reason, you must verify the performance of the system after completing the tuning process.

8 Performance Tuning (J2EE Application Execution Platform)

This chapter describes how to tune the performance of a system for executing J2EE applications.

You can maximize the performance of the system by optimizing the operating environment through performance tuning.

For determining the performance tuning of a batch application execution platform, see *9. Performance Tuning (Batch Application Execution Platform)*.

8.1 Points to be considered for performance tuning

This section explains the points to be considered for performance tuning of the J2EE application execution platform.

8.1.1 Viewpoints for performance tuning

Tune the performance of the J2EE application execution platform with the following viewpoints:

- **Optimizing the number of concurrent executions**
- **Optimizing the method of invoking the Enterprise Bean**
- **Optimizing the method of accessing the database**
- **Setting the timeout**
- **Optimizing the operation of the Web application**
- **Optimizing the operation of CTM**
- **Tuning of other items**

These points are explained below:

(1) Optimizing the number of concurrent executions

The objective of optimizing the number of concurrent executions is to enhance the throughput of the system by multi-processing in order to maximize the CPU performance. In the following cases, however, only multi-processing may not enhance throughput; sometimes the throughput may even deteriorate:

- Bottlenecks in I/O processing and lock processing
- Maximum throughput has already been reached
- Load exceeds the multiplicity when the CPU usage is already full
- Pending queue size is inappropriate
- Settings for maximum number of hierarchical executions is inappropriate

Performance tuning aims at optimizing the number of concurrent executions through proper tuning for the above points.

(2) Optimizing the method of invoking the Enterprise Bean

The objective of optimizing the method of invoking the Enterprise Bean is to restrict unnecessary network access by using the local invocation functionality of the local interface and the remote interface when invoking the components in the same J2EE application and the same J2EE server.

You can restrict the unnecessary network access caused by RMI-IIOP communication, by using the following functionality:

- Use of local interface
- Use of local invocation functionality of the remote interface

In addition, you can further enhance the processing performance by using pass by reference method of passing the argument and return value. Performance tuning aims at enhancing the processing performance by effectively using these functions depending on the features of the application and the system.

(3) Optimizing the database access method

The purpose of optimizing the database access method is to reduce the overheads during database access by generating in advance the connections and statements that are likely to need more time for processing.

Performance tuning enhances throughput by optimizing the database access by using the following functionality effectively:

- Connection pooling
- Statement pooling (pooling of PreparedStatement and CallableStatement)

(4) Setting the timeout

The purpose of setting the timeout is to detect a failure that may occur in the system and release the resources whenever required to avoid a delay in responding to the requests.

There are following types of timeout settings:

- Timeout of Web front-end system
- Timeout of back-end system
- Timeout of transaction
- Timeout of database

(5) Optimizing the Web application operations

The purpose of optimizing the Web application operations is to increase the processing speed by restricting unnecessary network access that results from the use of cache and determination of delivery method of contents, and enhance the system throughput with the help of load balancing.

(6) Optimizing the operation of CTM

The purpose of optimizing the operation of CTM is to improve the performance of the system by reducing the communication overhead through optimization of the communication interval between the processes used in CTM, and by promptly detecting and taking action when a trouble occurs. Moreover, by prioritizing the processing of requests by CTM, you can tune to execute quick processing of the important requests.

(7) Tuning other items

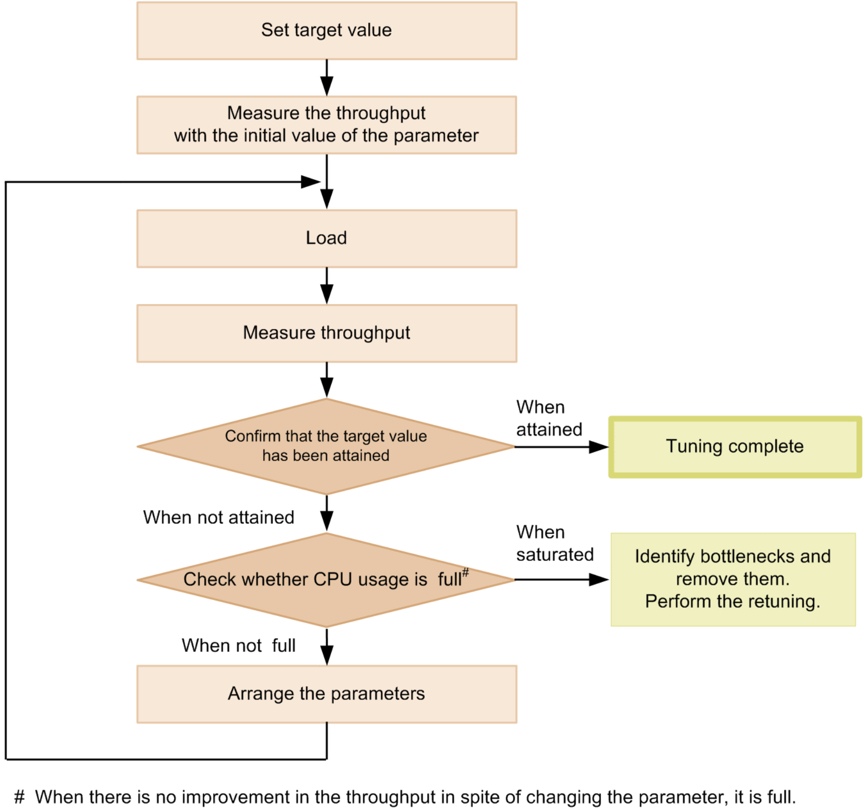
In addition to those explained above in points (1) to (6), application server has other items that can be tuned. These can be tuned whenever required.

8.1.2 Tuning procedure

Performance tuning is a task that involves detecting the best settings for system performance. In an environment that is already built, revising parameters, identifying and removing bottlenecks during actual processing and mock loading can continuously enhance the performance.

The procedure of tuning a number of concurrent executions is shown below as an example of performance tuning procedure:

Figure 8–1: Procedure of performance tuning (tuning the number of concurrent executions)


Set target value
Measure the throughput with the initial value of the parameter
Load
Measure throughput
Confirm that the target value has been attained
When attained
Tuning complete
When not attained
Check whether CPU usage is full#
When saturated
Identify bottlenecks and remove them. Perform the retuning.
When not full
Arrange the parameters
# When there is no improvement in the throughput in spite of changing the parameter, it is full.

For tuning, first of all, you need to decide the target value. In the above example, it could be any value such as the CPU usage.

Next, measure the throughput where the default value is set in each parameter, then while applying the simulated load, revise each parameter and search for the optimum value that is closest to the target value. Using a special tool for this purpose can simulate load-increase.

You can use tools such as the monitoring tool of the OS for estimating the CPU usage for the purpose of tuning. You can use tools such as the load generation tool for estimating the throughput. Moreover, you can use the statistics collection functionality to collect statistic information of Application Server such as the number of statistic threads. For details about confirmation methods, see *3. Monitoring the Statistics (Statistics Collection Functionality)* in the *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*.

The performance tuning is complete when throughput reaches a value specified as the target value. If the CPU usage is slightly less than 100%, bottlenecks are likely in the I/O and exclusion processing of the system. Identify and remove the bottlenecks and then retry performance tuning. You can use performance analysis trace to identify the bottlenecks in the Application Server system. For details about the functionality of performance analysis trace, and how to use the trace file acquired using the performance analysis trace, see *7. Performance Analysis by Using Trace Based Performance Analysis* in the *uCosminexus Application Server Maintenance and Migration Guide*.

8.1.3 Items that can be tuned for each type of application

The tuning items differ as per the type of application. The table below describes the tuning items for each component of the application.

Table 8–1: Tuning items of an application consisting of Servlet and JSP (Web application)

Tuning Items	Available functionality	Reference
Optimizing the number of request-processing threads of the Web container	Control of the number of request-processing threads in the Web container	*8.3.3*
Optimizing the number of concurrent executions	Concurrently executed thread control in the Web application (each Web container, Web application, or URL group)	*8.3.4*
Optimizing the method for accessing the database	Connection pooling	*8.5.1*
	Statement pooling	*8.5.2*
Setting the timeout	Setup of timeout in the Web front-end system	*8.6.2*
	Setup of timeout of method execution of business application	*8.6.7*
Optimizing the operation of Web application	Separation of the deployment of static contents and Web application	*8.7.1*
	Caching static contents	*8.7.2*

Table 8–2: Tuning items of application configured by Enterprise Bean

Tuning items	Available functionality	Reference
Optimizing the number of concurrent executions	Pooling of Stateless Session Bean instances	*8.3.5*
	Session control of Stateful Session Bean	
	Pooling of Message-driven Bean instances	
	Control the number of concurrent executions with CTM[#] (when using CTM)	*8.3.6*
Optimizing the method of invoking Enterprise Bean	Use of local interface	*8.4.1*
	Optimization of local invocation of remote interface	*8.4.2*
	Pass by reference for the remote interface	*8.4.3*
Optimizing the method of accessing database	Connection pooling	*8.5.1*
	Statement pooling	*8.5.2*
Setting the timeout	Setup of timeout in the back-end system	*8.6.3*
	Setup of transaction timeout	*8.6.4*
	Setup of timeout for database	*8.6.6*
	Setup of timeout for method execution of J2EE application	*8.6.7*

#
Applicable only for the Stateless Session Bean.

The following table describes the tuning items of CTM operation that can be set up in systems that use CTM. You can use CTM when Stateless Session Beans configure an application.

Table 8–3: Tuning items of CTM operation

Tuning items	Available functionality	Reference
Optimizing the operation of CTM	Tuning of the monitoring interval of the operation state of CTM domain managers and CTM daemons	*8.8.1*
	Tuning of the monitoring interval of the load status	*8.8.2*

Tuning items	Available functionality	Reference
	Setup of a timeout lock for CTM daemon	*8.8.3*
	Setup of a priority order for the requests distributed with CTM	*8.8.4*

8.2 Tuning Method

This section describes the tuning method and the way it differs according to the type of the object that will be set.

8.2.1 Tuning of J2EE server and Web server

You use the Easy Setup definition file of the Smart Composer functionality for tuning the J2EE server and Web server. In the Easy Setup definition file, specify type of the logical server (J2EE server or Web server) you want to set in *logical-server-type* under the *configuration* tag, and specify the parameter name and its value under the *param* tag. For details about the Easy Setup definition file, see *4.3 Easy Setup definition file* in the *uCosminexus Application Server Definition Reference Guide*.

If you are unable to use the Smart Composer functionality, you can define the tuning of the Web server by editing the definition file. For details, see the *uCosminexus Application Server HTTP Server User Guide*.

8.2.2 Tuning of applications or resources

Use the server management commands for tuning application and resources.

Edit the property file to use the server management command. For details about the property files, see the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

8.2.3 Tuning of the CTM operation

Use an Easy Setup definition file of the Smart Composer functionality for tuning CTM. In the Easy Setup definition file, specify the types of logical server (CTM domain manager or CTM) for setting up in `<logical-server-type>` under the *configuration* tag, and set up the parameter name and value under the *param* tag.

For details about the Easy Setup definition files, see *4.3 Easy Setup definition file* in the *uCosminexus Application Server Definition Reference Guide*.

8.2.4 Tuning other items

In addition to the above, there are other parameters that are used for tuning such as items set in the API and the items set in the database. For more information about these settings, see the sections on respective tuning parameters.

8.3 Optimizing the number of concurrent executions

This section describes the various tuning methods and the concept of optimizing the number of concurrent executions of application requests.

8.3.1 Concept of number of concurrent executions control and pending queue control

Multi-processing of several requests is an effective way of enhancing the throughput of applications in the Application Server system. In comparison to processing one request in one thread at one time, processing of requests in multiple threads is found to enhance the throughput.

If there are bottlenecks in I/O processing or exclusion processing, however, and if maximum throughput has already been achieved, multi-processing is not effective in enhancing the throughput. Check the following points as you tune the number of concurrent executions:

Removing bottlenecks from I/O processing and exclusion processing

If the CPU usage remains low even after multiplexing threads and the throughput is not enhanced, it is likely that there are bottlenecks in I/O processing and exclusion processing, for example when the application accesses a database. In such a case, tuning has to be done after identifying the process that has a bottleneck and removing the bottleneck. For example, the bottlenecks in I/O processing and exclusion processing can be removed by tuning the method of accessing the database and by changing the method of exclusion processing.

Check maximum throughput

If you go on increasing the multiplicity of requests and the number of threads, the free time available in the CPU goes on decreasing along with the increase in the number of threads and any further increase will result in almost no time available in the CPU. If this stage is reached, the throughput will not be enhanced even by increasing the number of threads.

This state indicates that there is a bottleneck in the CPU and the performance of the machine itself has reached the maximum limit. In short, the throughput at this time is the maximum throughput of the application on the machine.

To gain higher throughput, you have to enhance the hardware by increasing the number of machines and CPU.

Maintaining the throughput by controlling the number of concurrent executions

If multiplicity and the number of threads are increased in a state in which the CPU usage is full, there is an increase in the number of threads that cannot be allocated by the CPU in an executable state. This will lead to a lock conflict between the threads, and repetition of thread's context switch that can result in deterioration of the throughput.

If you increase the number of threads, the memory usage will also increase along with the number of threads in the application server. In short, to avoid increasing the memory usage and decreasing the throughput, it is necessary to limit the number of threads to the number of executable threads only.

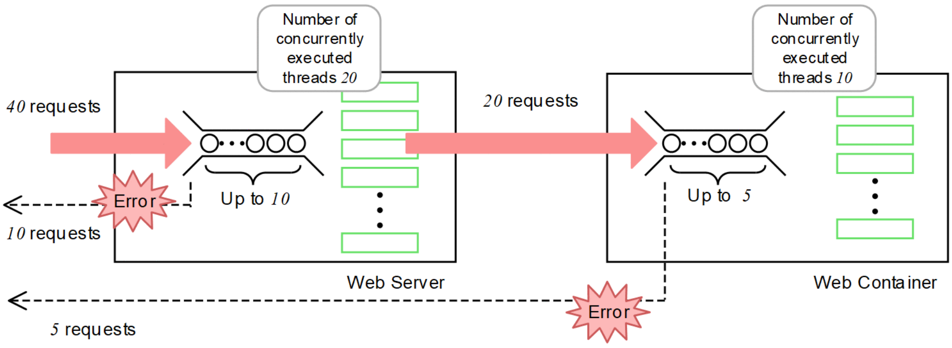
You can use the function for controlling the number of concurrent executions appropriately in order to hold the execution of requests exceeding the value of the maximum number of concurrent executions even when you have tuned the maximum number of concurrent executions and increased the multiplicity of the requests. As a result, a high throughput can be maintained despite temporary overloading and peak load status.

Tuning the pending queue size

When a request received in the application server exceeds the maximum number of concurrent executions, you can register the request in a queue and keep the request pending until the processing of all requests in progress is complete. However, if the queue in which the request is kept pending until processing of other requests is complete, has a maximum size limit (*pending queue size*), and there is a new request that exceeds the maximum size of the pending queue, the request is not registered in the queue and is returned to the client as an error. When setting a maximum value for the pending queue, it is necessary to secure enough number of requests that can be held pending in the queue.

The concepts of registering a request in the pending queue and returning the request as an error are explained below with the help of following figure:

Figure 8–2: Registering a request in a pending queue and returning a request as an error



- In the Web server, *20* threads can be concurrently executed for *40* requests and *10* requests can be registered in the pending queue.
  Error is returned if the upper limit of the pending queue is exceeded by *10* requests.
- In the Web container, *10* threads can be concurrently executed for *20* requests and *5* requests can be registered in the pending queue.
  Error is returned if the upper limit of the pending queue is exceeded by *5* requests.

Tip

Keeping requests in the pending queue is to prevent an error that may occur due to temporary overloading and peak-load state. Unnecessarily increasing the size of the pending queue for preventing failures is not an effective solution. Increase the number of concurrent executions or increase the number of machines or CPUs as necessary.

When a timeout has been specified on the client-side and if too much time has elapsed from the time the request was registered in the pending queue until it is actually executed, it is likely that the timeout will occur before the execution of the request and result in an error.

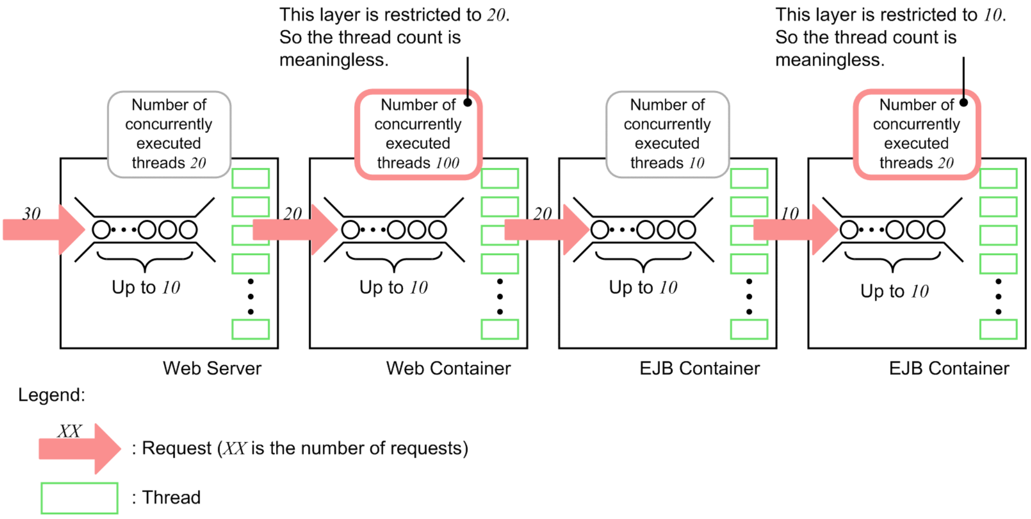
Specify an appropriate size for the pending queue.

Balancing the maximum number of concurrent executions in a hierarchical application

The performance of the entire system is not enhanced by tuning only a certain layer and increasing the number of concurrent executions because the layer with low performance is limiting the performance of the entire system.

The figure below illustrates an example in which meaningless settings are specified as a result of optimizing only a particular layer:

Figure 8–3: Example of meaningless settings due to optimization of a particular layer


This layer is restricted to 20. So the thread count is meaningless.
This layer is restricted to 10. So the thread count is meaningless.
Number of concurrently executed threads 20
Number of concurrently executed threads 100
Number of concurrently executed threads 10
Number of concurrently executed threads 20
30
20
20
10
Up to 10
Up to 10
Up to 10
Up to 10
Web Server
Web Container
EJB Container
EJB Container
Legend:
XX
: Request (XX is the number of requests)
: Thread

If you increase the number of concurrent executions, it is likely that resources such as memory will be consumed unnecessarily even in an idle state in which no request is being processed. Therefore, to control the number of concurrent executions, check the entire system and specify an appropriate number of concurrent executions for each layer.

8.3.2 Procedure for requesting the maximum number of concurrent executions and pending queue

You can tune the maximum number of concurrent executions and pending queue size according to the following procedure:

1. Increase the multiplicity of requests by using tools such as the load generation tool.

   At this point, if the CPU usage on the server-side is 80% to 90% proceed to step 2. If the CPU usage is not 80% to 90%, the performance is low and it is not possible to increase the throughput, it is likely that there are bottlenecks in the I/O processing and exclusion processing. In such case, identify the process with the bottleneck and enhance the performance.

2. Set the multiplicity in which the CPU usage on the server-side has reached 80% to 90% as the maximum number of concurrent executions in the tuning parameters.

   The throughput at this stage is the maximum throughput of a single machine. Enhance the hardware if you require a higher throughput.

3. Check whether maximum throughput can be maintained by applying additional load by using tools such as load generation tool.

   If a maximum throughput cannot be maintained, revise the tuning parameter to prevent loads that exceed the maximum throughput.

4. Estimate the number of requests when the actual system is temporarily overloaded and when the loading is at a peak state and decide the size of the pending queue.

5. In an application having a hierarchical structure, revise the tuning parameter so that the size of the pending queue and number of concurrent executions in each layer can be balanced.

8.3.3 Controlling the number of request-processing threads

In the case of Web front-end systems, the request-processing thread created by the Application server processes the requests from clients like the Web browser. The processing efficiency can be improved by appropriately controlling the number of request-processing threads.

This subsection describes the purpose of controlling the number of request-processing threads in Application servers, and the guidelines for tuning.

This subsection specifically describes how to tune the number of request-processing threads created by the Web container running on the J2EE server.

> **Reference note**
>
> When using the HTTP Server as a reverse proxy deployed at the front end of the J2EE server, you can accomplish the same tuning in the settings of the HTTP Server. For details, see the manual *uCosminexus Application Server HTTP Server User Guide*.

(1) Purpose of controlling the number of request-processing threads

The performance can be improved by tuning the number of request-processing threads according to the quality of the host on which the J2EE server is running, and according to the status of access from the client.

The process of generating request-processing threads puts a heavy load on the system. By generating and pooling the request-processing threads beforehand, you can reduce the load for a processing request from the client, such as a Web browser and can increase the processing performance.

In the Web container, you can generate and pool the request-processing threads as a batch at startup of the J2EE server, and use this pool to fulfil processing requests from a client such as a Web browser. This leads to an improvement in the processing performance when a processing request is received. By monitoring the number of pooled threads, you can generate additional threads when the number of pooled threads becomes less, and can secure these threads in the pool.

However, if you pool a large number of unused threads, it will lead to unnecessary consumption of resources. Therefore, depending on the processing contents of the system, you must appropriately control the number of request-processing threads to be pooled, and delete the unnecessary threads, if required.

When controlling the request-processing threads, set up appropriate values in the parameters by considering the above.

(2) Guidelines for setup

When controlling the number of request-processing threads, you can tune using the following parameters:

- Number of request-processing threads generated when the J2EE server is started
- Maximum number of connections to the Web client (number of request-processing threads)
- Maximum value of the Listen queue (backlog) of TCP/IP, when the maximum number of connections is exceeded
- Maximum number of request-processing threads

Make a note of the following points when setting up these parameters:

- Depending on the contents of the service to be provided, a large number of requests must be processed immediately after the J2EE server is started. In such a case, set up a large value in the number of the request-processing threads to be generated when the J2EE server is started.

- If you increase the maximum number of request-processing threads, you can immediately accommodate sudden increases in client access without a decline in processing performance. However, if you pool a large number of threads, many resources will be consumed. Therefore, be careful when setting the appropriate number of pooled threads, after estimating the presumed sudden increase in the number of accesses.
- If you want to keep pooling threads rather than deleting them, set the maximum number of request-processing threads to the same value as the minimum number of request-processing threads.

Apart from the above, consider the relationship with the number of concurrently executed threads of the Web application. For details about the number of concurrently executed threads of a Web application, see *8.3.4 Controlling the number of concurrent executions of a Web application*.

8.3.4 Controlling the number of concurrent executions of a Web application

In the case of a Web front-end system, the control of concurrently executed threads of a Web Application controls the number of threads in which the Web server will concurrently process the HTTP requests received from clients like the Web browser.

(1) Difference in the control of number of concurrently executed threads

The difference in the control of the number of concurrently executed threads in the case of each Web container, Web application and URL group is explained below:

Web container

You can set the number of threads for simultaneous processing of HTTP requests in the entire Web container.

Web application

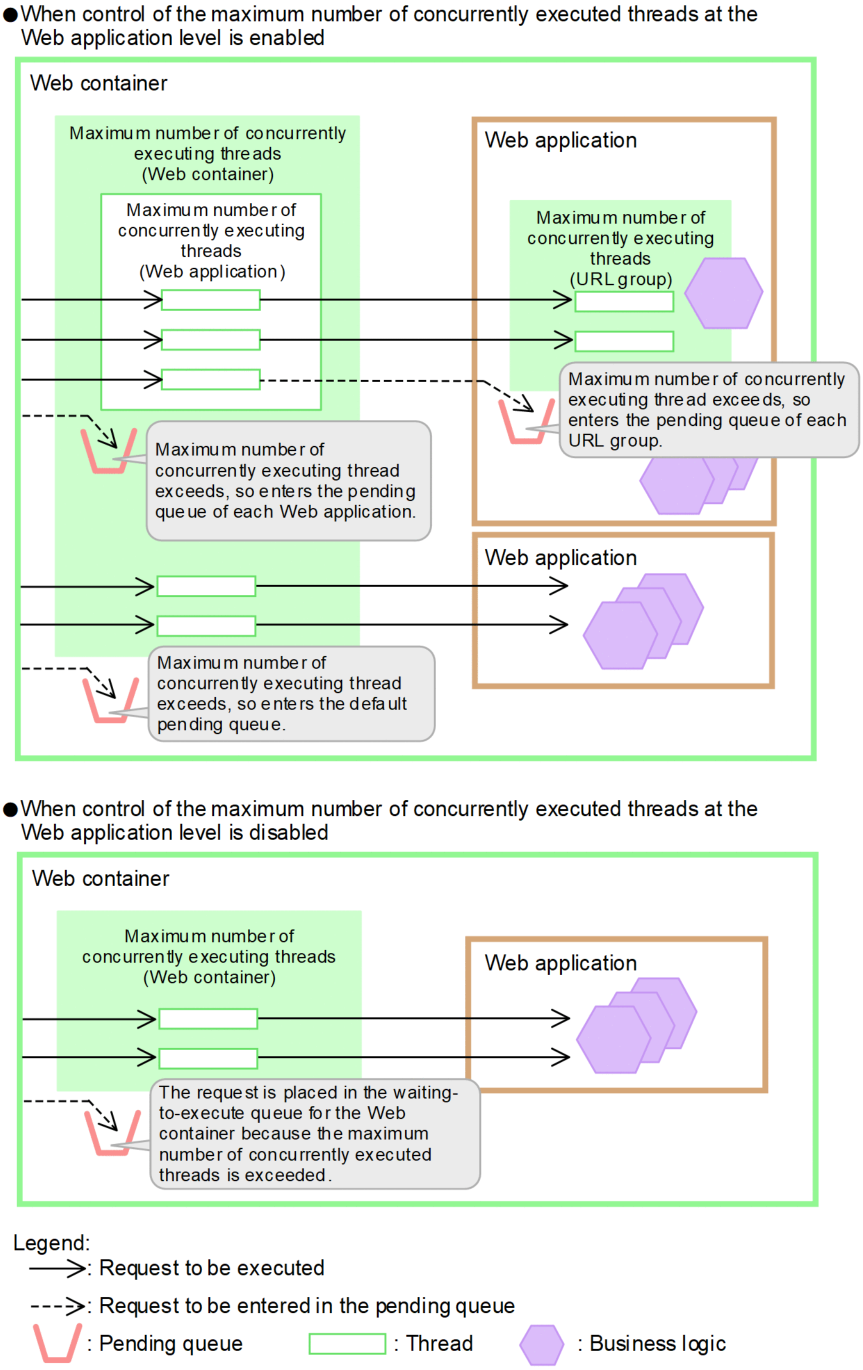
You can set the number of threads for simultaneous processing of HTTP requests for each Web application running in the Web container.

URL group

You can set the number of threads for simultaneous processing of HTTP requests for each URL when assigning HTTP requests to specific URLs of a Web application, each of which corresponds to specific business processing (business logic).

The figure below illustrates the relationship between concurrently executed threads of each Web container, Web application, and URL group.

Figure 8–4: Relationship between concurrently executed threads of each Web container, Web application, and URL group


●When control of the maximum number of concurrently executed threads at the Web application level is enabled
Web container
Maximum number of concurrently executing threads (Web container)
Maximum number of concurrently executing threads (Web application)
Web application
Maximum number of concurrently executing threads (URL group)
Maximum number of concurrently executing thread exceeds, so enters the pending queue of each URL group.
Maximum number of concurrently executing thread exceeds, so enters the pending queue of each Web application.
Web application
Maximum number of concurrently executing thread exceeds, so enters the default pending queue.
●When control of the maximum number of concurrently executed threads at the Web application level is disabled
Web container
Maximum number of concurrently executing threads (Web container)
Web application
The request is placed in the waiting-to-execute queue for the Web container because the maximum number of concurrently executed threads is exceeded.
Legend:
: Request to be executed
: Request to be entered in the pending queue
: Pending queue
: Thread
: Business logic

Important note

Control of the maximum number of concurrently executed threads at the Web container level only takes effect when control of the number of concurrently executed threads at the Web application level is disabled.

When the number of concurrently executed threads is controlled at the Web application level, the maximum number of concurrently executed threads at the Web container level is checked by the function that controls this number at the Web application level. For details, see *2.15.6 Example of setting the number of concurrently executing threads and the size of a pending queue (Web application)* in the *uCosminexus Application Server Web Container Functionality Guide*.

The execution of requests for a Web application is controlled by the number of concurrently executed threads set in each Web container, Web application, and URL group. A request that exceeds the number of concurrently executed threads that are set in each Web container, Web application, and URL group enters the respective pending queue.

(2) Guidelines for selection

The guidelines for selecting the unit of control for concurrently executed threads are explained below:

For details about the functionality for controlling the number of concurrently executed threads, see *2.13 Overview of control over the number of concurrently executed threads* in the *uCosminexus Application Server Web Container Functionality Guide*.

- **Guidelines for selecting a Web application**

  The J2EE server can manage not only the TCP connection requests but also the pending queue of the Web applications by controlling the number of concurrently executed threads of the Web application. Therefore, Hitachi recommends setting a number of concurrently executed threads of a Web application even when a single Web application is running on the J2EE server.

  Setting the number of concurrently executed threads in the Web application has the following advantages as compared to the setting of the number concurrently executed threads in each Web container:

  - Setting the maximum number of concurrently executed threads for each Web application will prevent the Web application with a large number of requests, corresponding to a particular business, from using the processing efficiency of the entire Web container. As a result, the other businesses can also be executed without any delay.
  - If there are several Web applications with different loads required for CPU and I/O processing, you can set the number of concurrently executed threads according to the conditions of respective Web applications.
  - As the size of the pending queue for requests can be set for each Web application separately, it is possible to control the pending queue according to the features of the Web application. In addition, if the requests exceeding the pending queue size of each Web application are sent, it can be communicated to the client using the HTTP response code.

  The number of concurrently executed threads of each Web application can be dynamically changed in the running J2EE server. For details about the procedure for dynamically changing the number of concurrently executed threads of Web applications executed on running J2EE servers, see *2.17.2 Flow of dynamically changing the number of concurrently executed threads* in the *uCosminexus Application Server Web Container Functionality Guide*.

- Guidelines for selecting a URL group

  If concurrently executed threads are controlled in a Web application, control the concurrently executed threads in a URL group if you want to further control the concurrently executed threads in business logic.

  Determine the settings of a URL group to include the following business logic in the Web application:

  - Business logic that you want to prioritize and execute without being influenced by other processes
  - Business logic where the CPU and I/O load is high or the required processing time is more as compared to other processes

  Setting the concurrently executed threads in a URL group has the following advantages as compared to setting only in a Web application.

  - The threads that need to be executed are allocated to the business logic (URL group) with a high priority. Even when the requests for another business logic increase, you can execute the business logic with a high priority instead of assigning the concurrently executed threads of the entire Web application to the business logic with more requests.
  - Setting the upper limit for concurrently executed threads of business logic (URL group) that require more processing time makes it possible to prevent a particular business logic from using up the concurrently executed threads of the entire Web application.

- If a Web application has several business logics (URL groups) with CPU and I/O having different loads, you can set the number of concurrently executed threads according to the business logic.
- Setting the queue size (pending queue size) for the request of each business logic (URL group) in the Web application allows the controlling of the pending queue according to the features of the business logic. If the pending queue of each URL group exceeds the upper limit, the same is notified to the client by using the HTTP response code 503 (Service Temporarily Unavailable).

8.3.5 Controlling the number of concurrently executed threads in a Enterprise Bean

The methods of controlling the number of concurrently executed threads in an Enterprise Bean are explained below for each type of Enterprise Bean. The functions of an EJB container are used to implement the concurrently executed thread control of Enterprise Bean by instance pooling and session control. For details about the functions of EJB container, see *2. EJB Container* in the *uCosminexus Application Server EJB Container Functionality Guide*.

(1) Types of functions that can be used for concurrently executed thread control of Enterprise Bean

The following two types of functions of the EJB container are used for controlling the concurrently executed threads of Enterprise Bean:

Instance Pooling

Instance pooling is a function that creates an instance of Enterprise Bean in advance and immediately processes the requests sent by the client. You can queue the execution of requests that exceed the upper limit by setting the maximum number of instances to be pooled. This helps in controlling the number of concurrently executed threads.

> **Reference note**
>
> When dynamically changing the number of concurrent executions of CTM during operation, you must set up no upper limit to the upper-limit value.

Session control

Session control is a function for controlling the number of sessions (instances) that is concurrently generated in the session.

The functions that can be used differ based on the type of Enterprise Bean.

The following table describes the functions of concurrently executed thread control that can be used based on the type of Enterprise Bean:

Table 8–4: Functions of concurrently executed thread control that can be used based on the type of Enterprise Bean

Type of Enterprise Bean	Control function that can be used
Stateless Session Bean#	Instance pooling
Stateful Session Bean	Session control
Entity Bean	Instance pooling and session control
Message-driven Bean	Instance pooling

#

When controlling the number of concurrent executions of a Stateless Session Bean, Hitachi recommends that you use CTM. For details about how to control the number of concurrent executions using CTM, see *8.3.6 Controlling the number of concurrent executions using CTM*.

Among the concurrently executed thread controls of Enterprise Bean, Message-driven Bean is the only one that manages the requests using the concept of queue for controlling the pending requests. In the case of Message-driven Bean, pending requests are controlled by using a JMS queue. In other Enterprise Beans, when a request sent from the client exceeds the maximum number of concurrent executions one of the following processes is performed, according to the setup:

- Request is kept in a queue until there is a free instance
- Immediately returned as an exception to the client
- Return the request as an exception to the client after timeout that is set for acquiring instance (in the case of method-ready pool or Entity Bean pool of the Stateless Session Bean)

(2) Controlling the number of concurrent executions in a Stateless Session Bean

Instance pooling can be used in the Stateless Session Bean. Set the normal simultaneous access count as the minimum value of instance pooling, and specify the value exceeding the maximum number of the expected simultaneous access count for the maximum value of instance pooling. This will save the time required for generating an instance during normal access and enhance the processing efficiency. Further, even when the access count increases, the access requests can be processed until it reaches the expected number of maximum access count and the execution of any request that exceeds the maximum access count is held in the pending queue.

By default, a request that exceeds the upper limit is not returned as an error. It remains in the queue until the pooled instance becomes empty. To return the request as an error, set the timeout for instance collection queue if required.

(3) Controlling the number of concurrent executions in a Stateful Session Bean

As the Stateful Session Bean has a state for each session for each client, the number of concurrent executions cannot be controlled accurately. Stateful Session Bean enables flow control (*session control*) in the session.

Specify the maximum expected concurrently executed sessions as the maximum value of session control. When the number of access requests increase and there is an access request that exceeds the maximum expected concurrently executed sessions, the session is not established and it is returned as an exception (`java.rmi.RemoteException`) to the client.

Tip

When it is necessary to control the number of concurrent executions of each request, it is possible to control the number of concurrent executions of Web application or control the number of concurrent executions using instance pooling, by invoking Stateful Session Bean through servlet, JSP or Stateless Session Bean.

(4) Controlling the number of concurrent executions in an Entity Bean

In the case of an Entity Bean, the number of concurrent executions can be controlled through the setup of maximum sessions that can be controlled for each client, and instance pooling. When maximum number of sessions is reached, request for a new session cannot be executed even if there is a vacancy in the instance pool.

When the maximum number of sessions is exceeded, an exception (`java.rmi.RemoteException`) is notified to the client immediately when there is a failure to generate the session. In the case of default settings, a request that exceeds the maximum number of instance pools is not returned as an error but kept pending in queue until there is a vacancy in the pooled instances. To return the request as an error, specify a timeout for instance collection queue as required.

The database is accessed in the execution of request for the Entity Bean. The number of connections for accessing the database also controls the number of requests that can be executed concurrently.

(5) Controlling the number of concurrent executions in a Message-driven Bean

You can use instance pooling in a Message-driven Bean. Specify the value of maximum messages as the maximum value of instance pooling. This helps to save the time required to generate an instance when a message arrives and enhance the processing performance. Even when the number of messages increases, keeping the messages pending in a queue can control the execution of the messages exceeding the maximum value for instance pooling.

In the case of Message-driven Bean, the incoming messages can be managed in the pending queue by using a JMS queue.

Reference note

When invoking Message-driven Bean from SUP of OpenTP1 using the TP1 inbound integrated function or when invoking Message-driven Bean using Cosminexus JMS provider, you must examine the number of concurrent executions in view of the component used along with the contents described in this section. For details, see *4. Invoking Application Server from OpenTP1 (TP1 inbound integrated function)* or *6. Cosminexus JMS Provider* in the *uCosminexus Application Server Common Container Functionality Guide*.

8.3.6 Controlling the number of concurrent executions using CTM

When you are using CTM, you can control the number of concurrent executions of a Stateless Session Bean.

CTM is a group of processes independent of the J2EE server. The number of concurrent executions is controlled when the invocation of the Stateless Session Bean between the EJB client and J2EE server is relayed, and the Stateless Session Bean is invoked. When using CTM, the number of concurrent executions will be controlled in each J2EE application.

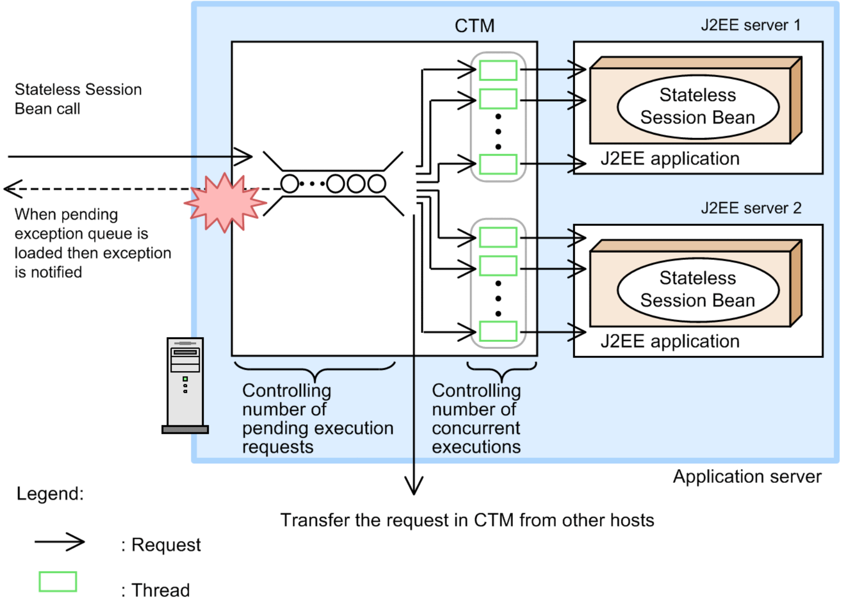
(1) Controlling the number of concurrent executions of a Stateless Session Bean using CTM

By controlling the number of concurrent executions (flow control) with CTM, you can perform the following tuning:

- If multiple J2EE applications with different loads for CPU and I/O processing are running on a J2EE server, you can set up the appropriate number of concurrent executions for each condition.
- By managing pending queues (schedule queues) with CTM, you can maintain the number of pending requests below a fixed number, and you can report an exception to the client when requests are sent in excess of this fixed number.
- If the load on a particular J2EE server is high, you can distribute the requests to other J2EE servers.

The following figure shows an example of controlling the number of concurrent executions of a Stateless Session Bean with CTM:

Figure 8–5: Example of controlling the number of concurrent executions of a Stateless Session Bean with CTM


CTM
J2EE server 1
Stateless Session Bean call
Stateless Session Bean
J2EE application
When pending exception queue is loaded then exception is notified
J2EE server 2
Stateless Session Bean
J2EE application
Controlling number of pending execution requests
Controlling number of concurrent executions
Application server
Legend:
Transfer the request in CTM from other hosts
: Request
: Thread

Tip

CTM can control the number of concurrently executed threads on a host by controlling the invocation of Stateless Session Beans on J2EE servers running on the same host. Although this varies with the machine specifications of the application server machine, Hitachi recommends a configuration in which one CTM daemon and two to four J2EE servers are started for each machine.

Note that the number of concurrently executed threads can also be changed dynamically with CTM in a running CTM daemon.

For details about the CTM functionality to control the number of concurrent executions, and the procedure to dynamically change the number of concurrently executed threads with CTM in a running CTM daemon, see *3.4 Flow-volume control of requests* in the *uCosminexus Application Server Expansion Guide*.

(2) Guidelines for proper usage with instance pooling of the EJB container

Hitachi recommends the control of the number of concurrent executions using CTM. Note that when you use CTM to control the number of concurrent executions, you can use the instance pooling functionality of the EJB container and controlling the number of concurrent executions at the same time.

The advantages of using CTM for controlling the number of concurrent executions, in addition to the use of the functionality of EJB container for controlling the number of concurrent executions are as follows:

- When the number of concurrent executions reaches the upper limit in an EJB container, the requests can be distributed to another J2EE server.
- If loading on a particular J2EE server is high even when the number of concurrent executions does not reach the upper limit, the requests can be distributed to another J2EE server.
- By managing pending queues (schedule queues) with CTM, you can maintain the number of pending requests below a fixed number, and you can report an exception to the client when requests are received in excess of this fixed number.

Tip

When you use CTM for controlling the number of concurrent executions along with the instance pooling functionality of the EJB container, you must set up the instance pooling count of Stateless Session Bean to a value higher than that of the number of concurrent executions of CTM.

For dynamically changing the number of concurrent executions of CTM during operation, you must set up no upper limit for the instance pooling count of the Stateless Session Bean. By default, no upper limit is setup. Do not change the default settings.

8.3.7 Tuning parameter for optimizing the number of concurrent executions

This section explains the method of setting up the tuning parameters used for optimizing the number of concurrent executions.

(1) Number of request-processing threads

This subsection explains how to set the tuning parameters that govern the number of request-processing threads of the Web container.

Set up the items listed in the following table in the Easy Setup definition file. Afterward re-build the system using the Smart Composer functionality.

Table 8–5: Tuning parameters for the number of request-processing threads of the Web container

Setup item	Setup target	Location of setup (parameter name)
Number of request-processing threads generated when the J2EE server is started	Logical J2EE server (`j2ee-server`)	`webserver.connector.nio_http.min_threads`
Maximum number of connections to the Web client or reverse proxy	Logical J2EE server (`j2ee-server`)	`webserver.connector.nio_http.max_connections`
Maximum value of TCP/IP Listen queue (backlog) when the maximum number of connections to the Web client or reverse proxy is exceeded	Logical J2EE server (`j2ee-server`)	`webserver.connector.nio_http.backlog`
Maximum number of request-processing threads	Logical J2EE server (`j2ee-server`)	`webserver.connector.nio_http.max_threads`

For details about the tuning parameters when using HTTP Server as a reverse proxy at the front end of the J2EE server, see the *uCosminexus Application Server HTTP Server User Guide*.

(2) Number of concurrent executions of a Web application

Set in each URL group, Web application or Web container.

(a) Number of concurrent executions of a URL group

This subsection explains how to set up the tuning parameters for number of concurrent executions of a URL group. The method and location of the setup is different for each setup item.

You specify the items listed in the following table using the Smart Composer functionality. You define the parameters in the Easy Setup definition file.

Table 8–6: Tuning parameters of the number of concurrent executions of a URL group (items to be specified using the Smart Composer functionality)

Setup item	Setup target	Location of setup (parameter name)
Maximum number of concurrently executed threads in each Web container	Logical J2EE server (`j2ee-server`)	`webserver.connector.nio_http.max_servlet_execute_threads`
Default pending queue size	Logical J2EE server (`j2ee-server`)	`webserver.container.thread_control.queue_size`

You specify the items listed in the following table using the server management command (`cjsetappprop`). You define parameters in the WAR property file.

Table 8–7: Tuning parameters of the number of concurrent executions of a URL group (items to be specified using the server management command (`cjsetappprop`))

Setup item	Location of setup (parameter name)
Maximum number of concurrently executed threads in each Web application	`<thread-control-max-threads>` under `<thread-control>` tag
Number of dedicated threads of the Web application	`<thread-control-exclusive-threads>` under `<thread-control>` tag
Pending queue size of the Web application	`<thread-control-queue-size>` under `<thread-control>` tag
Definition the name of concurrently executed thread control for a URL group	`<urlgroup-thread-control-name>` under `<thread-control>` `<urlgroup-thread-control>` tag
Maximum number of concurrently executed threads in a URL group	`<urlgroup-thread-control-max-threads>` under `<thread-control>` `<urlgroup-thread-control>` tag
Number of dedicated threads in a URL group	`<urlgroup-thread-control-exclusive-threads>` under `<thread-control>` `<urlgroup-thread-control>` tag
Pending queue size of a URL group	`<urlgroup-thread-control-queue-size>` under `<thread-control>` `<urlgroup-thread-control>` tag
URL pattern to be controlled for each URL group	`<urlgroup-thread-control-mapping>` under `<thread-control>` `<urlgroup-thread-control>` tag

(b) Number of concurrent executions in a Web application

This subsection explains how to set up the tuning parameters for the number of concurrent executions in each Web application. The method and location of setup is different for each setup item.

You specify the items listed in the following table using the Smart Composer functionality. You define the parameters in the Easy Setup definition file.

Table 8–8: Tuning parameters of the number of concurrent executions in a Web application (items to be specified using the Smart Composer functionality)

Setup item	Setup Target	Location of setup (parameter name)
Maximum number of concurrently executed threads in each Web container	Logical J2EE server (`j2ee-server`)	`webserver.connector.nio_http.max_servlet_execute_threads`
Default pending queue size	Logical J2EE server (`j2ee-server`)	`webserver.container.thread_control.queue_size`

You specify the items listed in the following table using the server management command (`cjsetappprop`). You define the parameters in the WAR property file.

Table 8–9: Tuning parameters of the number of concurrent executions in the Web application (items to be specified using the server management command (`cjsetappprop`))

Setup item	Location of setup (parameter name)
Maximum number of concurrently executed threads in each Web application	`<thread-control-max-threads>`
Number of dedicated threads of a Web application	`<thread-control-exclusive-threads>`
Pending queue size for each Web application	`<thread-control-queue-size>`

(c) Number of concurrent executions in a Web container

This subsection explains how to set up the tuning parameters for the number of concurrent executions in each Web container.

You specify the items listed in the following table using the Smart Composer functionality. You define the parameters in the Easy Setup definition file.

Table 8–10: Tuning parameter of the number of concurrent executions in a Web container

Setup item	Setup target	Location of setup (parameter name)
Maximum number of concurrently executed threads in each Web container	Logical J2EE server (j2ee-server)	`webserver.connector.nio_http.max_servlet_execute_threads`

(3) Number of concurrent executions in an Enterprise Bean

The number of concurrent executions of an Enterprise Bean is set for each Enterprise Bean. The settings are explained below for each type of Enterprise Bean:

(a) Number of concurrent executions of a Stateless Session Bean

This subsection explains how to set up the tuning parameters for the number of concurrent executions of a Stateless Session Bean.

You specify the items listed in the following table using the server management command (`cjsetappprop`). You define the parameters in the Session Bean property file.

Table 8–11: Tuning parameters of the number of concurrent executions of the Stateless Session Bean

Setup item	Location of setup (parameter name)
Maximum number of instances managed in the pool	`<maximum>` under `<stateless> <pooled-instance>` tag#
Minimum number of instances managed in the pool	`<minimum>` under `<stateless> <pooled-instance>` tag

#

When dynamically changing the number of concurrent executions of CTM during operation, you must set up no limit (set up 0) to the maximum value.

(b) Number of concurrent executions of a Stateful Session Bean

This subsection explains how to set up the tuning parameters for the number of concurrent executions of a Stateful Session Bean.

You specify the items listed in the following table using the server management command (`cjsetappprop`). You define the parameters in the Session Bean property file.

Table 8–12: Tuning parameters of the number of concurrent executions of a Stateful Session Bean

Setup item	Location of setup (parameter name)
Maximum number of sessions that can be generated from the client	`<maximum-active-sessions>` under `<stateful>` tag
Time (minutes) until the unused instances are deleted	`<removal-timeout>` under `<stateful>` tag

(c) Number of concurrent executions of an Entity Bean

This subsection describes how to set up the tuning parameters for the number of concurrent executions of an Entity Bean.

You specify the items listed in the following table using the server management command (`cjsetappprop`). You define the parameters in the Entity Bean property file.

Table 8–13: Tuning parameters of the number of concurrent executions of an Entity Bean

Setup item	Location of setup (parameter name)
Maximum value of the Entity Bean that can be generated from the client	`<maximum-instances>`
Time (minutes) until the EJBObject of the unused Entity Bean is deleted	`<entity-timeout>`

(d) Number of concurrent executions of a Message-driven Bean

This subsection describes how to set up the tuning parameters for the number of concurrent executions of a Message-driven Bean.

You specify the items listed in the following table using the server management command (`cjsetappprop`), and define the parameters in the Message-driven Bean property file.

Table 8–14: Tuning parameters of the number of concurrent executions of a Message-driven Bean

Setup item	Location of setup (parameter name)
Maximum number of instances managed in the pool	`<pooled-instance> <maximum>`
Minimum number of instances managed in the pool	`<pooled-instance> <minimum>`

(4) Number of concurrent executions controlled with CTM

This subsection describes how to set up the tuning parameters for the number of concurrent executions controlled with CTM. The items are set up in CTM daemon, application, and Stateless Session Bean.

- Items to be set up in CTM daemon

  Set up the items listed in the following table in the Easy Setup definition file. Afterward re-build the system using the Smart Composer functionality.

Table 8–15: Tuning parameter for the number of concurrent executions controlled with CTM (item to be specified using the Smart Composer functionality)

Setup item	Setup target	Location of setup (parameter name)
Maximum value of threads controlled with CTM, and number of requests registered in each queue	Logical CTM (`component-transaction-monitor`)	`ctm.DispatchParallelCount`

- Items to be set up in the application or Stateless Session Bean

  Set up the items listed in the following table using the server management commands. Define the parameters in the application property file or Session Bean property file.

Table 8–16: Tuning parameters for the number of concurrent executions controlled with CTM (items to be specified using the server management commands)

Setup item	Definition file	Setup target	Location of setup (parameter name)
Whether to control the number of concurrent executions with CTM in the application	Application property file	Application	`<managed-by-ctm>`
Number of concurrently executed threads of the application	Application property file	Application	*parallel-count* below the `<scheduling>` tag
Whether to control the number of concurrent executions with CTM in the Stateless Session Bean	Session Bean property file	Stateless Session Bean	`<enable-scheduling>`
Settings for the queues of each Bean#	Session Bean property file	Stateless Session Bean	*parallel-count* below the `<scheduling>` tag

#

This item is required when you deploy a schedule queue in each Bean. Set up the deployment of the schedule queue in the `<scheduling-unit>` tag of the application property file.

8.4 Optimizing the method of invoking the Enterprise Bean

This chapter describes how to optimize the method of invoking the Enterprise Bean.

Normally, the Enterprise Bean is invoked through an RMI-IIOP by using the remote interface. In this method, the overheads are similar to those in the case of remote connection, even when invoked from the Enterprise Bean running in the same J2EE application and the same J2EE server.

The following methods can be used in optimizing the method of invoking the Enterprise Bean and enhancing the throughput:

- Invocation by using local interface
- Local invocation of remote interface
- Pass by reference of the remote interface

Each of these methods has the following features as listed in the table below. In this table, the methods are compared on the basis of four features that are as follows:

1. Standard specifications: Does it conform to standard specifications?
2. Performance: Does it enhance the performance?
3. Location transparency: Is there location transparency?
4. Maintainability: Is it easily maintainable?

Determine the method to be used based on the features of the application and system.

Table 8–17: Features of the methods of invoking Enterprise Bean

Type of method used for invocation	Standard specifications	Performance	Location transparency	Maintainability
Local interface	F	F	N	F
Local invocation of remote interface	F	P	F	F
Pass by reference for remote interface	N	F	P	N

Legend:
F: Fully supported. Excellent.
P: Partially supported. Not good
N: Not supported. Poor

For more details about local interface, see the EJB specifications.

For details about the local invocation of the remote interface and the pass by reference of the remote interface, see *2.13 Invoking EJB remote interface* in the *uCosminexus Application Server EJB Container Functionality Guide*.

8.4.1 Using the local interface

This method uses the local interface that conforms to the J2EE standard specifications, during the development of the application.

Local interface is a functionality that as usual invokes the Enterprise Bean as a method invocation of the same thread. As a J2EE standard functionality, the local interface has portability. However, if you use local interfaces, location independence is lost, since using local interfaces requires you to create programs that use dedicated local interface for local invocation, at the server side as well as the client side.

The local interface is invoked by using the pass by reference method. The local interface does not support invocations between different J2EE applications even if they are running on the same J2EE server.

8.4.2 Using the functionality for optimizing the local invocation of the remote interface

This method uses the functionality (*local invocation optimization functionality*) that invokes by the remote interfacing of Enterprise Bean in the same J2EE applications or the same server by delaying the same threads. It maintains location transparency since it creates a program that uses the remote interface on both the client and server-side.

The default settings specify the use of the local invocation optimization functionality in the same application.

8.4.3 Using the pass by reference functionality of the remote interface

This method aims to increase the processing speed by using pass by reference for transferring large size data such as the objects used for argument and return value when local invocation optimization functionality is used.

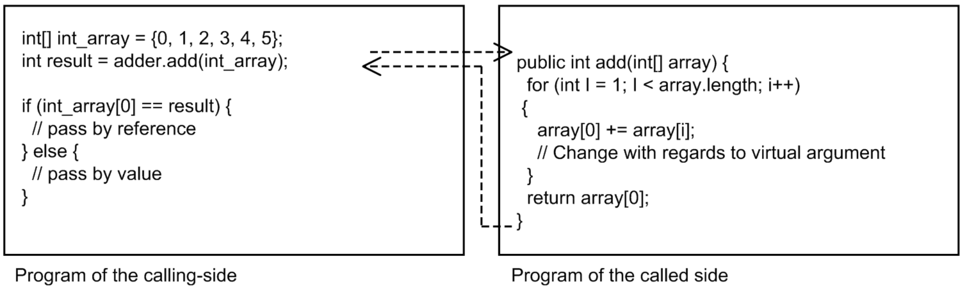
Processing speed can be expected to increase under the following conditions:

Table 8–18: Conditions under which the processing speed can be expected to increase by using pass by reference for remote interface

Item	Processing contents
Processing of business method	The data obtained through virtual argument is only passed by reference without being changed directly
Argument and return value of business method	In this process, classes like the array and collection transfer large data
Location of the Enterprise Bean to be invoked	In similar J2EE applications or similar J2EE servers

Transfer of data in remote interface uses call by value. A program that is created on the basis of call by value may not work properly. An example of a program that may not work properly is shown below:

Figure 8–6: Example of a program that may not work properly



```
int[] int_array = {0, 1, 2, 3, 4, 5};
int result = adder.add(int_array);

if (int_array[0] == result) {
  // pass by reference
} else {
  // pass by value
}
```

Program of the calling-side

```
public int add(int[] array) {
  for (int I = 1; I < array.length; i++)
  {
    array[0] += array[i];
    // Change with regards to virtual argument
  }
  return array[0];
}
```

Program of the called side

This functionality is not effective in the following cases:

- In the case of primitive type (simple type) (since it is always call by value)
- In case the size of the data to be transferred is small

This functionality can be set in the Enterprise bean or a J2EE server.

8.4.4 Tuning parameters for optimizing the method of invoking the Enterprise Bean

This section explains the methods for setting up the tuning parameters that are used for optimizing the method of invoking an Enterprise Bean.

(1) Using the local interface

Use the local interface defined in J2EE for creating an application.

(2) Using the local invocation functionality of the remote interface

This subsection explains how to set up the tuning parameters for the local invocation functionality of the remote interface.

You specify the items listed in the following table using the Smart Composer functionality and define the parameters in the Easy Setup definition file.

Table 8–19: Tuning parameters of a local invocation functionality of the remote interface

Setup item	Setup target	Location of setup (parameter name)
Scope of local invocation optimization functionality	Logical J2EE server (`j2ee-server`)	`ejbserver.rmi.localinvocation.scope`

(3) Using the pass by reference functionality of the remote interface

This subsection explains how to set up tuning parameters for the pass by reference functionality of the remote interface.

You specify the items listed in the following table using the Smart Composer functionality and define the parameters in the Easy Setup definition file.

Table 8–20: Tuning parameters of the pass by reference functionality of the remote interface (items to be specified using the Smart Composer functionality)

Setup item	Setup target	Parameter name
Usage of the pass by reference functionality of the remote interface (each J2EE server)	Logical J2EE server (`j2ee-server`)	`ejbserver.rmi.passbyreference`

You specify the items listed in the following table using the server management command (`cjsetappprop`), and define the parameters in the Session Bean property file or Entity Bean property file.

Table 8–21: Tuning parameters of the pass by reference functionality of the remote interface (items to be specified using the server management command (`cjsetappprop`))

Setup item	Parameter name
Usage of the pass by reference functionality of the remote interface (each Enterprise Bean)	`<pass-by-reference>`

8.5 Optimizing the database access method

The optimization of the method of accessing a database is explained below:

The methods of accessing a database in the J2EE server can be tuned in the following two ways:

- Connection pooling
- Statement pooling

8.5.1 Using connection pooling

The advantages of using connection pooling in the J2EE server and the functionality that can be used in relation to connection pooling are explained below:

In the JPA provider, you can also acquire a DB Connector connection.

(1) Advantages of using connection pooling

Establishing a connection with an EIS such as a database is a process that involves excessive loading. The load can be reduced by using connection pooling. *Connection pooling* is a function that aims to enhance the processing efficiency by pooling the connections that are secured and generated by a J2EE server and reusing those connections. Connection pooling is effective in preventing the deterioration of performance rather than acquiring a connection each time the database is accessed.

The functionality that can be used differs depending on the method used for connecting with the EIS. For details, see *3.14.1 Connection pooling* in the *uCosminexus Application Server Common Container Functionality Guide*.

By default, connection pooling is enabled.

(2) Usable functionality

The functionality that can be used in connection pooling and the guidelines for setup are explained below.

The following functionality can be used in the case of connection pooling:

- Specify the maximum and minimum number of connections for pooling
- Check the connections in the pool and delete the unnecessary connections#1
- Retry in the case of a failure in establishing a connection#2
- Use sweeper to delete unused connections from the pool#2
- Pool the connections in advance by using connection warming-up
- Register the requests for connection in the queue when all connections are exhausted
- Reduce the unnecessary connections gradually from the connection pool

#1

When this functionality is used, you can also set the timeout for checking the connections.

#2

This functionality is disabled by default. Use if required.

(a) Specify the minimum and maximum number of connections that can be pooled

The setup of connection pooling requires you to set the minimum and maximum number of connections that can be pooled. If you specify unlimited maximum number of connections, unlimited connections are established.

Set the minimum and maximum values by referring the values such as number of concurrently executed accesses to an EIS like the database, the number of transactions, and the number of concurrent executions of the business that occurs normally.

We recommend that the maximum number of connections to be pooled be set up in such a way so that the following relationship expression is satisfied between the number of concurrently executed threads and the Session Bean instance pool:

```
Maximum number of connections to be pooled ≥ Session Bean instance pool ≥
Number of concurrently executed threads
```

(b) Check the connections in the pool and delete unnecessary connections

Check that there is no failure in any connection in the pool when acquiring a connection or check for any failure in the connection pool at regular intervals and delete the connection in which a failure has occurred, from the pool (*detect connection failure*). This helps to prevent establishing a connection in which a failure has occurred at the point of time it was acquired and reduces the probability of connection failure. This functionality is effective in the case of connections for accessing the database using the DB Connector. This functionality cannot be used when `direct` is specified in the selectMethod property of the DB Connector.

The guidelines for deciding the timing for detecting the connection failure are given below. Decide according to the type of business.

Table 8–22: Guidelines for deciding the timing for detecting the connection failure

Type of business	Timing
• Business that does not allow failures in connecting with the EIS • Business having low connection frequency	Hitachi recommends settings to detect any failure when acquiring a connection. The processing time required for acquiring a connection is more as compared to when detection of any failure in acquiring a connection is not executed, but the risk of acquiring a connection in which a failure has occurred is reduced.
• Business with high connection frequency • Business that allows connection to a certain extent even when the secured connection has a failure	Hitachi recommends settings to regularly check for any failures. You can prevent the performance from deteriorating by the process of detecting a connection failure, by prolonging the check interval to some extent. There is a risk of acquiring a connection in which a failure has occurred.

When detection of connection failure is performed, you can also set a timeout for detecting a connection failure. Sometimes, when there is no response from the resource due to a server failure and a network failure, it is likely that there may be no response for detecting a connection failure. Timeout is set so that the process of detecting a connection failure can be finished and the process can continue even when there is no response from the resource. In such a case, it is assumed that a connection failure has occurred.

Tip

- When timeout is specified for detecting a connection failure, the connection management threads are generated according to the number of connections in the connection pool of the system. You need to be more careful because when timeout is specified for detecting a connection failure, a large amount of memory is consumed as compared to when a timeout is not specified. The number of connection management threads created is twice the maximum number of connections in the connection pool. Estimate the required memory appropriately.

- Connection management threads are used along with the timeout of connection adjustment functionality. The timeout of connection adjustment functionality is also enabled when a timeout is specified for detecting a connection failure.
- When detecting connection failure by enabling the timeout for detecting connection failure, the unused connections that are removed from the connection pool are not counted in the number of connections of connection pool. As a result, the sum of the connections in the connection pool and the unused connections that are removed from the connection pool might temporarily exceed the maximum number of connections specified for the connection pool.

(c) Retry connection when it fails

If an attempt to connect fails, it is no longer necessary to use a user program for retrying the connection. Set the retry connection so that even when there is decrease in the response of business processing, the business does not terminate due to the failure.

The guidelines for setting a retry frequency and retry interval are as follows:

- If you increase the retry frequency or lengthen the retry interval, there may be a waiting period if the process for acquiring the connection has started.
- If the time setting derived from multiplying the retry interval with the retry frequency is too long, timeouts occur in the RMI-IIOP communication. However, continued attempts for reconnecting are also made after the timeout. Therefore, set up values for the retry interval and the retry frequency such that their product is lesser than the timeout value.
- Although the retry frequency and retry interval will also depend on the approximate time required by the business that uses connection of the same resource, Hitachi recommends that you specify at least 10 seconds.

(d) Automatically delete the unused connections from the connection pool by using a sweeper

The connections that remain idle for a given period of time are automatically deleted from the pool by using a sweeper. Set this functionality when it is likely that the failure will occur if the unused connections are pooled. This functionality can be set in the resource adaptor.

(e) Pool the connections in advance by using connection warming up

This is a functionality for securing the minimum number of connections in advance and pooling them when a resource adaptor or a J2EE server is started. With this functionality, you can enhance the response for the connection request from the application immediately after beginning to use the connection pool. The time required for processing is longer when the resource adaptor or the J2EE server is started.

(f) Enter a request for acquiring a connection in a queue when all connections are exhausted

This functionality enters a request for acquiring a connection in a queue, when there is a request for connection after all the pooled connections are exhausted.

The request for connection can be re-opened immediately when a used connection is released or when a connection is deleted and the number of connections falls below the maximum number of pooled connections. Additionally, the waiting time can be specified, and hence, an error might occur when the connection cannot be established after the specified period of time is elapsed.

(g) Reduce unnecessary connections from the connection pool gradually

This functionality reduces the number of connections in case there are unnecessary connections in the connection pool (*Connection adjustment functionality*).

Check the number of connection pools at regular intervals. With the maximum number of concurrently executed connections until the check is performed as the standard value, delete the excess connections from the pool if there are connections in the pool that exceed this standard value. This will reduce the connections in the pool to a number adequate for actual operational performance and enable economizing on resources and costs of generating a connection.

You can specify a timeout value for the deletion process of connections. Use connection management thread for timeout. Connection management thread is also used for the timeout of detecting a connection failure.

> **Tip**
>
> Connection management thread is used along with the timeout of detecting a connection failure. When a timeout has been set for the connection adjustment functionality, the timeout is also enabled for detecting a connection failure.
>
> Also, if you use the connection adjustment functionality, the unused connections removed for adjusting the number of connections of connection pool are not counted in the number of connections of connection pool. As a result, the sum of the connections in connection pool and the unused connections that are removed from the connection pool might temporarily exceed the maximum number of connections specified for the connection pool.

8.5.2 Using statement pooling

The advantages of using a statement pooling and the guidelines for its setup are explained below:

This functionality can be used when a DB Connector is used. The use of this functionality depends on the method used for connecting with an EIS. A statement pooling cannot be used when the database version used is older than XDM/RD E2 11-01. For details, see *3.14.4 Statement pooling* in the *uCosminexus Application Server Common Container Functionality Guide*.

(1) Advantages of using a statement pooling

The process of generating statements such as SQL statements and stored procedure that is necessary for accessing the database increase the load. The use of a statement pooling helps to reduce the load. *Statement pooling* is a functionality that enhances the processing efficiency by pooling in advance the `PreparedStatement` and the `CallableStatement` that are generated once, and reusing them. The processing efficiency is enhanced as compared to the processing efficiency for generating the statements each time the database is accessed.

Now, the `PreparedStatement` and the `CallableStatement` are the instances of the `java.sql.PreparedStatement` and `java.sql.CallableStatement`, respectively that are the APIs of JDBC.

(2) Guidelines for setup

To use statement pooling, it is necessary to consider the following points before starting the application:

- Use a `java.sql.Connection#prepareStatement` method having the same signature.

- Specify the same argument value in the `java.sql.Connection#prepareStatement` method.

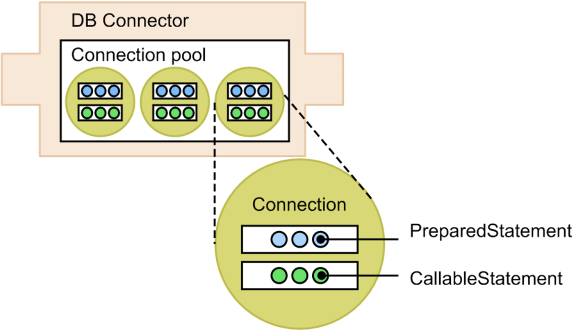
Statement pooling does not work properly in the applications that do not meet the above conditions.

Use statement pooling after understanding its relationship with the connection pooling. The points you must note while configuring the application and setting up the environment are explained as follows:

- The `PreparedStatement` and the `CallableStatement` are pooled for each instance of the physical connection. If there are multiple connections pooled when acquiring a connection, it is likely that a connection, for which statements are not yet pooled, may be allocated.
- You need to set the pool size keeping in mind the maximum value of each connection of the JDBC driver that is used, in order to pool the instances of the respective statements.

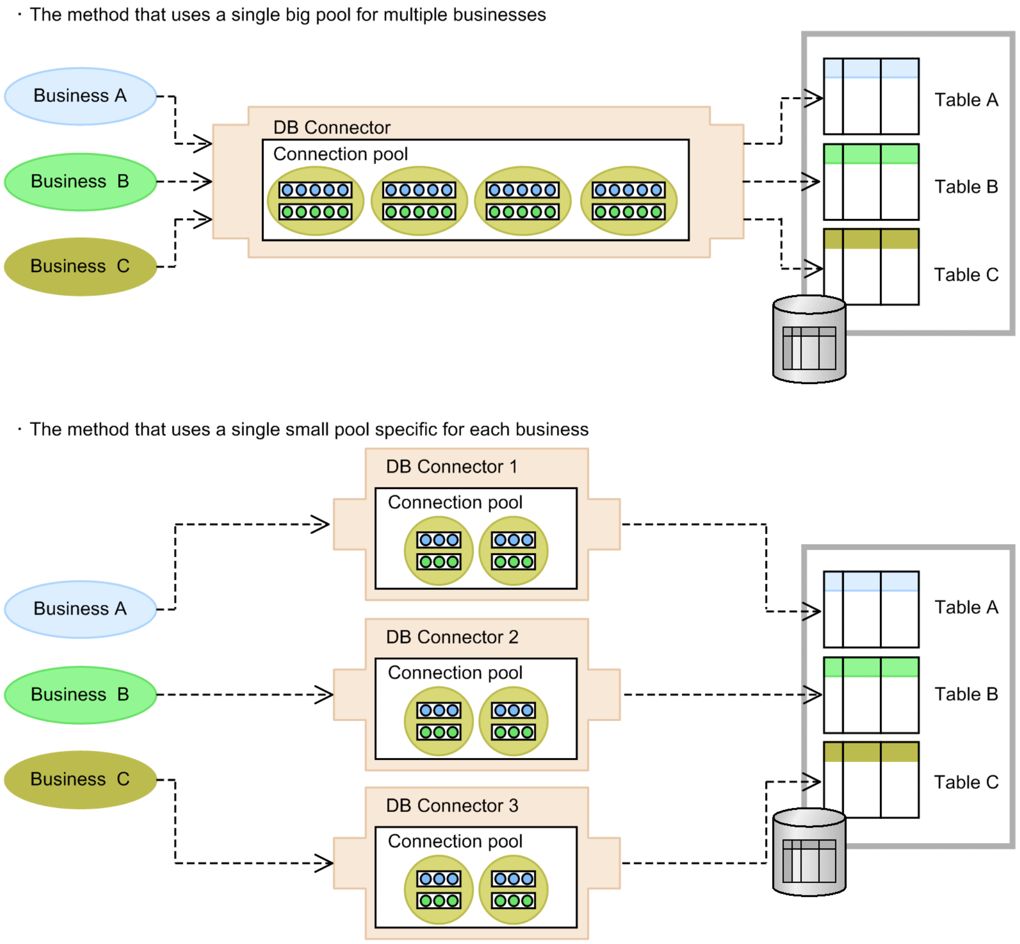
The following figure shows the relationship between the connection pool and the statement pool:

Figure 8–7: Relationship between a connection pool and a statement pool



Determine the size of a pool in the case of statement pooling after considering its relationship with connection pooling according to the contents of the business process. For example, in the case of accessing different tables and using a different SQL even when the same database is used in different business processes, it may be more effective to prepare separate DB Connectors for each business process and prepare the connection pool and the statement pool of only the size required for that business instead of increasing the size of the statement pool and connection pool with one DB Connector.

Figure 8–8: Use of a connection pool and statement pool according to the business process



· The method that uses a single big pool for multiple businesses

· The method that uses a single small pool specific for each business

If you decrease the size of the pool by using multiple DB Connectors, then there is no surplus of connections pooled for each business that is likely to result in a shortage of connections during the peak access. Estimate in detail the number of concurrent executions of the business and tune the settings to avoid any shortage of connections as far as possible. In addition, tune the size of the pool for each business depending on how many statements are used.

Determine the size of the statement pool when a single DB Connector is used by adding up the following values. Estimate the pool size considering that DB Connector uses one statement pool internally.

How to calculate the size of PreparedStatement pool and the guidelines for its setup

Assume the resource limit of the JDBC driver of each connection as the maximum value, and calculate the size based on the number of usages of `PreparedStatement` that uses the same DB Connector. Set the resource limit of the JDBC driver according to the limit value of the JDBC driver that is being used.

The number of usages is obtained by adding up following numbers:

- Number of invocations for `java.sql.Connection#prepareStatement` method by specifying different arguments from the servlet and JSP
- Number of invocations for `java.sql.Connection#prepareStatement` method by specifying different arguments from the Session Bean and the Entity Bean (BMP)
- Number of SQLs used in `PreparedStatement` internally by the J2EE server for the Entity Bean (CMP)
- Number of invocations for `java.sql.Connection#prepareStatement` method by specifying different arguments from the Message-driven Bean

How to calculate the size of CallableStatement pool and the guidelines for its setup

Assume the resource limit of the JDBC driver for each connection as the maximum value, and calculate the size based on the number of usages of `CallableStatement` that use the same DB Connector. Set the resource limit of the JDBC driver according to the limit value of the JDBC driver that is being used.

The number of usages is obtained by adding up the following values:

- Number of invocations for `java.sql.Connection#prepareCall` method by specifying different arguments from the servlet and JSP
- Number of invocations for `java.sql.Connection#prepareCall` method by specifying different arguments from the Session Bean and the Entity Bean (BMP)
- Number of invocations for `java.sql.Connection#prepareCall` method by specifying different arguments from the Message-driven Bean

8.5.3 Tuning parameters for optimizing the method of accessing the database

How to set up the tuning parameters used to optimize the method of accessing a database is explained in this section.

(1) Connection pooling

This subsection explains how to set up tuning parameters for connection pooling.

Specify the items listed in the following table using the server management command (`cjsetresprop`/ `cjsetrarprop`) for each resource adapter and define the parameters in the Connector property file.

Table 8–23: Tuning parameters of connection pooling

Setup item	Location of setup (parameter name) #1
Minimum number of connections for pooling in the connection pool	`MinPoolSize` specified in `<property>` tag
Maximum number of connections for pooling in the connection pool	`MaxPoolSize` specified in `<property>` tag
Select the method for checking connection failures within the pool	`ValidationType` specified in `<property>` tag
Time interval for regularly checking for any connection failure within the pool	`ValidationInterval` specified in `<property>` tag
Specify whether the requests for connections are to be queued when all connections are in use	`RequestQueueEnable` specified in `<property>` tag
Waiting time when a connection request is managed in a queue when all connections are being used	`RequestQueueTimeout` specified in `<property>` tag
Retry frequency in the case of a connection failure#2	`RetryCount` specified in `<property>` tag
Retry interval in the case of a connection failure#2	`RetryInterval` specified in `<property>` tag
Time until it is decided to automatically delete the connections, from the time connection was used last	`ConnectionTimeout` specified in `<property>` tag
Time interval before automatic deletion of connection is executed (connection sweeper)	`SweeperInterval` specified in `<property>` tag

Setup item	Location of setup (parameter name) #1
Specify whether connection warm-up is to be used	`Warmup` specified in `<property>` tag
Specify whether to set a timeout for detecting the connection failure#3	`NetworkFailureTimeout` specified in `<property>` tag
Specify whether to set a timeout for the deletion process of connection adjustment functionality#3	
Time interval for executing the connection adjustment functionality	`ConnectionPoolAdjustmentInterval` specified in `<property>` tag

#1

Set the member resource adapter when using the cluster connection pool functionality (for compatibility).

#2

You cannot set this item when using the cluster connection pool functionality (for compatibility).

#3

Common connection management method is used when timeout is used for detecting a connection failure and the connection adjustment functionality. When either of the timeout is set, timeout is enabled for both connection failure detection and connection adjustment functionality. When you want to change the timeout of connection failure detection and connection adjustment functionality, change the settings of the parameter (`ejbserver.connectionpool.validation.timeout`) for the J2EE server setup in the Easy Setup definition file. For details, see *4.11.2 Parameters used for setting up the user properties for the J2EE server* in the *uCosminexus Application Server Definition Reference Guide*.

Tip

We recommend the following settings to use the connection pooling functionality in HiRDB or XDM/RD E2:

- To use the connection pooling functionality in HiRDB
  Set the HiRDB client environment variable such that the HiRDB server does not sever the connection. For details, see the *HiRDB UAP Development Guide*.
- To use the connection pooling functionality in XDM/RD E2
  Specify `0` in the `SVINTERVAL` parameter of the control statement for invoking the control space or the control statement for invoking the server space of the DB connection server. For details on this parameter, see the manual *VOS3 Database Connection Server*.

If the above settings are not specified, the pooled connections might be disconnected from the database due to a timeout.

- If the error detection functionality is used
  The connections disconnected from the database due to a timeout are destroyed using the error detection functionality. A connection can be acquired normally.
- If the error detection functionality is not used
  The connections disconnected from the database due to a timeout are acquired.

(2) Statement pooling

This subsection explains how to set up the tuning parameters of statement pooling.

Specify the items listed in the following table using the server management command (`cjsetresprop` / `cjsetrarprop`), and define the parameters in the Connector property file.

Table 8–24: Tuning parameters of a statement pooling

Setup item	Parameter name#
Number of PreparedStatement to be pooled for each physical connection	`PreparedStatementPoolSize` specified in `<config-property>` tag
Number of CallableStatement to be pooled for each physical connection	`CallableStatementPoolSize` specified in `<config-property>` tag

#

When using the cluster connection pooling functionality (for compatibility), set this parameter in the member resource adapter.

8.6 Setting a timeout

In the Application Server, you can set a timeout at several points in order to prevent the state where there is no response to a request in case of an error.

This section describes the points where you can set a timeout in the entire system and the guidelines for setting the timeout.

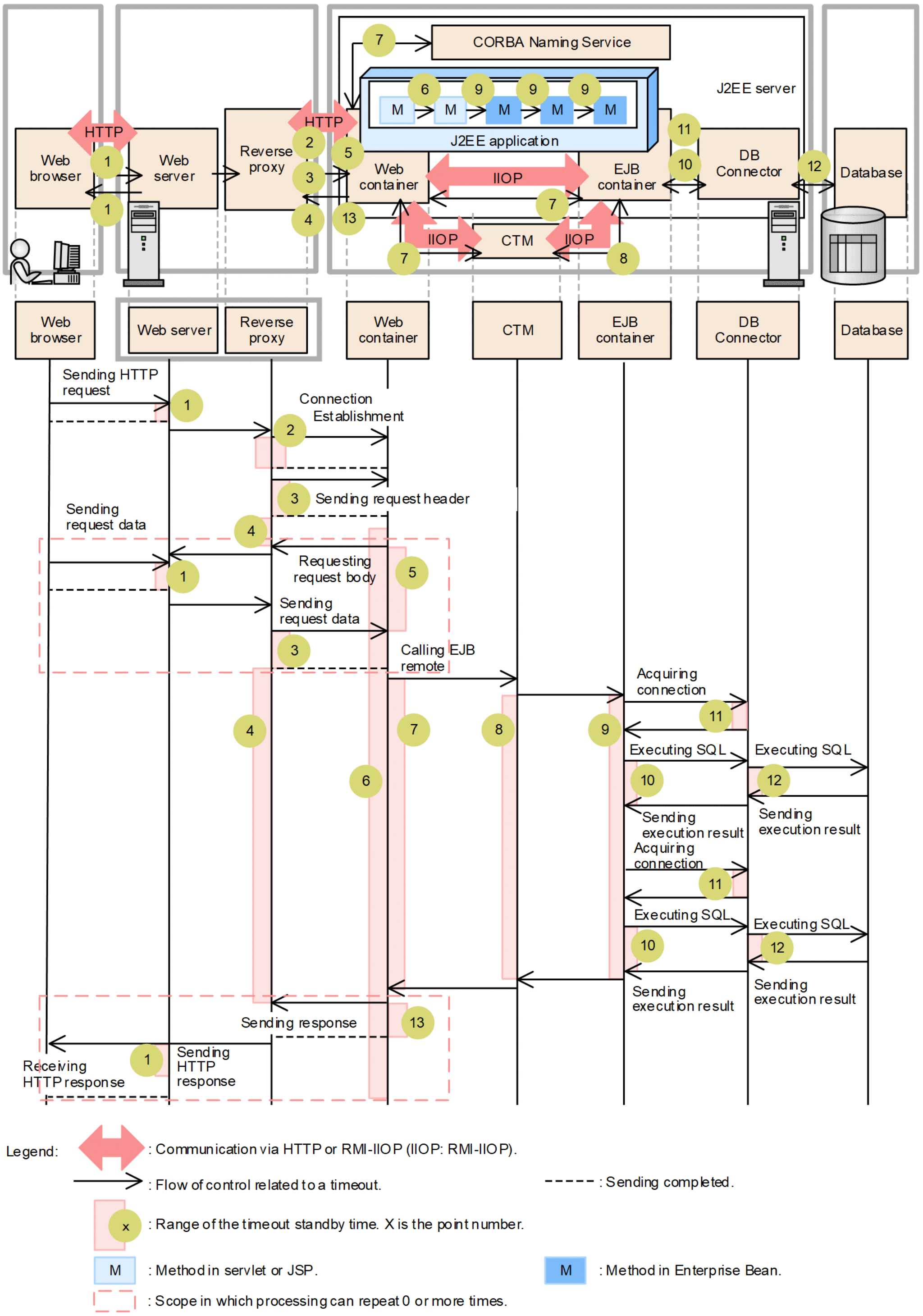
Reference note

When invoking Application Server from OpenTP1 using the TP1 inbound integrated function, you must perform timeout settings in view of the settings in OpenTP1 besides the contents described in this section. For details, see *4. Invoking Application Server from OpenTP1 (TP1 inbound integrated function)* in the *uCosminexus Application Server Common Container Functionality Guide*.

8.6.1 Points where a timeout can be set

In the systems used for executing J2EE applications, you can set up timeouts at the points shown in the following figure. In the following figure, a Web browser is used as the client. The points will differ when you integrate with a Web server, and when you send and receive requests directly to and from the J2EE server without going through a Web server.

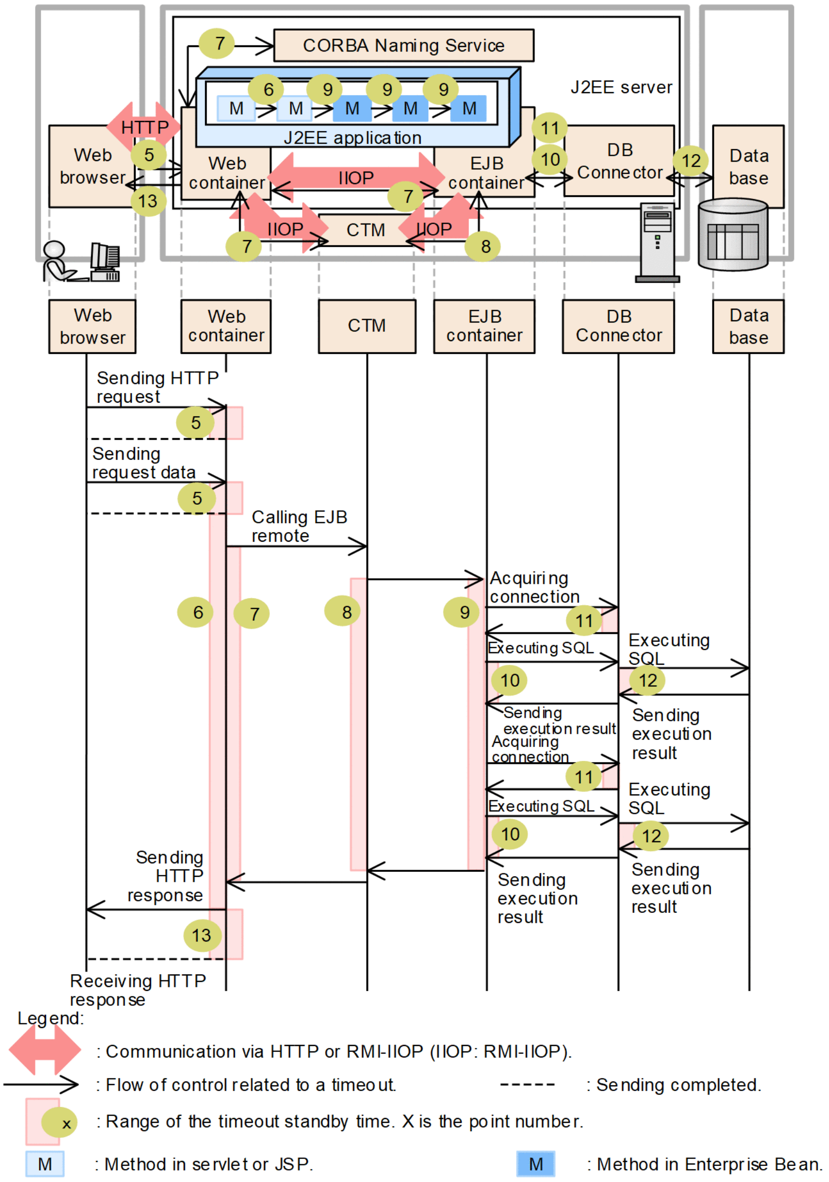
Figure 8–9: Points where a timeout can be set (for Web server integration)



If the client is an EJB client, replace the Web container with the EJB client. You can set the timeout ranging from the EJB client up to the database.

When sending and receiving requests directly to and from the J2EE server without going through the Web server, this process does not apply to the Web server and the reverse proxy. For this reason, points 1 to 4 do not apply. The following figure shows the points where a timeout can be set when sending and receiving requests directly to and from the J2EE server.

Figure 8–10: Points where a timeout can be set (for direct requests to and from J2EE server)



The timeout specified at each point has a specific use that is described in the table below:

Table 8–25: Purpose of timeout set at each point and default timeout settings

Points	Type of timeout	Primary usage
1	Timeout set in the Web server for receiving the request from the client and sending the data to the client	Detecting failures in the communication path or the Web server
2	Connection timeout specified on the reverse proxy side in the processing for sending requests to the Web container	Detecting failures in the communication path or the Web container
3	Request header and request body transmission timeout specified on the reverse proxy side in the processing for sending requests to the Web container	Detecting failures in the communication path or the Web container
4	Timeout for data reception from the Web container set on the reverse proxy side	Detecting failures in business processing (such as infinite loop and deadlock) of the J2EE server or the communication path

Points	Type of timeout	Primary usage
5	Timeout for data reception from the reverse proxy or Web client set on the Web container side	Detecting failures in the communication path, failures in the Web server, or access from unauthorized clients
6	Timeout set in the Web application for the method execution time	Detecting failures in business processing (such as infinite loop and deadlock) of the J2EE server
7	Timeout set in the EJB client for remotely invoking the Enterprise Bean (RMI-IIOP communication) and for invoking the JNDI Naming Service	Detecting failures in business processing (such as infinite loop and deadlocks) of the J2EE server or the communication path
8#	Timeout set up in the EJB client for invoking the Enterprise Bean from CTM	Detecting failures in business processing (such as infinite loop and deadlocks) of the J2EE server or the communication path
9	Timeout set in the EJB for the method execution time	Detecting failures in business processing (such as infinite loop and deadlock) of the J2EE server
10	Timeout set in the EJB container for the database transaction	Detecting failures in database server (such as server is down or a deadlock has occurred) or preventing the extended exclusive use of the resources
11	Timeout set in DB Connector for acquiring a connection	Detecting errors when a connection is acquired (communication path errors or resource depletion)
12	Database timeout	Detecting failures in database server (such as server is down or a deadlock has occurred) or preventing the extended exclusive use of the resources
13	Timeout for sending a response to the reverse proxy or Web client set on the Web container side	Detecting failures in the communication path and failures in the Web server

#

This point exists only when you are using CTM. For a configuration in which CTM is not used, the range of point 7 extends from the time of execution of remote invocation of the EJB from the Web container to the EJB container, until the dispatch of execution result from the EJB container to the Web container.

The basic guidelines for setting the above timeouts are as follows:

- The general rule for setting a timeout value is that the closer a point is from the invocation origin (Web client or EJB client), the higher the timeout value. It is, therefore, recommended to use the following relationship for setting the timeout.
  - Point 1 < Point 5
  - Point 4 > Point 6 > Point 7
  - Point 7 = Point 8 > Point 9 > Point 10
  - Point 10 > Point 11
  - Point 9 > Point 12
  - Point 1 < Point 13
- When setting the timeout values for points 4, 7, 10 and 12, first check the amount of time normally taken by the invocation process, and then calculate and set the timeout value for each invocation process (business).

The points 1 to 13 can be divided into the following three categories depending on their location in the system:

- For more information on points (1 to 6, and 13) that need to be considered in a Web front-end system.
  For details, see *8.6.2 Setting the timeout in a Web front-end system*.
- Points (7 to 9) that need to be considered in the back system
  For details, see *8.6.3 Setting a timeout in the back-end system*.

- Point (10 to 12) that need to be considered during database connection

  This point needs to be further classified into a transaction timeout, DB Connector timeout, and database timeout. For details, see *8.6.4 Setting the transaction timeout* and *8.6.6 Setting the database timeout*.

For details about the settings of each point, see *8.6.8 Tuning parameters for setting the timeout*.

Reference note

The default values for each point are as follows:

Point	Default value
1	60 seconds
2	60 seconds
3	60 seconds
4	60 seconds
5	300 seconds
6	Not set. No timeout.
7	Not set. Continues to wait for response.
8	A value same as point 7 is automatically inherited and set up when the Enterprise Bean is invoked.
9	Not set. No timeout.
10	180 seconds
11	Differs according to the location of the timeout setup. • A timeout in establishing a physical connection: 8 seconds • A timeout in the request for connection during connection depletion: 30 seconds • A timeout in detecting a connection error: 5 seconds
12	Differs according to the type of database and the location of timeout setting. For HiRDB Unlock waiting timeout: 180 seconds Response timeout: 0 seconds (The HiRDB client continues to wait until there is a response from the HiRDB server.) Request interval timeout: 600 seconds For Oracle (when global transaction is used) Unlock waiting timeout: 60 seconds For SQL Server Timeout in acquiring memory: -1 (For details about the operations when -1 is specified, see the SQL Server documentation) Unlock waiting timeout: -1 (Continues to wait until the lock is released) For XDM/RD E2 Unlock waiting timeout: None (timeout is not monitored) CPU timeout during SQL execution: 10 seconds SQL execution timeout: 0 seconds (timeout is not monitored) Transaction timeout: 600 seconds Response timeout: 0 seconds (The HiRDB client continues to wait until there is a response from the XDM/RD E2 server.)
13	300 seconds

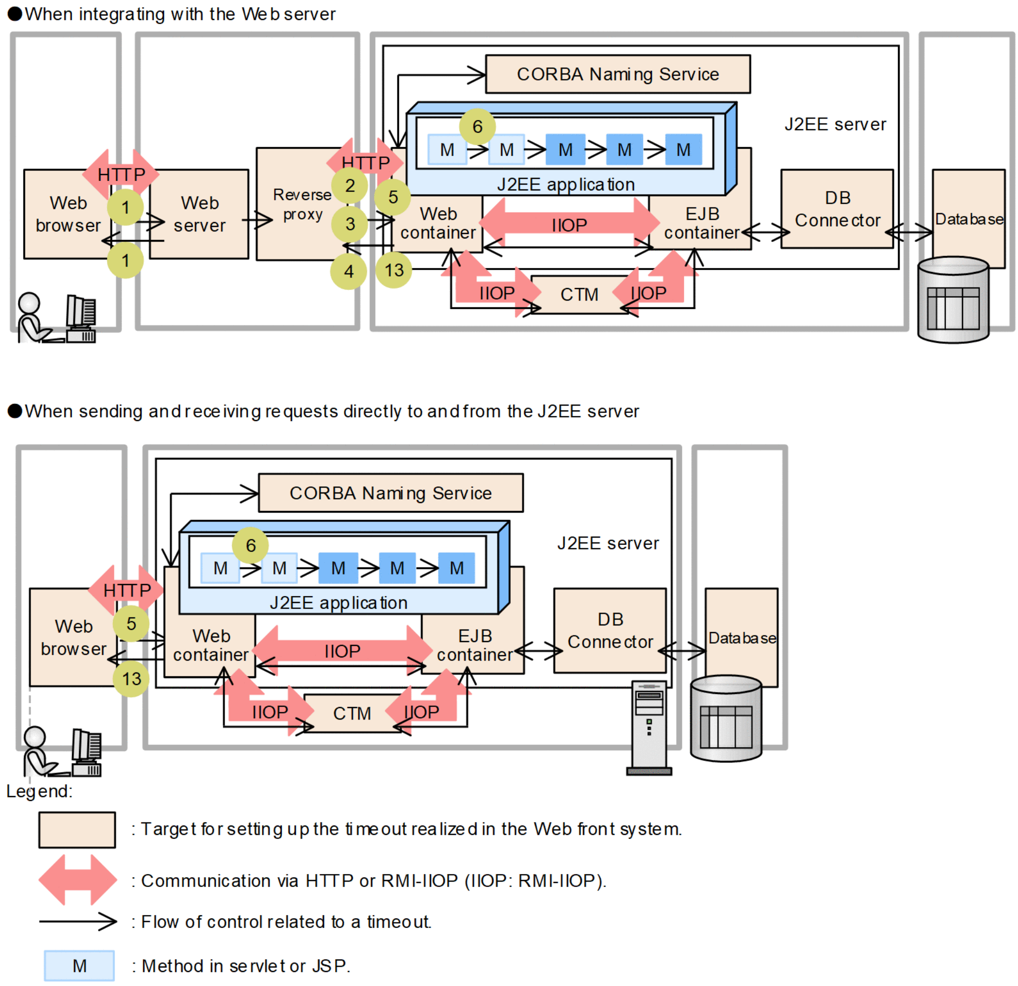
8.6.2 Setting the timeout in a Web front-end system

This section explains the settings of timeout in a Web front-end system.

When setting the timeout in a Web front-end system, amongst all the timeout values for the entire system, you need to consider points 1 to 6 and 13 shown in the following figure. These numbers correspond to *Figure 8-9* or *Figure 8-10*.

> **Tip**
>
> When sending requests directly to and from the J2EE server without going through the Web server, you can set timeouts at points 5 to 13. Points 1 to 4 do not apply in this situation.

Figure 8–11: The timeout points (points 1 to 6 and 13) to be considered in the case of a Web front-end system



- **Waiting time in the Web server for receiving requests from the client and sending the data to the client (point 1)**

  When there is a backlog of requests from the Web browser, the timeout is used to release Web server resources. When there is a backlog of responses to the Web browser (when the Web browser is not receiving the responses), the timeout is used to release resources for the Web server and for the Web container in the J2EE server.

  The waiting time is set to the same value in each case. A timeout can only be set at point 1 in the case of Web server integration.

- **Waiting time for sending requests to the Web container in the reverse proxy registered in the Web server (points 2 and 3)**

When sending a request from the reverse proxy to the Web container, control might be lost if a problem occurs in the Web container itself or in the communication path between the Web server and the Web container. In this case, the timeout triggers the release of Web server resources. At the same time, the error is notified to the Web browser. You can set the timeout at this point only in the case of Web server integration.

Point 2 is the waiting time for establishing connection with the Web container and Point 3 is the waiting time for the process of sending requests to the Web container.

- **Waiting time for receiving data from the Web container in the reverse proxy registered in the Web server (point 4)**

  If control is lost due to a problem in the J2EE application, the timeout triggers the release of Web server resources. At the same time, the error is notified to the Web browser. You can set the timeout at this point only in the case of Web server integration.

  > **Tip**
  >
  > Timeout points are set at the level of the reverse proxy data transfer destination. For this reason, if the time required for processing varies from business process to business process, we recommend that you define reverse proxy data transfer destinations at the level of Web applications corresponding to business processes and set timeouts accordingly.

- **Waiting time for receiving data from the reverse proxy or the Web client in the Web container (point 5)**

  In the case of Web server integration, when sending a request from the reverse proxy to the Web container, a backlog of requests might form due to a problem in the Web server itself or a problem in the communication path between the Web server and the Web container. In this case, the timeout triggers the release of J2EE server (Web container) resources.

  When sending and receiving requests directly to and from the J2EE server without going through the Web server, when there is a backlog of requests from the Web browser, the timeout triggers the release of J2EE server (Web container) resources.

- **Waiting time for processing request in the Web container (point 6)**

  The functionality of monitoring the execution time of J2EE application is used to set this timeout. See *8.6.7 Setting the method timeout in the J2EE application*.

- **Waiting time for sending response from the Web container to the reverse proxy or the Web client (point 13)**

  In the case of Web server integration, when sending a response from the Web container to the reverse proxy, control might be lost due to a problem in the Web server itself or a problem in the communication path between the Web server and the Web container. In this case, the timeout triggers the release of Web container resources.

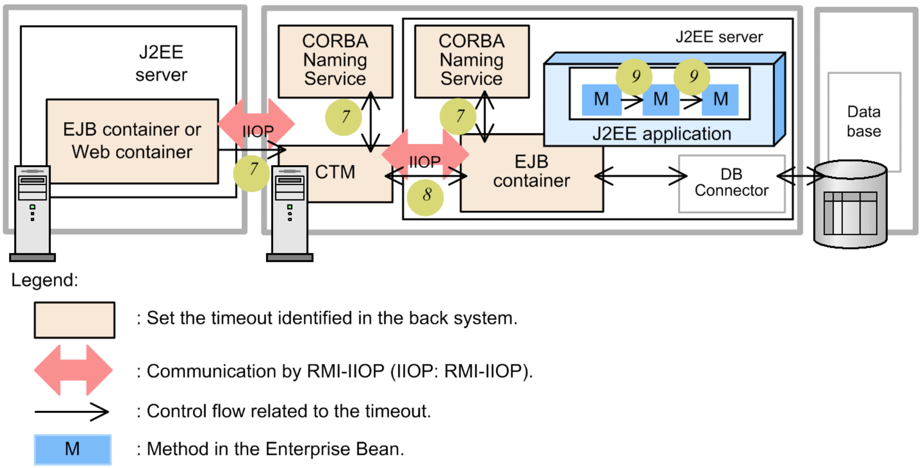
  When sending and receiving requests directly to and from the J2EE server without going through the Web server, if communication with the Web browser is stalled, the timeout triggers the release of Web container resources.

8.6.3 Setting a timeout in the back-end system

This section describes the timeout settings in the back-end system. Among the timeout values set in the back-end system, the value related to the transactions with EIS, such as a database, is described in *8.6.4 Setting the transaction timeout*. The section here describes the timeout related to the EJB client and the EJB container.

When setting a timeout in the back-end system, amongst the timeout values of the entire system, you need to consider the points 7 and 8 shown in the following figure. These numbers correspond to *Figure 8-9* or *Figure 8-10*.

Figure 8–12: Timeout points to be considered in the case of a back-end system


J2EE server
EJB container or Web container
IIOP
7
CORBA Naming Service
7
CTM
IIOP
8
CORBA Naming Service
7
EJB container
J2EE server
9
9
M
M
M
J2EE application
DB Connector
Data base
Legend:
: Set the timeout identified in the back system.
: Communication by RMI-IIOP (IIOP: RMI-IIOP).
: Control flow related to the timeout.
M
: Method in the Enterprise Bean.

- **Waiting time in the client when the Enterprise Bean is invoked remotely (RMI-IIOP communication) and when the CORBA Naming Service is invoked (point 7)**

  If the control does not return due to an error in accessing the CORBA Naming Service or the J2EE application, the error is notified to the EJB client by the timeout.

- **Waiting time at client side when the Enterprise Bean is invoked from CTM or the EJB client (point 8)**

  For trouble such as infinite loop or deadlock occurs in J2EE applications, the CTM resources are released. The error is reported to EJB client.

  > **Tip**
  >
  > When invoking the Enterprise Bean from the EJB client, you can specify the timeout in `usrconf.properties` or in the API (method of the `com.hitachi.software.ejb.ejbclient.RequestTimeoutConfig` class) provided by Application Server.
  >
  > The definition of the `usrconf.properties` affects the entire process. The timeout value specified in the API affects only the thread or the object that invokes the business method. The API specifications override the definitions of `usrconf.properties`.
  >
  > Hitachi, therefore, recommends that you define the standard values that you want to set for the entire process, in the `usrconf.properties` and use the appropriate API to set the detailed values as per the invoked business.

  If there is a timeout during the invocation of the Enterprise Bean, the `javax.rmi.RemoteException` (such as `org.omg.CORBA.TIMEOUT`) exception is notified to the EJB client-side. Consequently, the request from the client-side is cancelled. At this point, however, the process will continue if the Enterprise Bean process on the server-side has already started. As a result, the server-side processing terminates normally even in the event of a timeout.

- **Waiting time for method processing in the EJB container (point 9)**

  The functionality of monitoring the execution time of the J2EE application is used to set this waiting time. See *8.6.7 Setting the method timeout in the J2EE application*.

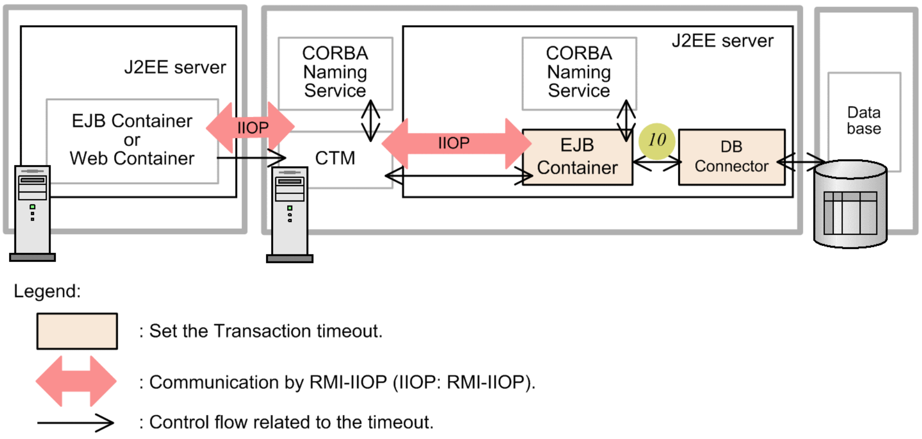
In addition, in point 9, you can also set the timeout value for instances of the `method-ready' pool of the Stateless Session Bean and `pool' of the Entity Bean that cannot be obtained within the specified time. In this case, `java.rmi.RemoteException` (in the case of a remote client) or `javax.ejb.EJBException` (in the case of a local client) is sent to the client.

8.6.4 Setting the transaction timeout

This section explains the transaction timeout settings. Set the timeout for transactions with EIS such as the database system. This section describes the transaction timeout when DB Connector is used for accessing the database.

When setting the transaction timeout, amongst all the timeout values of the entire system, you need to consider the transaction of the database with the EJB container (point 10 in the figure). These numbers correspond to *Figure 8-9* or *Figure 8-10*.

Figure 8–13: Timeout points to be considered in the case of transaction with EIS



The Application Server will execute the following processes when there is a transaction timeout:

- The active transactions are rolled back.
- Close the connections participating in the transaction and delete them from the connection pool.

Tip

The process for defining the transaction timeout differs for CMT and `UserTransaction`.

- **For CMT**

  You can define the transaction timeout for CMT in `usrconf.properties` or you can set the timeout as an attribute of the Enterprise Bean, interface or method. Use the server management commands for setting Enterprise Bean, interface or method attributes.

  The definition of the `usrconf.properties` affects the entire process. The timeout value specified in the Enterprise Bean, interface or method attributes affect only the transactions used by the relevant Enterprise Bean, interface or method. These specifications override the definitions in `usrconf.properties`.

  Hitachi, therefore, recommends that you define the standard values to be set in the entire process, in the `usrconf.properties` and set the detailed values, as per the invoked business, as the attributes of the Enterprise Bean, interface or method.

- **For UserTransaction**

  You can specify a transaction timeout for `UserTransaction` in `usrconf.properties` or in JTA API (the `javax.transaction.UserTransaction#setTransactionTimeout` method). The definition of the `usrconf.properties` affects the entire process. The timeout value specified in API only affects the transactions that issued the API. The API specifications override the definitions in `usrconf.properties`.

  Hitachi, therefore, recommends that you define the standard values to be set in the entire process, in the `usrconf.properties` and use the appropriate API to set the detailed values as per the invoked business.

Tip

Set the values related to transaction timeout and connection establishment in such a way that the following relationship is satisfied.

transaction-timeout > *timeout-for-connection-establishment-when-all-connections-are-in-use* + *timeout-when-failure-is-detected* + *maximum-wait-time-for-connection-establishment* × *execution-count-of-connection-establishment* + *retry-interval* × *retry-count*

This relationship, if shown with parameters, is as follows:

```
ejbserver.jta.TransactionManager.defaultTimeOut > RequestQueueTimeout
+ ejbserver.connectionpool.validation.timeout + loginTimeout ×
(RetryCount+1) + RetryInterval × RetryCount
```

When setting the transaction timeout, you need to account for the execution time of the transaction in addition to the time taken for connection establishment.

If a transaction timeout occurs, the exception is not notified to the user application. The message KDJE31002-W, however, is output to the log file and the J2EE server. An exception is notified when you attempt to use the JTA interface or the JDBC interface by using the relevant transaction from the user application.

8.6.5 Setting up a timeout in DB Connector

This subsection describes the timeout settings in DB Connector.

You can set up the following three types of timeouts with DB Connector:

- **Timeout in establishing a physical connection**
  This timeout occurs when a physical connection is established. You can set the timeout in HiRDB, Oracle, and SQL Server.
  The operation executed when this timeout occurs is as follows:
  - An exception is reported to the user application.
- **Timeout in the request for connection during connection depletion**
  This timeout occurs when waiting for a connection request during connection depletion.
  The operation executed when this timeout occurs is as follows:
  - An exception (`java.sql.SQLException`) is reported to the user application.
- **Timeout in detecting a connection error**
  This timeout occurs when a connection error is detected. This timeout occurs when an error is detected during a connection request.
  The operation executed when this timeout occurs is as follows:
  - The message KDJE48602-W indicating a timeout is output. The connection in which the error was detected is destroyed and a new connection is returned to the user.

8.6.6 Setting the database timeout

This section explains the database timeout settings for the following databases:

- HiRDB
- Oracle
- SQL Server
- XDM/RD E2

In the case of Oracle, the items that can be set differ depending on whether the global transaction or the local transaction is being used.

(1) Timeout in HiRDB

You can set the following three types of timeout values in HiRDB:

- **Unlock waiting timeout**

  Set this timeout for preventing deadlocks and extended exclusive use of the resources. Set the timeout value in the pd_lck_wait_timeout parameter of the common system definition of the HiRDB server. The timeout value set here is the maximum time for monitoring the exclusion waiting time. The exclusion waiting time is the time period from when the exclusion request is in a queue until it is released from the queue.

  When this timeout occurs, the Application Server and HiRDB execute the following operations:

  - An exception (`java.sql.SQLException`) is notified to the user application.
  - The HiRDB message KFPA11770-I, indicating that timeout has occurred, or the message KFPA11911-E, indicating that a deadlock has occurred, are output.
  - The active transactions are rolled back.
  - After terminating the business method of the user application, the connection is closed and deleted from the connection pool.

- **Response timeout**

  Set this timeout for detecting a failure on the server-side of the database system.

  Set the timeout value in the environment variable PDCWAITTIME of the HiRDB client. The timeout period set here is the maximum waiting time of the HiRDB client from when it makes a request to the HiRDB server until a response is returned. Specify this timeout value in cases such as monitoring the time for a long-term SQL.

  When this timeout occurs, the Application Server and HiRDB execute the following operations:

  - An exception (`java.sql.SQLException`) is notified to the user application.
  - The HiRDB message KFPA11732-E, indicating that timeout has occurred, is output.
  - The active transactions are rolled back.
  - The connection is closed and deleted from the connection pool.

- **Request interval timeout**

  Set this timeout for detecting a failure on the client-side of the database system.

  Set the timeout value in the environment variable PDSWAITTIME of the HiRDB client. The timeout value set here is the maximum waiting time of the HiRDB server from when the HiRDB server returns a response to the request from the HiRDB client until the next request is sent by the HiRDB client. The time is monitored for the transactions that are being processed (from the start of SQL execution until commit or rollback). The timeout value is reset when the request from the HiRDB client reaches the HiRDB server.

  When this timeout occurs, the Application Server and HiRDB execute the following operations:

    - An exception (`java.sql.SQLException`) is notified to the user application.
    - The HiRDB message KFPA11723-E, indicating that timeout has occurred, is output.
    - The active transactions are rolled back.
    - The connection is closed and deleted from the connection pool.

- **Connection timeout in the non-block mode**

  This timeout is used to detect a LAN error quickly.

  Specify the timeout value in the HiRDB client environment variable `PDNBLOCKWAITTIME`. The timeout value specified here monitors the connection between the HiRDB server and the HiRDB client.

  Application Server and HiRDB execute the following operation when this timeout is detected:

    - An exception (`java.sql.SQLException`) is reported to the user application.

(2) Timeout in Oracle (for local transaction)

When you use the local transaction in Oracle, you can set up the following timeout values:

- **Query timeout**

  Query timeout is a timeout that you can set up only when you use the Oracle JDBC Thin Driver as a JDBC driver. You use the `setQueryTimeout` method of the `java.sql.Statement` interface to specify a timeout value. For details on the notes when Oracle JDBC Thin Driver is used to connect to Oracle, see *3.6.6 Preconditions and notes on connecting to Oracle* in the *uCosminexus Application Server Common Container Functionality Guide*.

If a deadlock occurs, the Oracle message `ORA-00060` is output. After the Application Server terminates the business method of the user application, the connection is closed and is deleted from the connection pool.

(3) Timeout in Oracle (for global transaction)

You can set up the following timeout values when the global transaction is being used in Oracle:

- **Query timeout**

  For details about the query timeout, see the explanation related to the query timeout in *(2) Timeout in Oracle (for local transaction)*.

- **Unlock waiting timeout**

  You set up this timeout for preventing deadlocks and extended exclusive use of resources. Set up the timeout value in the `DISTRIBUTED_LOCK_TIMEOUT` parameter of the Oracle server definition. When this timeout occurs, the Application Server and Oracle execute the following operations:

    - An exception (`java.sql.SQLException`) is reported to the user application.
    - The Oracle message `ORA-02049`, indicating that the timeout has occurred or the Oracle message `ORA-00060` indicating that a deadlock has occurred, is output.
    - After terminating the business method for the user application, the connection is closed and deleted from the connection pool.

  The active transactions are not rolled back.

(4) Timeout in SQL Server

You can set the following two types of timeout values in the SQL Server:

- **Timeout in acquiring memory**

Set this timeout for monitoring the waiting time for acquiring the memory for SQL execution. Set the timeout value in the query wait parameter of the environment settings option of the SQL Server. The timeout period set here is the waiting time for acquiring the memory, when the memory required for executing SQL is not obtained.

When the timeout occurs, the Application Server and the SQL Server execute the following operations:

- An exception (`java.sql.SQLException`) is notified to the user application.
- The SQL Server message 8645, indicating that timeout has occurred, is output.
- The active transactions are rolled back.
- After terminating the business method of the user application, the connection is closed and deleted from the connection pool.

- **Unlock waiting timeout**

  Set this timeout for preventing deadlock and extended exclusive use of the resources. Set the timeout value by executing the SET LOCK_TIMEOUT statement of the SQL Server. The timeout period set here is the waiting time until the lock is released.

  When timeout occurs, the Application Server and the SQL Server execute the following operations:

  - An exception (`java.sql.SQLException`) is notified to the user application.
  - The SQL Server message 1222, indicating that timeout has occurred, is output.
  - After terminating the business method of the user application, the connection is closed and deleted from the connection pool.

  When a deadlock occurs in the SQL Server, the Application Server and the SQL Server execute the following operations:

  - An exception (`java.sql.SQLException`) is notified to the user application.
  - The SQL Server message 1205, indicating that a deadlock has occurred, is output.
  - The active transactions are rolled back.
  - After terminating the business method of the user application, the connection is closed and deleted from the connection pool.

(5) Timeout in XDM/RD E2

You can set the following five types of timeout values in XDM/RD E2:

- **Unlock waiting timeout**

  Set this timeout for preventing deadlock and extended exclusive use of resources. Set the timeout value in the TIMER parameter of the system option definition of XDM/BASE. The timeout value set here is the maximum time for monitoring the exclusion waiting time. The exclusion waiting time is the time from which the exclusion request is pending in a queue until it is released from the queue.

  When timeout occurs, the Application Server and XDM/RD E2 execute the following operations:

  - An exception (`java.sql.SQLException`) is notified to the J2EE application.
  - The message JXZ1911I of XDM/RD E2, indicating that a timeout or deadlock has occurred, is output.
  - The active transactions are rolled back.
  - After terminating the business method of the J2EE application, the connection is closed and deleted from the connection pool.

  The operations executed when a deadlock occurs in XDM/RD E2 are similar to the operations that are executed when a timeout occurs in unlock waiting timeout.

- **CPU timeout during SQL execution**

Set this timeout for monitoring the CPU processing time during SQL execution. Set the timeout value in the SQLCTIME parameter of the control statement for invoking control space or the control statement for invoking server space of the DB connection server. The timeout period set here is the maximum time for monitoring the CPU processing time during the execution of one SQL. Specify this timeout value when monitoring the time for the long-term SQL.

When timeout occurs, the Application Server and XDM/RD E2 execute the following operations:

- An exception (`java.sql.SQLException`) is notified to the J2EE application.
- A message indicating that timeout has occurred is output. The message differs depending on the value specified in the VPARTOPTION parameter of the control statement for invoking control space of the DB connection server. When the value is not specified or when ERROR NORMAL is specified, the HiRDB client message KFPA11723-E is output. If a value other than that given above is specified, the XDM/RD E2 message JXZ1874I is output.
- The active transactions are rolled back.
- If VPARTOPTION parameter is not specified or if ERROR NORMAL is specified, the connection is closed and deleted from the connection pool. If a value other than that given above is specified, the connection is closed and deleted from the connection pool when the database is accessed for the first time after the timeout occurs, or after the business method of the J2EE application terminates.

- **SQL execution timeout**

  Set this timeout for monitoring the time elapsed during SQL execution. Set the timeout value in the SQLETIME parameter of the control statement for invoking control space or the control statement for invoking server space of the DB connection server. The timeout period set here is the maximum time for monitoring the time elapsed in the execution of one SQL. Specify this timeout value when monitoring the time for the long-term SQL.

  When timeout occurs, the Application Server and XDM/RD E2 execute the following operations:

  - An exception (`java.sql.SQLException`) is notified to the J2EE application.
  - The message indicating that timeout has occurred is output. The message differs depending on the value specified in the VPARTOPTION parameter of the control statement for invoking control space of the DB connection server. When the value is not specified or when ERROR NORMAL or ERROR SQLCTIME is specified, the HiRDB client message KFPA11723-E is output. If a value other than that given above is specified, the message JXZ1874I of XDM/RD E2 is output.
  - The active transactions are rolled back.
  - When VPARTOPTION parameter is not specified, or if ERROR NORMAL or ERROR SQLCTIME is specified, the connection is closed and deleted from the connection pool. If a value other than that given above is specified, the connection is closed and deleted from the connection pool when the database is accessed for the first time after the timeout occurs, or after the business method of the J2EE application terminates.

- **Transaction timeout**

  Set this timeout for monitoring the elapsed time from the starting of the transaction. Set the timeout value in the SVETIME parameter of the control statement for invoking control space or the control statement for invoking server space of the DB connection server. The timeout value set here is the maximum time for monitoring the elapsed time of the transaction.

  When timeout occurs, the Application Server and XDM/RD E2 execute the following operations. If an SQL is being executed when timeout occurs, it is executed at that time. If the SQL is not being executed, it is executed during the SQL execution after the timeout occurs.

  - An exception (`java.sql.SQLException`) is notified to the J2EE application.
  - The message indicating that timeout has occurred is output. The message differs depending on the value specified in the VPARTOPTION parameter of the control statement for invoking control space of the DB connection server. When the value is not specified, or when ERROR NORMAL or ERROR SQLCTIME is specified, the HiRDB

client message KFPA11723-E is output. If a value other than that given above is specified, the XDM/RD E2 message JXZ1874I is output.

  - The active transactions are rolled back.
  - If the VPARTOPTION parameter is not specified, or if ERROR NORMAL or ERROR SQLCTIME is specified, the Application Server and XDM/RD E2 the connection is closed and deleted from the connection pool. If a value other than that given above is specified, the connection is closed and deleted from the connection pool when the database is accessed for the first time after the timeout occurs, or after the business method of the J2EE application terminates.

- **Response timeout**

  Set this timeout for detecting failures on the server-side of the database system.

  Set the timeout value in the environment variable PDCWAITTIME of the HiRDB client. The timeout value set here is the maximum waiting time of the HiRDB client from when it makes a request to the XDM/RD E2 server until the response is returned. Specify this timeout value when monitoring the time for the long-term SQL.

  When timeout occurs, the Application Server and HiRDB execute the following operations:

  - An exception (`java.sql.SQLException`) is notified to the J2EE application.
  - The HiRDB client message KFPA11732-E, indicating that timeout has occurred, is output.
  - The active transactions are rolled back.
  - The connection is closed and deleted from the connection pool.

(6) Processing of the user application when timeout or deadlock occurs during database access

When an exception occurs in the user application due to a deadlock or timeout of the database, roll back the active transactions and suspend the processing of the business method. Check and, if necessary, revise the tuning parameters explained in this section.

8.6.7 Setting the method timeout in the J2EE application

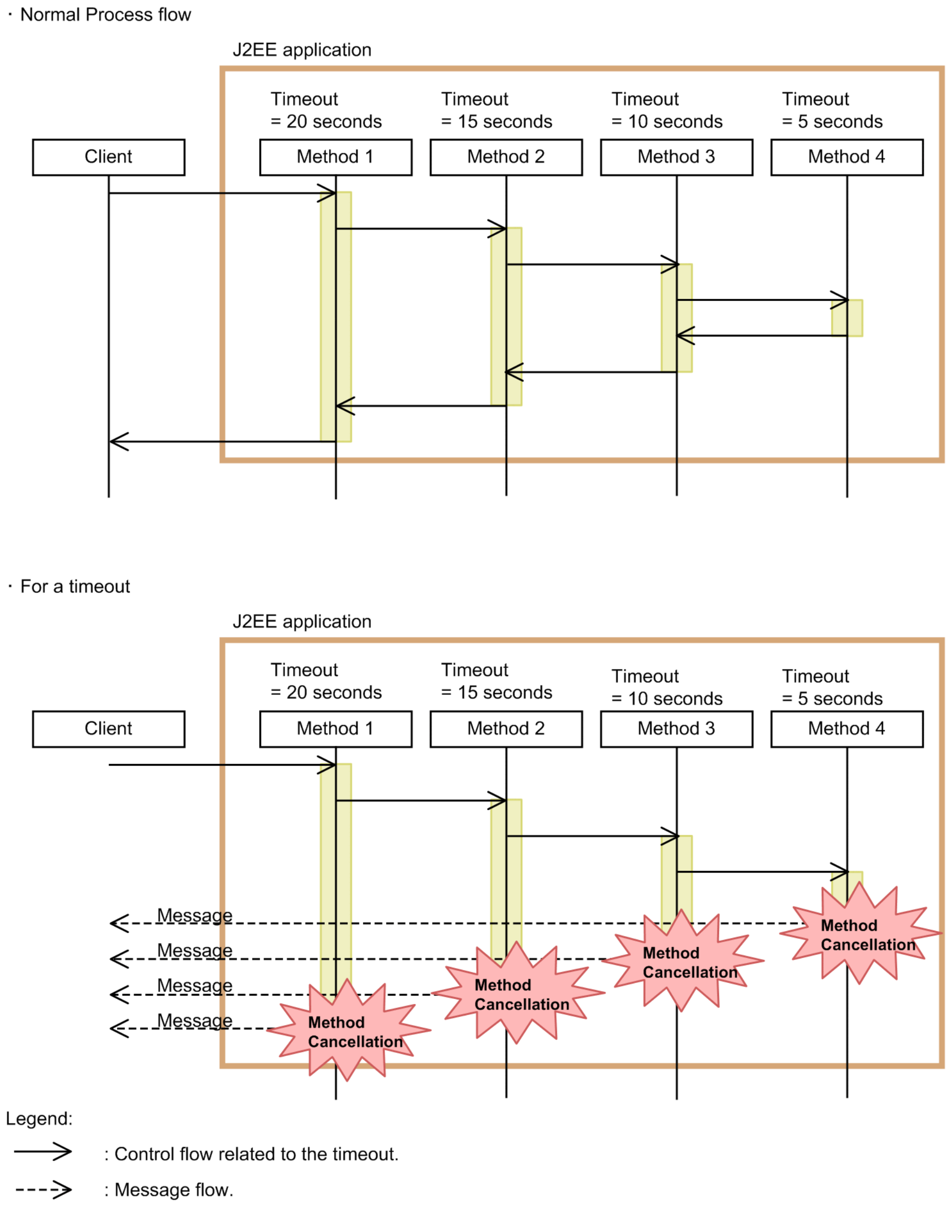
This section explains the settings for method timeout in the J2EE application. The method timeout settings can help you to detect the occurrence of an infinite loop in the business processing on the Web container or EJB container, in Point 6 and Point 8. You can also forcibly cancel (*method cancellation*) the method that detected the timeout. For details about the method cancellation functionality, see *5.3.2 Monitoring the execution time of J2EE application* in the *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*.

The following subsections explain the concept of setting the method timeout:

When method invocation is nested in the J2EE application, specify the order for invoking the method so that a higher value can be set as the timeout value at the invocation source.

An example of setup is given below:

Figure 8–14: Example of setup of method timeout



In this example, a high value is set for the method that is close to the invocation source. As a result, if timeout occurs somewhere in the method, timeout will occur sequentially from the method farthest from the client. The timeout is notified through a message. Depending on the settings, method cancellation can be automatically executed at this time.

If method timeout and method cancellation are set for the methods for executing remote invocation, you need to pay attention to the invocation order. In the method cancellation functionality, the methods that are being remotely invoked are considered to be running in the protected area. If timeout occurs first in the method closer to the invocation source, the method cannot be cancelled since the method is being remotely invoked. As shown in the above example, if you set a high value in the methods closer to the invocation source, since the timeout occurs sequentially from the methods farthest from the invocation source, the method in which timeout has occurred is not being remotely invoked. You can, therefore, definitely cancel the method.

Even if the method timeout and method cancellation are set for the methods for executing local invocation, you can integrate the timed-out method with the cancelled method by canceling sequentially from the methods farthest from the invocation source.

For details about the methods that can be used to set up the timeout value, see *5.3.4 Method Cancellation* in the *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*.

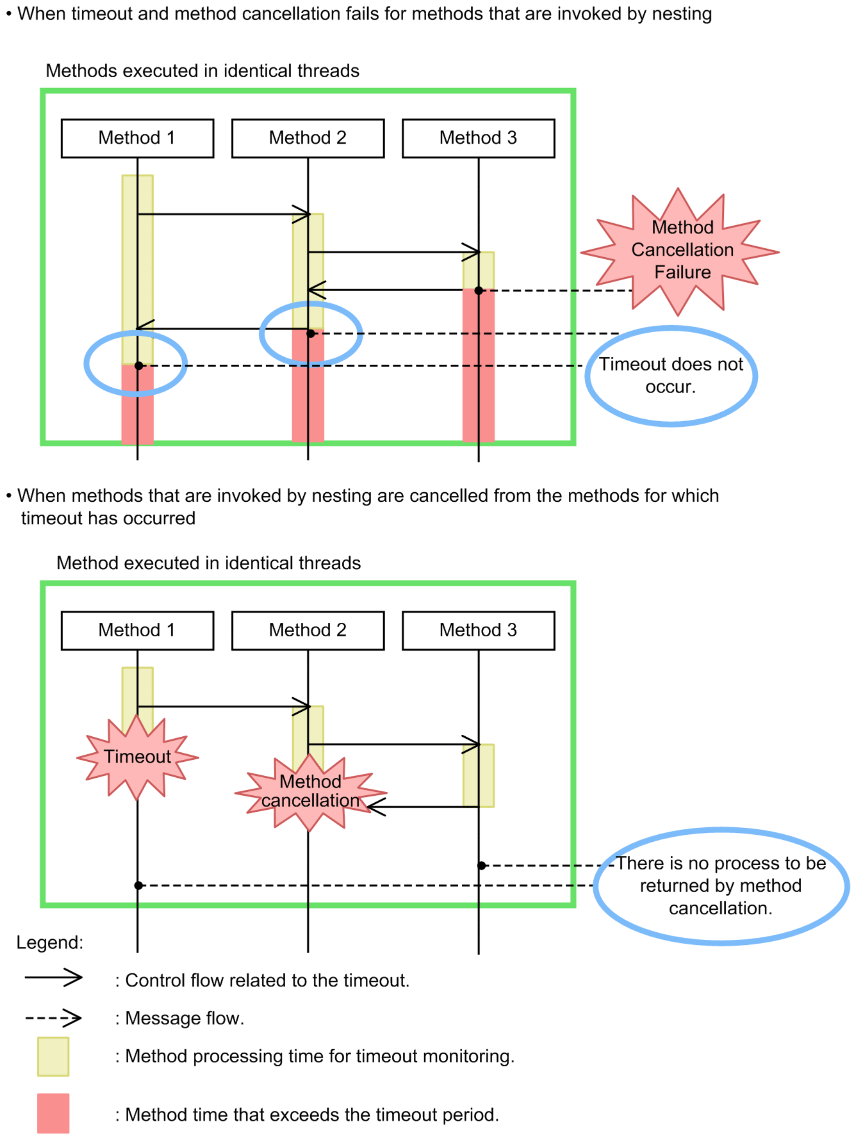
Precautions when a locally invoked method is invoked by nesting

If `app` or `all` are specified in the `ejbserver.rmi.localinvocation.scope` of the `<param-name>` tag of the Easy Setup definition file, a locally invoked method that is invoked by nesting from one method is executed in all the identical threads. At this time, note the following points:

- If timeout and method cancellation of a method invoked by nesting fails, timeout will not occur in the other methods of the identical threads until that method is terminated.
- When method cancellation is executed for the method that has been invoked by nesting from a timed-out method, the container cancels only the method for which method cancellation was executed. The invocation source method in which timeout has occurred will not be cancelled. As a result, even after the method is cancelled, the invoked methods will be terminated sequentially, as in the case of the normal invocation of a nested method. The timeout will also be monitored for such sequentially invoked methods.

The following figure shows the precautions in the case of nested invocation of a locally invoked method in the same thread:

Figure 8–15: Precautions in the case of nested invocation of a locally invoked method in the same thread



• When timeout and method cancellation fails for methods that are invoked by nesting
Methods executed in identical threads
Method 1
Method 2
Method 3
Method Cancellation Failure
Timeout does not occur.
• When methods that are invoked by nesting are cancelled from the methods for which timeout has occurred
Method executed in identical threads
Method 1
Method 2
Method 3
Timeout
Method cancellation
There is no process to be returned by method cancellation.
Legend:
: Control flow related to the timeout.
: Message flow.
: Method processing time for timeout monitoring.
: Method time that exceeds the timeout period.

8.6.8 Tuning parameters for setting the timeout

This section explains how to set up tuning parameters used for timeout settings.

(1) Timeout set in the Web server for receiving requests from the client and sending the data to the client

This is a tuning parameter for setting the timeout at point 1 of *Figure 8-9* or *Figure 8-10*. You can set this parameter only in the case of Web server integration.

Table 8–26: Tuning parameters for the timeout to be set in the Web server for receiving requests from the client and sending the data to the client

Setup item	Location of setup
Timeout for receiving requests from the client and sending data to the client	`Timeout` directive of `httpsd.conf`

(2) Timeout set in the reverse proxy for sending data to the Web container

This is a tuning parameter for setting the timeout at point 2 and point 3 of *Figure 8-9*. The following table describes the tuning parameters for timeout settings in the reverse proxy. You can specify the tuning parameter only in the case of Web server integration.

Specify the items listed in the following table with the Smart Composer functionality. You define the parameters in the Easy Setup definition file.

Table 8–27: Tuning parameters for the timeout to be set in the reverse proxy

Point	Setup item	Setup target	Location of setup (parameter name)#
2	Connection timeout for Web container when sending requests	Logical Web server (`web-server`)	`timeout` key of `manager.web.reverseproxy.mapping` parameter
3	Timeout for sending requests	Logical Web server (`web-server`)	`timeout` key of `manager.web.reverseproxy.mapping` parameter

(3) Timeout set in the reverse proxy for receiving data from the Web container

This is a tuning parameter for setting the timeout at point 4 of *Figure 8-9*.

This parameter is set at the level of the data transfer destination of the reverse proxy. The following table describes the tuning parameters for timeout settings in the reverse proxy.

You specify the items listed in the following table using the Smart Composer functionality and define the parameters in the Easy Setup definition file.

Table 8–28: Tuning parameters for the timeout to be set in the reverse proxy

Setup item	Setup target	Location of setup (parameter name)
Communication timeout of waiting for response data	Logical Web server (`web-server`)	`timeout` key of `manager.web.reverseproxy.mapping` parameter

You can specify this tuning parameter only in the case of Web server integration.

(4) Timeout set in the Web container for receiving data from the reverse proxy or the Web client

This is a tuning parameter for setting the timeout at point 5 of *Figure 8-9*.

You set up the tuning parameter for each J2EE server. The following table describes tuning parameters for the timeout to be set in the Web container.

You specify the items listed in the following table using the Smart Composer functionality and define the parameters in the Easy Setup definition file.

Table 8–29: Tuning parameters for the timeout to be set in the Web container

Setup item	Setup target	Location of setup (parameter name)
Timeout when waiting for response from reverse proxy or Web client	Logical J2EE server (`j2ee-server`)	`webserver.connector.nio_http.receive_timeout`

(5) Timeout set in the Web container for receiving data from the reverse proxy or the Web client

This is a tuning parameter for setting the timeout at the point 13 of *Figure 8-9*.

You set up the tuning parameter for each J2EE server. The following table describes the tuning parameters for the timeout to be set up in the Web container.

You specify the items listed in the following table using the Smart Composer functionality and define the parameters in the Easy Setup definition file.

Table 8–30: Tuning parameters for the timeout to be set in the Web container

Setup item	Setup target	Parameter name
Timeout of response sending process	Logical J2EE server (`j2ee-server`)	`webserver.connector.nio_http.send_timeout`

(6) Timeout set in the EJB client for remotely invoking the Enterprise Bean (RMI-IIOP communication) and for invoking the Naming Service by JNDI

This is a tuning parameter for setting the timeout at point 7 of *Figure 8-9* or *Figure 8-10*.

Set the tuning parameter for each J2EE server, EJB client application, or invocation by API.

The following table describes the tuning parameters (remote invocation by RMI-IIOP communication) for timeout to be set in the EJB client:

Table 8–31: Tuning parameters for the timeout to be set in the EJB client (remote invocation by RMI-IIOP communication)

Units	Method of setup	Setup item	Location of setup
Each J2EE server	Smart Composer functionality	Communication timeout between client and server	Definition file Easy Setup definition file Setup target Logical J2EE server (`j2ee-server`) Parameter name `ejbserver.rmi.request.timeout`
Each EJB client application	Specify the system property to be specified at the time of editing or starting a file		Definition file (in the case of edit file) `usrconf.properties` Parameter name `ejbserver.rmi.request.timeout` key
Each API	API		When setting for each object RequestTimeoutConfig#setRequestTimeout (`java.rmi.Remote` obj, int sec) method# When setting for each thread RequestTimeoutConfig#setRequestTimeout (int sec) method#

#

The name of the package is `com.hitachi.software.ejb.ejbclient`.

The following table describes the tuning parameters (invoking the Naming Service) for timeout to be set in the EJB client:

Table 8–32: Tuning parameters for the timeout to be set in the EJB client (invoking the Naming Service)

Units	Method of setup	Setup item	Location of setup
Each J2EE server	Smart Composer functionality	Period of communication timeout with Naming Service	Definition file Easy Setup definition file Setup target Logical J2EE server (`j2ee-server`) Parameter name `ejbserver.jndi.request.timeout`
Each EJB client application	Specify the system property to be specified at the time of editing or starting a file		Definition file (in the case of edit file) `usrconf.properties` Parameter name `ejbserver.jndi.request.timeout` key

(7) Timeout set up in the EJB client for invoking the Enterprise Bean from CTM

This is a tuning parameter for setting up timeout at point 8 of *Figure 8-9* or *Figure 8-10*.

You set up the tuning parameter for each J2EE server, EJB client application, or each invocation by API.

A value same as that set up in *(6) Timeout set in the EJB client for remotely invoking the Enterprise Bean (RMI-IIOP communication) and for invoking the Naming Service by JNDI* is inherited as the setup value of this timeout.

(8) Timeout set in the EJB container for the database transaction (when DB Connector is used)

This is a tuning parameter for setting the timeout at point 10 of *Figure 8-9* or *Figure 8-10*.

Set the tuning parameter for each J2EE server, Enterprise Bean, interface, method (in the case of CMT), or each invocation by API (in the case of BMT).

The following table describes the tuning parameters for transaction timeout:

Table 8–33: Tuning parameters for transaction timeout

Units	Method of setup	Setup item	Location of setup
Each J2EE server	Smart Composer functionality	Default transaction timeout value of a transaction	Definition file Easy Setup definition file Setup target Logical J2EE server (`j2ee-server`) Parameter name `ejbserver.jta.TransactionManager.defaultTimeOut`
Each Enterprise Bean, interface, method (in the case of CMT)	The `cjsetappprop` command of the server management command	Transaction timeout time	Definition file Session Bean property file, Entity Bean property file, or Message-driven Bean property file Parameter name <ejb-transaction-timeout>
Each API (in the case of BMT)	API		UserTransaction#setTransactionTimeout method[#]

#

The name of the package is javax.transaction.

(9) DB Connector timeout

This is a tuning parameter for setting the timeout at point 11 of *Figure 8-9* or *Figure 8-10*.

Set the tuning parameter for each DB Connector.

The following table describes the tuning parameters for DB Connector.

Table 8–34: Tuning parameters for DB Connector

Units	Method of setup	Setup item	Location of setup
Each DB Connector	Server management commands `cjsetrarprop` or `cjsetresprop`	Timeout in establishing a physical connection	**Definition file** Connector property file **Setup target** DB Connector **Parameter name** `loginTimeout`

Units	Method of setup	Setup item	Location of setup
		Timeout in the request for connection during connection depletion	**Definition file** Connector property file **Setup target** DB Connector **Parameter name** `RequestQueueTimeout`
Each J2EE server	Smart Composer functionality	Timeout in detecting a connection error	**Definition file** Easy Setup definition file **Setup target** Logical J2EE server (`j2ee-server`) **Parameter name** `ejbserver.connectionpool.validation.timeout`#

#

The same property as the timeout value of the connection adjustment functionality.

(10) Database timeout

This is a tuning parameter for setting the timeout at point 12 of *Figure 8-9* or *Figure 8-10*.

The database timeout differs according to the type of database you are using. This section explains how to set the timeout value when accessing the HiRDB, Oracle, SQL Server, or XDM/RD E2 by using the DB Connector.

Reference note

When Oracle is used, you can use the tuning parameters for setting the timeout only when global transaction is used. When local transaction is used, you cannot use the tuning parameters for setting timeout. You can, however, set the query timeout that is set by the method, in the case of both global and local transactions.

(a) Timeout settings in HiRDB

Set the timeout value in the common system definition of the HiRDB server or the environment variable of the HiRDB client. For details, see the manual *HiRDB System Definition* or the manual *HiRDB UAP Development Guide*.

The following table describes the tuning parameters used for setting the timeout in HiRDB:

Table 8–35: Tuning parameters for setting the timeout in HiRDB

Type of timeout	Location of setup	Method of setup (parameter name)	Settings
Unlock waiting timeout	Common system definition of the HiRDB server	`pd_lck_wait_timeout` parameter	You can specify any value you wish.
Response timeout	Environment variable of the HiRDB client	`PDCWAITTIME`	You can specify any value you wish. However, in the case of a global transaction, specify a larger value than the transaction timeout value.
Request interval timeout	Environment variable of the HiRDB client	`PDSWAITTIME`	You can specify any value you wish. However, in the case of a global transaction, specify a larger value than the transaction timeout value.

Reference note

If PDCWAITTIME and PDSWAITTIME are smaller than the transaction timeout value, even if the processing is within the time limit for the transaction, the database processing exceeds the time limit and a timeout occurs.

In this case, the database connection is lost regardless of the transaction being in process, and the transaction manager cannot conclude the transaction.

Also, in the case of a global transaction, the transaction must be recovered because the instruction about transaction conclusion does not reach after the connection is lost.

(b) Timeout settings in Oracle (when global transaction is used)

Set the timeout value in the DISTRIBUTED_LOCK_TIMEOUT parameter of the Oracle server definition.

In addition to the above, the setting of `SesTm` parameter of `XAOpenString` affects the timeout. This parameter cannot be tuned.

(c) Timeout settings in SQL Server

Set the timeout value by executing the parameter or statement in the environment settings option of the SQL Server.

The following table describes the tuning parameters for setting the timeout in the SQL Server.

Table 8–36: Tuning parameters for setting the timeout in the SQL Server

Type of timeout	Location of setup	Method of setup (name of parameter or statement	Settings
Timeout in acquiring memory	Server configuration option	`query wait` parameter	You can specify any value you wish.
Unlock waiting timeout	--	`SET LOCK_TIMEOUT` statement	You can specify any value you wish.

Legend:

--: Not applicable.

(d) Timeout settings in XDM/RD E2

Set the timeout value in the system option definition of the XDM/BASE, the environment variable of the HiRDB client, and the control statement for invoking control space or the control statement for invoking server space of the DB connection server.

The following table describes the tuning parameters for setting the timeout in XDM/RD E2:

Table 8–37: Tuning parameters for setting the timeout in XDM/RD E2

Type of timeout	Location of setup	Method of setup (parameter name)	Settings
Unlock waiting timeout	System option definition of XDM/BASE	`TIMER`	You can specify any value you wish.#1
CPU timeout during SQL execution	The control statement for invoking control space or control statement	`SQLCTIME`	You can specify any value you wish. However, in the case of a global transaction,

Type of timeout	Location of setup	Method of setup (parameter name)	Settings
	for invoking server space of the DB connection server		specify a larger value than the transaction timeout value.#2
SQL execution timeout	The control statement for invoking control space or control statement for invoking server space of the DB connection server	`SQLETIME`	You can specify any value you wish. However, in the case of a global transaction, specify a larger value than the transaction timeout value.#2
Transaction timeout	The control statement for invoking control space or control statement for invoking server space of the DB connection server	`SVETIME`	You can specify any value you wish. However, in the case of a global transaction, specify a larger value than the transaction timeout value.#2
Response timeout	HiRDB client environment variable	`PDCWAITTIME`	You can specify any value you wish. However, in the case of a global transaction, specify a larger value than the transaction timeout value.#3

#1

For details, see the manual *VOS3 Data Management System XDM E2 Node System Definition (XDM/BASE, SD, or TM2)*.

#2

For details, see the manual *VOS3 Database Connection Server*.

#3

For details, see the manual *HiRDB XDM/RD E2 Connection Functionality*.

(11) Method timeout in J2EE application

This is a tuning parameter for setting the timeout at point 6 and point 9 of *Figure 8-9* or *Figure 8-10*.

Set the timeout value as an application attribute when you want to set the timeout for each method in the Web application or the Enterprise Bean. Set the operations in the case of timeout as an application attribute as well. You specify these items using the server management command (`cjsetappprop`).

The following table describes the tuning parameters for setting the timeout in the method execution time. The location of setup differs for each point.

Table 8–38: Tuning parameters for setting the timeout for the method execution time

Points to set	Type of timeout and operations in the case of timeout	Location of setup
6	Request processing method for the filter, servlet or JSP	Definition file Servlet property file Parameter name `<method-observation-timeout>`
9	Request processing method for the Enterprise Bean	Definition file Session Bean property file, Entity Bean property file, or Message-driven Bean property file Parameter name `<ejb-method-observation-timeout>`
6 and 9	Operations for each application when timeout occurs	Definition file Application property file Parameter name `<method-observation-recovery-mode>`

8.7 Optimizing the operations of the Web application

This chapter explains how to tune the performance of the Web application. Determine the tuning in the case of a Web front-end system.

The following three types of tuning methods are explained below:

- Separate the deployment of the static contents and the Web application
- Caching static contents

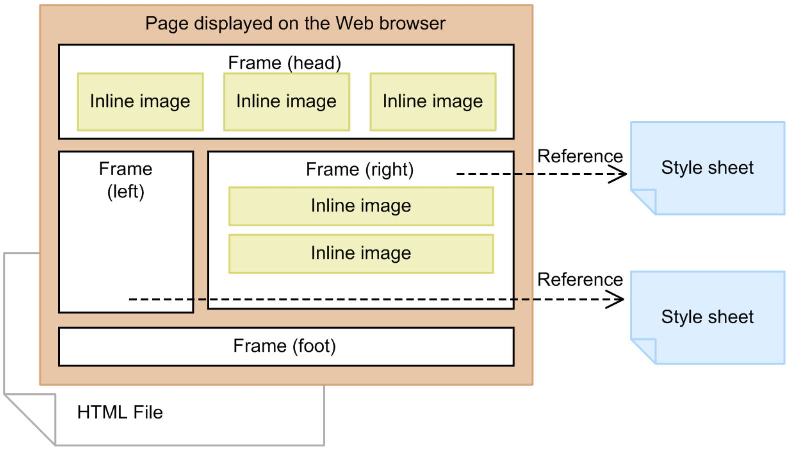
8.7.1 Separating the deployment of the static contents and the Web application

From the files that are used for responding to the requests from the client such as HTML file and image file, the contents that remain the same and do not change according to the contents of the request are called *static contents*. On the other hand, the contents that are dynamically generated according to the requests from the client, such as servlets and JSPs, are called *dynamic contents*.

The following section explains how to enhance the performance by separating and deploying the static contents and Web applications that are the dynamic contents.

Each example of setup is explained on the basis of an example of displaying an HTML page consisting of frames and inline images, as shown in the following figure, in the web browser:

Figure 8–16: Example of HTML page consisting of static and dynamic contents



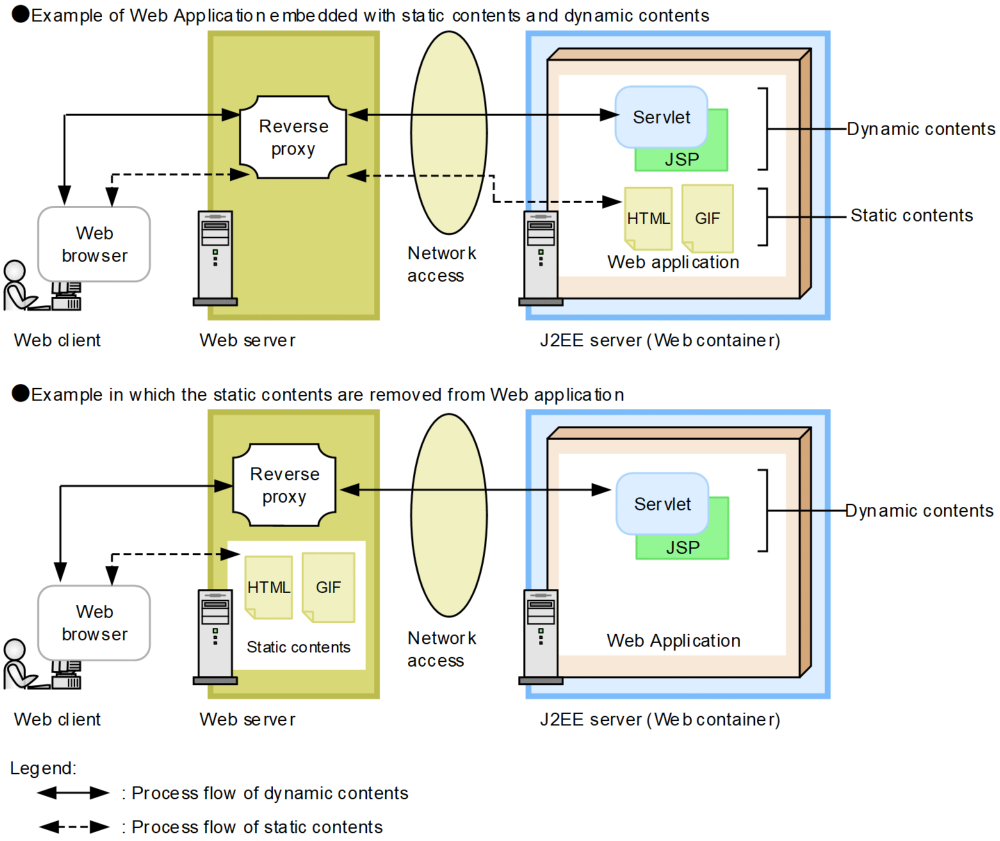
In the above configuration, the following files form the static contents that are not generated dynamically:

- Style sheet (such as a CSS file)
- Inline image (image file)
- HTML file that defines frames

When embedding static content into a Web application, there is constant access between the Web server and the Web container even when transferring static content that does not need to be processed by the Web container. In the case of image files, the processing time takes longer due to the large data size.

Hitachi recommends that the static contents be segregated from the Web application and deployed on the Web server. This helps to reduce the frequency of network access and the size of exchanged data that in turn will enhance the performance. The following figure shows an image of the processing of the static contents and the Web application:

Figure 8–17: Image of static contents and Web application processing


●Example of Web Application embedded with static contents and dynamic contents
Reverse proxy
Servlet
JSP
Dynamic contents
HTML
GIF
Static contents
Web browser
Network access
Web application
Web client
Web server
J2EE server (Web container)
●Example in which the static contents are removed from Web application
Reverse proxy
Servlet
JSP
Dynamic contents
HTML
GIF
Static contents
Web browser
Network access
Web Application
Web client
Web server
J2EE server (Web container)
Legend:
: Process flow of dynamic contents
: Process flow of static contents

The method of deploying the static contents that have been segregated from the Web application is explained below using the HTML page configuration that is shown in Figure 8-16.

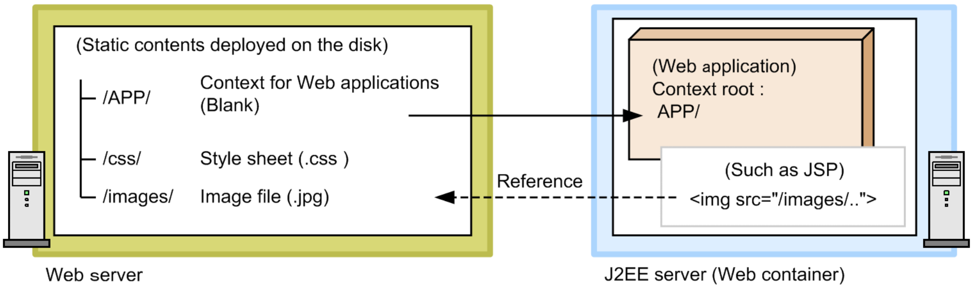
In case of Figure 8-16, the performance is enhanced by deploying the following static contents on the Web server:

Contents deployed on the Web server

- Style sheet (such as a CSS file)
- Inline image (image file)
- HTML file that defines frames

The following figure shows an example of deployment:

Figure 8–18: Deploying the static contents on the Web server (in the case of Web server integration)



/APP/ is mapped to the context root of Web application.

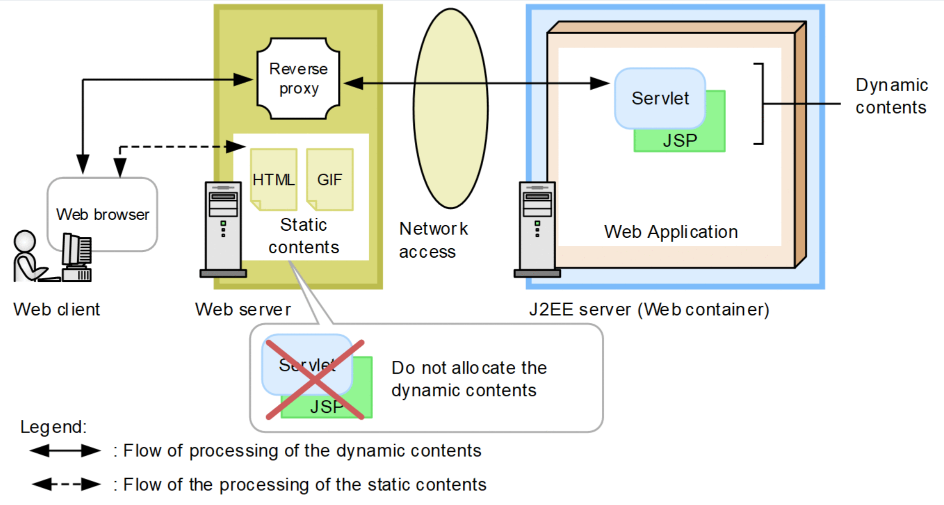
Define the above mapping as shown below. For details, see the description of the `ProxyPass` directive in *6.2.6(15)* in the *uCosminexus Application Server HTTP Server User Guide*.

Definition of reverse proxy

```
ProxyPass /APP/    http://myj2eeserver/APP/
```

Important note

During the Web server integration, instead of allocating all Web application files to the Web server, allocate only the static contents that are directly accessed from the client.


Reverse proxy
HTML
GIF
Static contents
Network access
Servlet
JSP
Web Application
Dynamic contents
Web browser
Web client
Web server
J2EE server (Web container)
Servlet
JSP
Do not allocate the dynamic contents
Legend:
: Flow of processing of the dynamic contents
: Flow of the processing of the static contents

The reverse proxy might incorrectly identify a request for dynamic content as a request for static content if it cannot correctly read the extension or path information of the request. In this case, the reverse proxy sends the content on the Web server directly to the client without reassigning the request to processing in the J2EE server. If dynamic contents, such as servlets and JSPs, are allocated to a Web server, and if a request for dynamic contents is handled as a request for static contents, the source of the contents, such as the class file entities and JSPs, might be sent as a response to the request source client.

8.7.2 Caching static contents

In a Web container, the static contents can be saved in the memory (cache). Once the static contents are accessed, they are saved in the memory. This helps to reduce the frequency of subsequent access to the file system and enhance the response speed.

The method of tuning required for caching the static contents is explained below.

The following items can be tuned:

- Selecting whether to cache the static contents
- Maximum size of the memory used for the static contents cache
- Maximum file size of the static contents for caching

Each of these items is explained below. You can set the above items for each Web container and Web application. The Web application settings override the Web container settings. If you want to specify a default value for the entire J2EE server, specify the value for each Web container and if you want to specify detailed settings, specify the value for each Web application.

> **Tip**
>
> **Relationship with the estimation of the memory size**
>
> In the static contents cache, the memory is used to enhance the response speed. When this functionality is used, it is important to tune the size according to the memory size that can be used in the server.
>
> Set the size of the memory used for the static contents cache for each Web application. The sum total of all memory sizes specified for each Web application is the maximum memory size that can be used for the caching the static contents in the entire J2EE server. To use this functionality, therefore, add the total value of the memory size specified for each Web application, for estimating the memory size required in the J2EE server.

(1) Selecting whether to cache the static contents

The static contents caching is a functionality that is not used by default. You need to change the parameters if you want to use this functionality.

The use of static contents cache can be set for each Web container and Web application. The Web application settings override the Web container settings. If, however, the settings are forcibly disabled in the Web container, the Web application settings will also be disabled. The settings can be forcibly disabled in the following cases:

- To check the difference between the usage of memory when static contents cache is enabled and when the static contents cache is disabled
- To temporarily disable the static contents cache when the settings for each Web application are saved

(2) Maximum size of the memory used for the static contents cache

To cache the static contents, you can set the size of the memory to be used for each Web application. If the total size of the cache becomes more than the size specified in the respective Web application, the time that is not accessed, is deleted from the largest cache. Deletion continues until the total cache size falls below the size that was set.

The guidelines for setting the memory size are explained below:

Guidelines for setting the memory size

- Set the maximum value of the total size of the static contents contained in the Web application.
- The optimum size to be specified differs depending on the contents of the Web application. After setting the size, estimate the response speed of the request for the static contents and find the optimum size of the cache that will give the best response speed.

(3) Maximum file size of the static contents for caching

You can set the maximum file size of the static contents to be saved in the static contents. When the maximum size has been set, the contents of the file that exceed the maximum size are not saved in the cache, but are retrieved from the file system every time they are used.

The guidelines for setting the file size are explained below:

Guidelines for setting the file size

- From the static contents contained in the Web application, specify a file size that is no more than the maximum file size.
- Specify a value keeping in mind that the cache of static contents that does not have a high access frequency is not destroyed due to the cache of large size static contents.

(4) Check the operation status of the static contents

You can check the operation status of the static contents from details in the message KDJE39234-I that is output when the Web application is stopped. Tune all the parameters as required, according to the total size of the static contents and the total number of contents saved in the cache that is output to the above message.

8.7.3 Tuning parameters for optimizing the operations of the Web application

This section explains how to set up the tuning parameters used for optimizing the operations of a Web application.

(1) Tuning parameter for separating the deployment of the static contents and the Web application

Specify the separation of the deployment of the static contents and the Web application as the parameter of the file that defines the operations of the Web server. The setup locations, files and parameters differ depending on the type of Web server used.

The following table explains the method and location of setup:

Table 8–39: Tuning parameters for separating the deployment of the static contents and the Web application

Method of setup	Location of setup
Smart Composer functionality	Definition file Easy Setup definition file Setup target Logical Web server (`web-server`) Parameter name `manager.web.reverseproxy.mapping`

(2) Tuning parameters for caching static contents

The tuning parameters for cache of static contents are explained below. These tuning parameters are set for each Web container or Web application.

The following table describes how to set up the tuning parameters for each Web container. You specify these items using the Smart Composer functionality.

Table 8–40: Tuning parameter for caching static contents (items to be set for each Web container)

Setup items	Locations of setup
Select whether static contents cache is to be used	Definition file Easy Setup definition file Setup target Logical J2EE server (`j2ee-server`) Parameter name `webserver.static_content.cache.enabled`
Setup of maximum memory size for each Web application	Definition file Easy Setup definition file Setup target Logical J2EE server (`j2ee-server`) Parameter name `webserver.static_content.cache.size`
Setup of maximum file size of the static contents for cache	Definition file Easy Setup definition file Setup target Logical J2EE server (`j2ee-server`) Parameter name `webserver.static_content.cache.filesize.threshold`

The tuning parameters to be set for each Web application are described below. Set the items in the Web application either directly by editing `web.xml` or using the server management commands. Edit `web.xml` for setting the items in the Web application before deployment. You use the server management command (`cjsetappprop`) for setting the items in the Web application after deployment.

The following table describes the settings:

Table 8–41: Tuning parameters for caching of static contents (items to be set for each Web application)

Setup items	Settings#
Select whether to use the cache of static contents	*param-name* tag `com.hitachi.software.web.static_content.cache.enabled` *param-value* tag (Setup value)
Setup of maximum size of memory for each Web application	*param-name* tag `com.hitachi.software.web.static_content.cache.size` *param-value* tag (Setup value)
Setup of maximum file size of the static contents for caching	*param-name* tag `com.hitachi.software.web.static_content.cache.filesize.threshold` *param-value* tag (Setup value)

Note

For details about the values that can be specified in (setup value), see *2.19.2 Definitions in DD (Settings for each Web application)* in the *uCosminexus Application Server Web Container Functionality Guide*.

#

For directly editing `web.xml`, add `<context-param>` tag within the `<web-app>` tag and add the `<param-name>` and `<param-value>` tags within the `<context-param>` tag.

To use the server management commands, add `<context-param>` tag within the `<hitachi-war-property>` tag of the `WAR property` file and add `<context-param>` and `<param-value>` tags within the `<context-param>` tag.

8.8 Optimizing the operation of CTM

This section explains how to tune the performance of a system that uses CTM. Determine in for a back-end system using CTM.

Following four types of tuning procedures are described here:

- Tuning the monitoring interval of the operation state of CTM domain managers and CTM daemons
- Tuning the monitoring interval of the load status
- Setting up a timeout lock for a CTM daemon
- Setting up a priority order for the requests distributed with CTM

8.8.1 Tuning the monitoring interval of the operation state of CTM domain managers and CTM daemons

A communication process is executed periodically between the multiple CTM domain managers present in a system, and the multiple CTM daemons within a CTM domain, to monitor the mutual operation state.

This subsection describes the concept of tuning the respective communication process intervals.

(1) Tuning the monitoring interval of the operation state between CTM domain managers

The CTM domain managers periodically exchange information about the CTM daemons present in the mutual hosts. Based on this information, the requests received in the local host are distributed properly in the CTM daemon of the other host.

When exchanging information, the mutual operation state of the CTM domain managers is also checked concurrently. If the CTM domain manager of the other host is not running, requests will not be distributed to that host. While exchanging information, if a CTM domain manager does not receive a response until the lapse of a time period derived by multiplying with a fixed coefficient of time, the other CTM domain manager is assumed to be in a stopped state.

The setup location for the interval of exchanging information differs depending on whether the other CTM domain manager is present within the same network segment, or in another network segment. The default value of the coefficient used to judge the operation state is 2. For example, when checking the operation (online) information of a CTM domain manager present within the same network segment of a system built using Management Server, if there is no response for a time period derived by multiplying 2 with the dispatch interval of the CTM domain configuration information, the CTM domain manager is assumed to be in a stopped state. If the dispatch interval of the CTM domain configuration information is 60 seconds, the CTM domain manager will be assumed to be in a stopped state if there is no response for 120 seconds.

You can change this coefficient value to optimize the communication process.

Determine an appropriate value to be specified in the coefficient, after considering the load of the processing generated as a result of communication. If necessary, determine the standard dispatch interval as well. If you reduce the coefficient, the interval will be shortened, and you can immediately detect if the other CTM domain manager is in stopped state. This will help in preventing the dispatch of requests to the host in which the CTM daemon has stopped running. However, if the interval becomes too short, the communication processing will increase, thus increasing the communication overload.

(2) Tuning the monitoring interval of the operation state between CTM daemons

The CTM daemons mutually distribute requests based on the information exchanged between the CTM domain managers.

If no response is received from the CTM daemon to which requests are distributed, the CTM daemon distributing the requests assumes that the other CTM daemon is in stopped state, and does not distribute any more requests.

By default, a CTM daemon waits for `180 seconds` to receive a response. If no response is received within `180 seconds`, it is assumed that the other CTM daemon is in stopped state. By changing this value, the unnecessary wait time can be shortened.

You set up an appropriate value after considering the size of the data to be sent as a request. By shortening the interval, any trouble in the other host can be detected promptly, and therefore, enabling you to disconnect from the system at a stage when the effect of the trouble is minimum. However, if the interval is too short, timeouts may occur while a large-sized data is being transferred.

8.8.2 Tuning the monitoring interval of the load status

The multiple CTM daemons present within a CTM domain monitor the loading information of the respective schedule queues. The monitoring result is used during distribution of requests between CTM daemons.

You can tune the interval of monitoring the load status. The default value is `10 seconds`.

By shortening the interval of monitoring the load status, you can distribute the requests according to the corresponding status. However, if the interval becomes too short, the communication will occur frequently, thus increasing the load.

Note that if you set up the interval of load status monitoring to `0`, the load status during startup of the J2EE server will be collected only once, and that value will be used repeatedly.

8.8.3 Setting up a timeout lock for CTM daemon

If an error occurs in the J2EE server corresponding to the CTM daemon, timeouts will occur in the requests sent by the CTM daemon. If you continue the operation in this state, the CTM daemon will continue to send requests to the J2EE server in which the trouble occurred, and therefore, the timeout will occur in the request.

To handle this, you can set up the timeout lock in the CTM daemon. The *timeout lock* is a functionality to lock the schedule queue of a CTM daemon when request timeouts occur more frequently than the timeout frequency specified for a fixed period of time. Thus, no more requests are received by the CTM daemon in which the trouble occurred, and the requests are received by other CTM daemons. This leads to the distribution of requests to a J2EE server running normally.

8.8.4 Setting up a priority order for the requests distributed with CTM

You can set up a priority order for the requests controlled with CTM. By setting up a high priority order for the requests that must be executed promptly, processing can be done fast without the accumulation of requests in the schedule queue.

The priority order of requests can be set up in the EJB client application or J2EE server that sends requests to CTM. The high-priority requests sent from the EJB client application or J2EE server are processed before the requests sent by other clients that are saved in the schedule queue.

8.8.5 Tuning parameters for optimizing the operation of CTM

This section explains how to set up the tuning parameters used to optimize the operation of CTM.

(1) Tuning parameters for setting up the monitoring interval of the operation state of CTM domain managers and CTM daemons

This subsection explains the parameters for tuning the monitoring interval of the operation state of CTM domain manager.

You set up the items listed in the following table in the Easy Setup definition file. Afterward re-build the system using the Smart Composer functionality.

Note that the monitoring interval is the product of the dispatch interval and the coefficient.

Table 8–42: Parameters for tuning the monitoring interval of the operation state of CTM domain manager

Target	Setup item	Setup target	Location of setup (parameter name)
CTM domain managers present in the same network segment	Dispatch interval	Logical CTM domain manager (`ctm-domain-manager`)	`cdm.SendInterval`
	Coefficient	Logical CTM domain manager (`ctm-domain-manager`)	`cdm.AliveCheckCount`
CTM domain managers present in a different network segment	Dispatch interval	Logical CTM domain manager (`ctm-domain-manager`)	`cdm.SendHostInterval`
	Coefficient	Logical CTM domain manager (`ctm-domain-manager`)	`cdm.AliveCheckCount`

The parameter for tuning the monitoring interval of the operating status of CTM daemons is as follows:

You set up the items listed in the following table in the Easy Setup definition file. Afterward re-build the system using the Smart Composer functionality.

Table 8–43: Parameter for tuning the monitoring interval of the operation state of CTM daemon

Setup item	Setup target	Location of setup (parameter name)
Timeout during transfer between CTM daemons	Logical CTM	`ctm.DCSendTimeOut`

(2) Tuning parameters for setting up the monitoring interval of the load status

This subsection explains the parameter for tuning the monitoring interval of the load status.

You set up the items listed in the following table in the Easy Setup definition file. Afterward re-build the system using the Smart Composer functionality.

Table 8–44: Parameter for tuning the monitoring interval of the load information

Setup item	Setup target	Location of setup (parameter name)
Timeout during transfer between CTM daemons	Logical CTM	`ctm.LoadCheckInterval`

(3) Tuning parameters for setting up the timeout lock for CTM daemon

The timeout lock is executed by setting up the timeout occurrence frequency and the monitoring interval.

This subsection explains the parameters for tuning the timeout lock of a CTM daemon.

You set up the items listed in the following table in the Easy Setup definition file. Afterward re-build the system using the Smart Composer functionality.

Table 8–45: Parameters for tuning the timeout lock of a CTM daemon

Setup item	Setup target	Location of setup (parameter name)
Timeout occurrence frequency	Logical CTM	`ctm.RequestCount`
Monitoring interval	Logical CTM	`ctm.RequestInterval`

(4) Tuning parameters for setting up a priority order for the requests distributed with CTM

The setting of the priority order of requests distributed with CTM is different for an EJB client application and for a J2EE server. Moreover, for a J2EE server, the setup location differs depending on how the system is built. The following table describes the tuning parameters for setting up the priority order of requests distributed with CTM:

Table 8–46: Tuning parameters for setting up the priority order of requests distributed with CTM

Setup unit	Method of setup	Location of setup
EJB client application	Specify the system property to be specified while editing the file or starting the EJB client application	Definition file (when editing the file) `usrconf.properties` Parameter name `ejbserver.client.ctm.RequestPriority` key
J2EE server	Smart Composer functionality	Definition file Easy Setup definition file Setup target Logical J2EE server (`j2ee-server`) Parameter name `ejbserver.client.ctm.RequestPriority`

9 Performance Tuning (Batch Application Execution Platform)

This chapter explains how to setup the performance tuning of systems for executing batch applications.

You can optimize the operating environment through the performance tuning, and can maximize the performance of systems.

For determining the performance tuning of a J2EE application execution platform, see *8. Performance Tuning (J2EE Application Execution Platform)*.

9.1 Points to be considered for performance tuning

This section explains the points to be considered for the performance tuning of a batch application execution platform.

9.1.1 Viewpoints of performance tuning

Tune the performance of the batch application execution platform from the following viewpoints:

- **Optimizing the database access method**
- **Setting up timeouts**
- **Setting up threshold value for the memory usage causing a full garbage collection**

Explanation regarding these points is as follows:

(1) Optimizing the database access method

The purpose of optimizing the database access method is to reduce overhead during database access by pooling the connections and statements that will require more time for generation.

The performance tuning enhances the throughput by optimizing the database access through an effective use of the following functionality:

- Connection pooling
- Statement pooling (pooling of `PreparedStatement` and `CallableStatement`)

The database access method can be optimized when you are using a DB Connector to establish a connection with the database.

(2) Setting up timeouts

The purpose of setting up timeouts is to detect trouble in the system and release resources whenever required to avoid a delay in responding to requests.

You can set up the following types of timeouts:

- Timeout when invoking an Enterprise Bean
- Timeout of a transaction
- Timeout of a database

(3) Setting up a threshold value for the memory usage causing a full garbage collection

The threshold value used to control a full garbage collection is set up when the same resources are used for online processing and for batch processing. The purpose of this is to avoid the interruption of the online processing due to full garbage collection in the batch server.

By controlling the execution of full garbage collection, you can properly execute full garbage collection when the resources are not excluded.

9.1.2 Tuning procedure

The performance tuning is a task that involves detecting the best settings for system performance. For already built environments, revising parameters, and identifying and removing bottlenecks during the actual processing can continuously enhance the performance.

During tuning, first of all, determine target values. The next task is to measure the throughput when the initial value is set up for each parameter. Keep adjusting each parameter to detect the optimum value closest to the target value.

When tuning, you can use the monitoring tools provided as accessories with the OS to measure the CPU usage rate. You can check the statistical information of Application Server related to JavaVM using the statistics collection functionality. For details about how to check, see *3. Monitoring the Statistics (Statistics Collection Functionality)* in the manual *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*.

Note that when the CPU usage rate is saturated at a level considerably lower than 100%, bottlenecks, such as I/O processing and exclusion processing might be present in the system. Identify and remove the bottlenecks, and then re-execute the performance tuning. In Application Server, you can use performance analysis trace to identify the bottlenecks of the system. For details about the functionality of performance analysis trace and for details about how to use the trace file acquired using performance analysis trace, see *7. Performance Analysis by Using Trace Based Performance Analysis* in the manual *uCosminexus Application Server Maintenance and Migration Guide*.

9.1.3 Tuning items

The following table describes the tuning items of the batch application execution platform:

Table 9–1: Tuning items of the batch application execution platform

Tuning item	Available functionality	Reference
Optimizing the database access method	Connection pooling#1	*8.5.1*#2
	Statement pooling#1	*8.5.2*#2
Setting up timeouts	Setup of timeouts during invocation of an Enterprise Bean	*8.6.3*#2#3
	Setup of transaction timeouts	*9.3.2*
	Setup of timeouts in a database	*8.6.6*#2
Controlling full garbage collection#1	Setup of threshold values	*9.4*

#1

This functionality is available when you are using a DB Connector.

#2

See the explanation about a J2EE application execution platform. When reading the description, please substitute *J2EE server* for *batch server*. Similarly, substitute *J2EE application* for *batch application*.

#3

When invoking Enterprise Beans, you can set up timeouts for the items same as for the back-end systems of the J2EE application execution platform.

9.2 Tuning procedure

This section explains how to execute tuning. The tuning procedure differs depending on the type of setup target.

9.2.1 Tuning a batch server

Use the Easy Setup definition file of the Smart Composer functionality to tune a batch server. In the Easy Setup definition file, specify the type of the logical server (J2EE server[#]) you want to set up in `<logical-server-type>` under the `<configuration>` tag, and specify the parameter name and its value under the `<param>` tag. For details about the Easy Setup definition file, see *4.3 Easy Setup definition file* in the manual *uCosminexus Application Server Definition Reference Guide*.

#

The Smart Composer functionality handles a batch server as a J2EE server.

9.2.2 Tuning a resource

Use the server management commands for tuning a resource.

If you are using the server management commands, edit the `Connector property` file. For details about the `Connector property` file, see *4.1 HITACHI Connector Property file* in the manual *uCosminexus Application Server Application and Resource Definition Reference Guide*.

9.3 Setting up timeouts

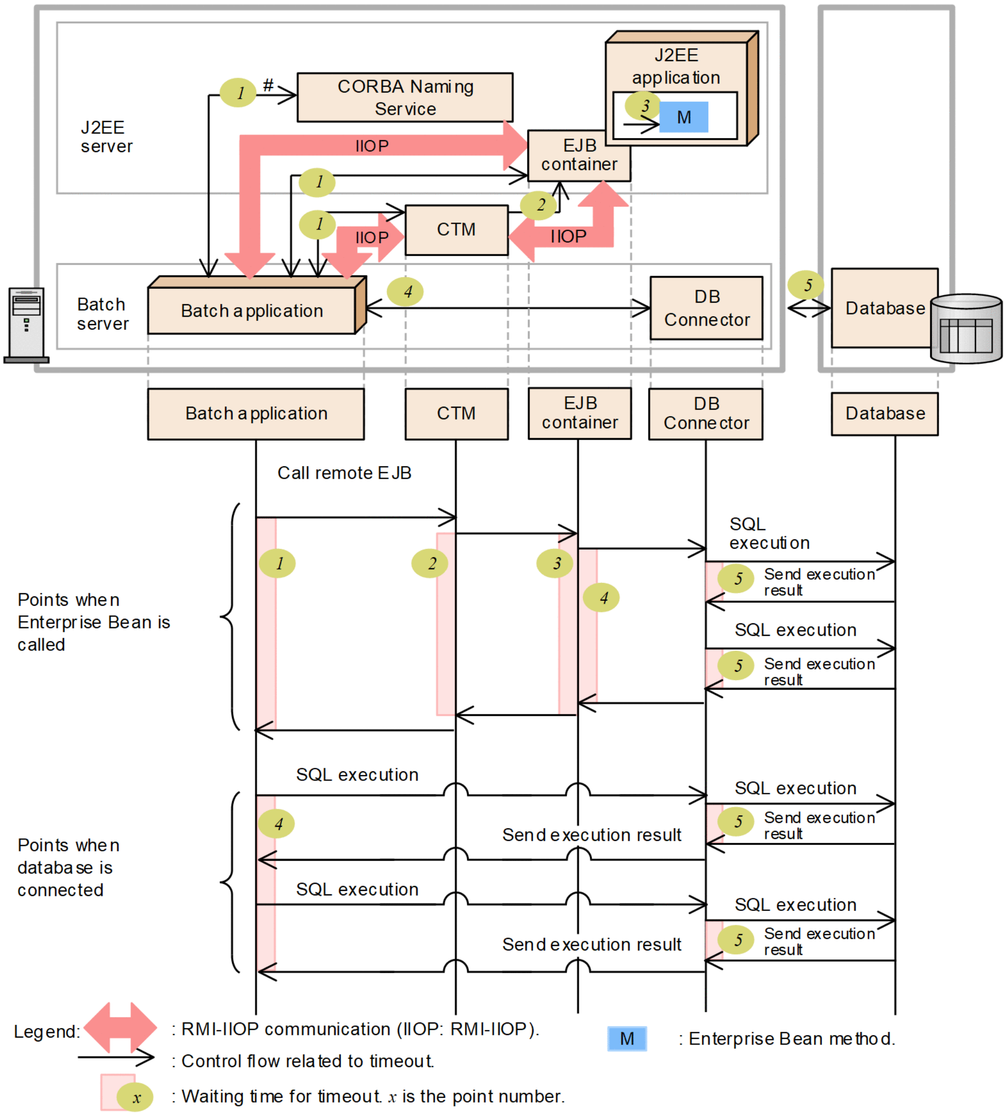
In Application Server systems, you can set up timeouts at several points to prevent states in which no response is received for a request when trouble occurs.

This section describes the points where you can set up timeouts in the entire system, and the guidelines for setting up timeouts.

9.3.1 Points where you can set up a timeout

In the systems for executing batch applications, you can set up timeouts at the points shown in the following figure:

Figure 9–1: Points where a timeout can be set up



J2EE server
CORBA Naming Service
J2EE application
M
EJB container
IIOP
CTM
Batch server
Batch application
DB Connector
Database
1
2
3
4
5
#
Call remote EJB
SQL execution
Send execution result
Points when Enterprise Bean is called
Points when database is connected
Legend:
: RMI-IIOP communication (IIOP: RMI-IIOP).
: Control flow related to timeout.
: Waiting time for timeout. x is the point number.
: Enterprise Bean method.

The timeout in CORBA Naming Service is the time from the issue of an inquiry from the EJB client of Web container to the CORBA Naming Service as per the JNDI.

The timeout specified at each point has a specific use that is described in the table below:

Table 9–2: Purpose of the timeout set up at each point and the default timeout settings

Point	Type of timeout	Primary usage
1	Timeout set up in the batch server for remotely invoking the Enterprise Bean (RMI-IIOP communication) and for invoking the JNDI Naming Service	Detecting failures in the business processing (such as infinite loop and deadlocks) of the batch server or the communication path
2#	Timeout set up in the batch server for invoking the Enterprise Bean from CTM	Detecting failures in the business processing (such as infinite loop and deadlocks) of the batch server or the communication path
3	Timeout set up for the method execution time in the EJB that accesses the Enterprise Bean through invocation	Detecting failures in the business processing (such as infinite loop and deadlock) of the J2EE server
4	Timeout set up in the batch server for the database transaction	Detecting failures in database server (such as, server is down or a deadlock has occurred) or preventing the extended exclusive use of resources
5	Database timeout	Detecting failures in database server (such as server is down or a deadlock has occurred) or preventing the extended exclusive use of resources

#

This point exists only when you are using CTM. For a configuration in which CTM is not used, the range of point 2 extends from the time of execution of remote invocation of the EJB from the batch server to the EJB container, until the dispatch of execution result from the EJB container to the batch server.

The basic guidelines for setting up the above timeouts are as follows:

- The general rule for setting up a timeout value is the closer the point is to the source of invocation (batch server), the higher is the timeout value. Therefore, Hitachi recommends that you use the following relationship for setting up a timeout:
    - Point 1 = Point 2 > Point 3 > Point 4 > Point 5
- When setting up a timeout value for points 1, 4, and 5, first check the amount of time normally taken by the invocation process, then calculate and set up a timeout value for each invocation process (business).

Points 1 to 5 can be divided into following two categories depending on their location in the system:

- Points (1 to 3) that must be considered during invocation of the Enterprise Bean

  The items for which timeouts are to be set up at these points are same as the items that can be set up in back-end systems of the J2EE application execution platform. For details, see *8.6.3 Setting a timeout in the back-end system*.
- Points (4 and 5) that must be considered when establishing a connection with the database

  These points must be considered by classifying them furthermore into transaction timeout and database timeout.

  For details about the transaction timeout, see *9.3.2 Setting up the transaction timeout*.

  The items for which the database timeouts are to be set up are same as the items that can be set up in the back-end systems of the J2EE application execution platform. For details, see *8.6.6 Setting the database timeout*.

For details about the settings for each point, see *9.3.3 Tuning parameters for setting up the timeout* for the batch application execution platform and *8.6.8 Tuning parameters for setting the timeout* for the J2EE application execution platform.

Reference note

The default value of each point is as follows:

Point	Default value
1	Not set up. Continues to wait for a response.
2	A value same as point 1 is automatically inherited and set up when the Enterprise Bean is invoked.
3	Not set up. A timeout does not occur.
4	180 seconds
5	Differs according to the type of the database and the location of setup of the timeout# For HiRDB Unlock waiting timeout: 180 seconds Response timeout: 0 seconds (The HiRDB client continues to wait until a response is received from the HiRDB server.) Request interval timeout: 600 seconds For an SQL Server Timeout while waiting to acquire memory: -1 (For details about the operations when -1 is specified, see the SQL Server documentation) Unlock waiting timeout: -1 (Continues to wait until the lock is released) For XDM/RD E2 Unlock waiting timeout: None (The timeout period is not monitored) CPU timeout during SQL execution: 10 seconds SQL execution timeout: 0 seconds (The timeout period is not monitored) Transaction timeout: 600 seconds Response timeout: 0 seconds (The HiRDB client continues to wait until a response is received from the XDM/RD E2 server.)

#

In Oracle, there is no default value for the unlock waiting timeout.

9.3.2 Setting up the transaction timeout

This section explains the settings for a transaction timeout. Set up a transaction timeout in transactions with EIS, such as database systems. The transaction timeout set up when accessing a database using a DB Connector is as follows:

When setting up the transaction timeout, of all the timeout values of the entire system, you must consider the transactions between the batch server and database.

When a transaction timeout occurs, the Application Server executes the following processing:

- The active transactions are rolled back.
- The connections participating in the transaction are closed, and then deleted from the connection pool.

Tip

The transaction management method used in this case is BMT. You can specify the transaction timeout either in `usrconf.properties` or in the JTA API (`javax.transaction.UserTransaction#setTransactionTimeout` method).

The definition of the `usrconf.properties` affects the entire process. The timeout value specified in API affects only the transactions that issued the API. The API specifications override the definitions in `usrconf.properties`.

> Hitachi, therefore, recommends that you define the standard values to be set up in the entire process, in the `usrconf.properties` and use the appropriate API to set up the detailed values as per the business to be invoked.

When a transaction timeout occurs, the exception is not reported to the batch application. However, the message KDJE31002-W is output to the log file and the batch server console. After a transaction timeout occurs, an exception is reported when you attempt to use the JTA interface or the JDBC interface using the relevant transaction from the batch application.

9.3.3 Tuning parameters for setting up the timeout

This section explains how to set up the tuning parameters used for timeout settings.

(1) Timeout set up in the batch server for remotely invoking the Enterprise Bean (RMI-IIOP communication) and for invoking the Naming Service by JNDI

A tuning parameter for setting up the timeout at point 1 of *Figure 9-1*.

The setup method is same as for the J2EE application execution platform. See *(6) Timeout set in the EJB client for remotely invoking the Enterprise Bean (RMI-IIOP communication) and for invoking the Naming Service by JNDI* of *8.6.8 Tuning parameters for setting the timeout*.

Note that *point 7* of *8.6.8 Tuning parameters for setting the timeout* correspond to *point 1* of *Figure 9-1*.

(2) Timeout set up in the EJB client for invoking the Enterprise Bean from CTM

A tuning parameter for setting up the timeout at point 2 of *Figure 9-1*.

The setup method is same as for the J2EE application execution platform. See *(7) Timeout set up in the EJB client for invoking the Enterprise Bean from CTM* of *8.6.8 Tuning parameters for setting the timeout*.

Note that *point 8* of *8.6.8 Tuning parameters for setting the timeout* correspond to *point 2* of *Figure 9-1*.

(3) Timeout for the method execution time set up in the EJB that is accessed by invoking the Enterprise Bean

This is a tuning parameter for setting up a timeout at point 3 of *Figure 9-1*.

The setup method is same as for the J2EE application execution platform. See *(11) Method timeout in J2EE application* of *8.6.8 Tuning parameters for setting the timeout*.

Note that *point 9* of *8.6.8 Tuning parameters for setting the timeout* correspond to *point 3* of *Figure 9-1*.

(4) Timeout set up in the batch server for the database transaction (when a DB Connector is used)

This is a tuning parameter for setting up the timeout at point 4 of *Figure 9-1*.

You set up the tuning parameter for each batch server, Enterprise Bean, interface, or each invocation by API (in BMT).

The following table describes the tuning parameters for a transaction timeout:

Table 9–3: Tuning parameters for a transaction timeout

Unit	Setup method	Setup item	Location of setup
Each batch server	Smart Composer functionality	Default transaction timeout value of a transaction	Definition file Easy Setup definition file Setup target Logical J2EE server (`j2ee-server`) Parameter name `ejbserver.jta.TransactionManager.defaultTimeOut`
Each API (BMT)	API	Transaction timeout period	`UserTransaction#setTransactionTimeout` method#

#

The name of the package is `javax.transaction`.

(5) Database timeout

This is a tuning parameter for setting up a timeout at point 5 of *Figure 9-1*.

The setup method is same as for the J2EE application execution platform. See *(10) Database timeout* of *8.6.8 Tuning parameters for setting the timeout*.

Note that *point 12* of *8.6.8 Tuning parameters for setting the timeout* correspond to *point 5* of *Figure 9-1*.

9.4 Setting the thresholds for GC control

You can control the execution timings of Full GC of batch servers by setting up threshold values for memory usage. Hitachi recommends that the threshold value be set up when the same resources are accessed during batch processing and online processing. By setting up the appropriate threshold value, you can secure the throughput of both online processing and batch processing.

Tip

For details about the Full GC, see *7.2.7 Relationship between GC occurrence and memory space*.

9.4.1 Purpose of setting up a threshold value

When the same resources are accessed during online processing and batch processing, you must take into consideration the fact that there is no influence on the throughput of the online processing.

When the available memory space becomes less during the execution of a batch application, JavaVM executes Full GC of the batch server. In such a case, the processing of all programs running on the batch server is interrupted. If some resources are excluded for the batch application, those resources remain in the exclusion state even during the execution of Full GC of the batch server. If a process uses the excluded resources during the online processing, the online processing will also be interrupted.

To avoid this, you must setup a threshold value for the memory usage, and execute Full GC before the memory becomes insufficient. The explicit Full GC can be controlled such that it occurs when the resources are not excluded. By increasing the available memory space before JavaVM executes Full GC, you can prevent the execution of the Full GC while the resources are in the exclusion state.

When a threshold value is set up, Full GC will be executed in the following conditions. However, if some resources are excluded for the batch application at the same time, the Full GC will not be executed until the exclusion is cancelled.

When Serial GC is enabled

- Ratio of the Tenured area consumption size to the Tenured area total size ≥ Threshold value
- Ratio of the New area total size to the Tenured area maximum available size ≥Threshold value
- Ratio of the consumed Metaspace area to the maximum Metaspace area size ≥ Threshold value

When G1 GC is enabled

- Ratio of the consumed Java heap area size to the total Java heap area size ≥ Threshold value
- Ratio of the consumed Metaspace area size to the maximum Metaspace area size ≥ Threshold value

9.4.2 Concept of setting up the threshold value

You can calculate the threshold value by keeping the value calculated with the following formula as the standard:

```
Threshold-value ≤ 100 - (100 × Free-memory-size-required-while-waiting-for-c
ancellation-of-resource-exclusion) / Maximum-memory-size
```

Consider the following points when setting up the threshold value:

- **Occurrence frequency of Full GC**
- **Free memory required while waiting for cancellation of the resource exclusion**

The relationship between the occurrence frequency of Full GC and the threshold value, and the method for estimating the free memory size required while waiting for cancellation of resource exclusion are as follows:

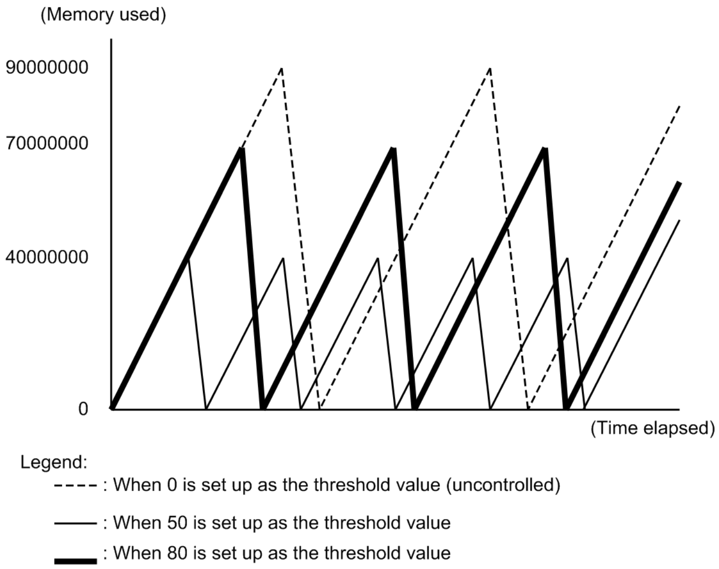
(1) Relationship between the occurrence frequency of Full GC and the threshold value

The processing of Full GC takes more time as compared to the execution speed of any program. Therefore, when tuning JavaVM, try to avoid the occurrence of Full GC as far as possible. For details about the concept of JavaVM tuning, see *7.3.1 Concept of tuning*.

You must also tune in such a way so that setting up a threshold value reduces the frequency of a Full GC to be executed explicitly.

The following figure shows an example of memory usage for each setup threshold value:

Figure 9–2: Example of memory usage for each setup threshold value



The memory usage in JavaVM keeps increasing along with the passage of time, and reduces when Full GC occurs.

When you set up `0` as the threshold value, Full GC will not be executed until JavaVM executes it automatically. As compared to the specification of a large value, when you set up a small value as the threshold value, the frequency of occurrence of Full GC increases. In the figure, when `80` is set up as the threshold value, the frequency of execution of Full GC can be suppressed down as compared to the case where `50` is set up as the threshold value.

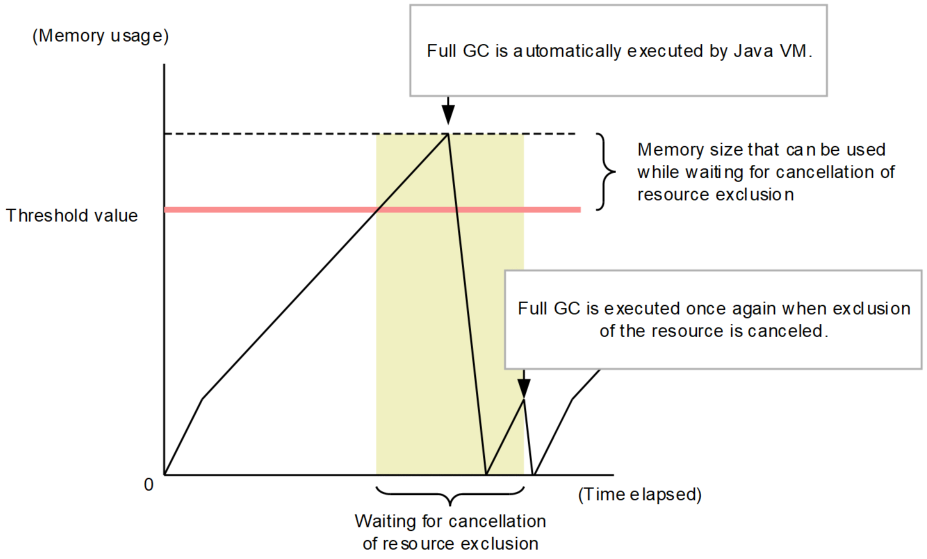
However, when the execution frequency of Full GC is reduced, the time taken for the execution of one Full GC becomes longer as compared to when Full GC is executed more frequently.

(2) Estimating the free memory required while waiting for cancellation of the resource exclusion

When the memory usage exceeds the threshold value, Full GC will not be executed until the resource exclusion is cancelled. However, while waiting for cancellation of resource exclusion, if the memory required by JavaVM becomes insufficient, Full GC will be executed by JavaVM without waiting for cancellation of resource exclusion.

The following figure shows an example in which the memory becomes insufficient while waiting for cancellation of resource exclusion:

Figure 9–3: Example of a case when the memory becomes insufficient while waiting for cancellation of resource exclusion



Tip

For details about the free memory size required to avoid the occurrence of Full GC in JavaVM, see *7.2.7 Relationship between GC occurrence and memory space*.

When you set up the memory size for the occurrence of Full GC to `100`, the memory size that can be used while waiting for cancellation of resource exclusion will be `100` - Threshold value (%).

For example, if you set up a large value, such as `95` as the threshold value, the free memory size that can be used while waiting for cancellation of resource exclusion will be minimal at 5%, and therefore, Full GC might be executed automatically by JavaVM before the cancellation of the resource exclusion.

Therefore, when estimating the threshold value, set up a value with a considerable margin such that the memory does not become insufficient while waiting for cancellation of the resource exclusion.

Important note

When JavaVM executes Full GC due to insufficient free memory size while waiting for cancellation of resource exclusion, Full GC will be executed once again when the exclusion of the resource is cancelled, due to Full GC control.

9.4.3 Tuning parameters for setting thresholds for GC control

This section explains how to set the tuning parameters used to set the threshold values that control when to execute Full GC on the batch server.

Table 9–4: Tuning parameters for setting thresholds for Full GC execution on batch server

Setup item	Location of setup
Threshold value	Definition file Easy Setup definition file Setup target Logical J2EE server (`j2ee-server`) Parameter name `ejbserver.batch.gc.watch.threshold`

Appendixes

A. Efficient Usage of the Explicit Heap Used in an HTTP Session

When you use the Explicit Memory Management functionality, the HTTP session-related objects are allocated to the Explicit heap with the default J2EE server settings. For details on the allocation and release times of the Explicit memory block area to which an HTTP session is allocated, see *7.4.1 Objects related to HTTP session* in the *uCosminexus Application Server Expansion Guide*. Note that for details on the concept of an object life span and the types of Explicit memory blocks that are automatically released, see *Appendix B Effect on the Explicit Memory Management Functionality Due to the Life Span of the Objects Allocated to the Explicit heap*.

You can apply the Explicit Memory Management functionality to an HTTP session efficiently, if you implement an application from the following perspective:

- Life span of the objects stored in an HTTP session
- Update frequency of the objects stored in an HTTP session
- Timing at which an HTTP session is created

Furthermore, you can check whether the application is being implemented as per this perspective by using the J2EE server log.

This section describes how to implement an application and how to check the implementation using the J2EE server log, for each perspective.

A.1 Considering the life span of the objects stored in an HTTP session

This subsection describes how to implement an application considering the life span of the objects stored in an HTTP session and how to check the implementation using the J2EE server log.

(1) Considerations for the implementation of applications

We recommend that only the objects released when an HTTP session is destroyed be stored in an HTTP session. If the percentage of such objects is high, the time taken for the automatic release processing reduces. If there are objects that remain even after the HTTP session is destroyed, a new Explicit memory block is generated, the objects are transferred to that block, and are subject to the subsequent automatic release processing. Also, the transferred objects do not support the HTTP session.

With the Explicit Memory Management functionality, the objects referenced directly or indirectly from the objects stored in the HTTP session using the `setAttribute` method are assumed to have been stored in the HTTP session. These objects move to the Explicit memory block according to the timing of promotion. If the reference relationship from the objects stored in the HTTP session becomes complicated, there is a higher possibility that the objects that continue to be used even after the HTTP session is destroyed are stored in the HTTP session.

Therefore, we recommend that the objects to be stored in an HTTP session be made as simple as possible, such as the `String` type objects and the objects storing the primitive type or primitive type array.

From the application design point of view, if the objects that continue to be used even after the HTTP session is destroyed must be stored in the HTTP session, consider using the functionality for specifying the classes to be excluded from the Explicit Memory Management functionality. For details on the functionality for specifying the classes to be excluded from the Explicit Memory Management functionality, see *7.10 Reducing time required for automatic release processing of Explicit memory blocks* in the *uCosminexus Application Server Expansion Guide*.

(2) How to check using the J2EE server log

You check the life span of the objects stored in an HTTP session from the contents of the thread dump.

To check:

1. Start the J2EE server, and execute the general business until the HTTP session is destroyed.
2. After the HTTP session is destroyed, execute the `eheapprof` command to obtain the thread dump.
3. Find the Explicit memory block where `<EM_NAME>` is `NULL`.
   If a `NULL` Explicit memory block does not exist, this indicates that no objects continue to be used even after the HTTP session is destroyed.
   If a `NULL` Explicit memory block exists, check the value of `used` (used size of the Explicit memory block).
   An example of output is as follows:

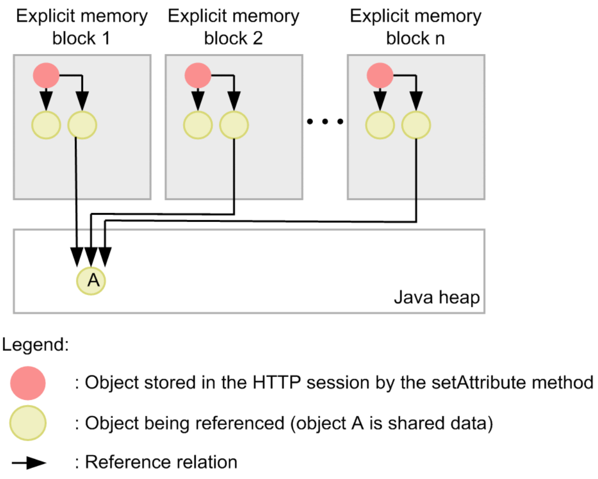
```
"NULL" eid=1(0x02f25610)/A, total 542K, used 501K, garbage 0K (92.4% used/
total, 0.0% garbage/used, 0 blocks) Enable
```

   This indicates that this Explicit memory block has objects that continue to be used even after the HTTP session is destroyed. However, if the `used` value is from a few kilobytes to a few megabytes, the impact during the automatic release processing is not great.

(3) Mechanism of using the objects stored in an HTTP session after the HTTP session is destroyed

This section describes the mechanism by which the objects stored in an HTTP session are used even after the HTTP session is destroyed. As a typical example, the following figure shows an example of referencing the common data (object) from the objects stored in an HTTP session.

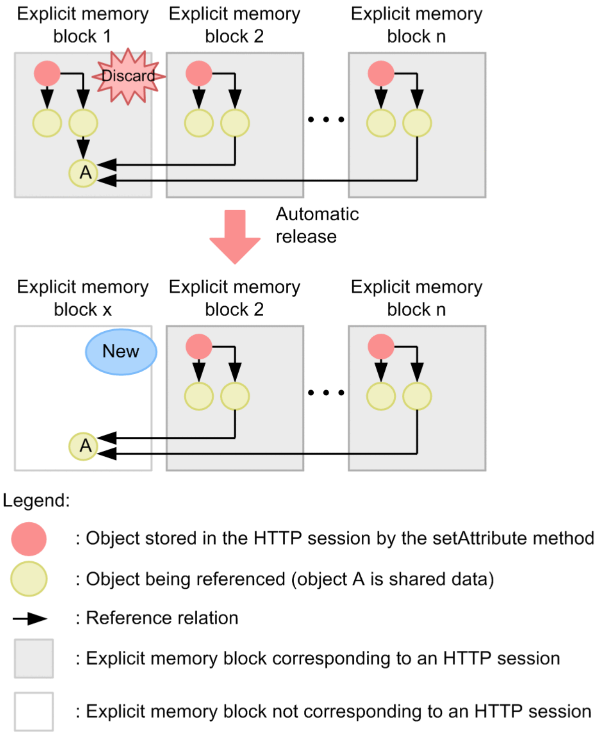
Figure A–1: Example of referencing the common data (object) from the objects stored in an HTTP session


Explicit memory block 1
Explicit memory block 2
Explicit memory block n
A
Java heap
Legend:
● : Object stored in the HTTP session by the setAttribute method
○ : Object being referenced (object A is shared data)
→ : Reference relation

In this example, the `setAttribute` method is used to reference the common data, object A, from the objects stored in an HTTP session. The object A does not appear to have been stored in an HTTP session, but due to the transfer of objects based on a reference relationship, the object is moved to an Explicit memory block corresponding to an HTTP session at the reference source. In this example, the object is assumed to have moved to the Explicit memory block 1. For details on the transfer of objects based on a reference relationship, see *7.6.5 Moving the objects from the Java heap to the Explicit memory block based on a reference relation* in the *uCosminexus Application Server Expansion Guide*.

Thereafter, if the HTTP session corresponding to the Explicit memory block 1 is destroyed and the automatic release processing occurs, the object A referenced from outside (Explicit memory block that is not a target of the release processing) is moved to the new Explicit memory block x. This Explicit memory block x does not support the HTTP session. The following figure shows the status of the Explicit memory block after the HTTP session is destroyed.

Figure A–2: Status of the Explicit memory block after the HTTP session is destroyed


Explicit memory block 1
Explicit memory block 2
Explicit memory block n
Discard
A
Automatic release
Explicit memory block x
Explicit memory block 2
Explicit memory block n
New
A
Legend:
: Object stored in the HTTP session by the setAttribute method
: Object being referenced (object A is shared data)
: Reference relation
: Explicit memory block corresponding to an HTTP session
: Explicit memory block not corresponding to an HTTP session

Greater the number of objects in use in the target Explicit memory block, longer is the time taken by the automatic release processing corresponding to the destruction of the HTTP session. Also, the objects, which are being used during the automatic release processing corresponding to the destruction of the HTTP session, are moved to an Explicit memory block that does not support the HTTP session, and are subject to the automatic release processing. The longer the period, during which these objects cannot be released, the higher the frequency at which these objects become subject to the automatic release processing and this might lead to deterioration in the throughput and latency.

In this example, the references to the common data, object A, last until one of the *n* number of HTTP sessions is destroyed. During this time, the automatic release processing of the new Explicit memory block `x` is repeated and the operation becomes inefficient. If the size of the common data is small, the impact is not particularly great, but in the case of hundreds of megabytes or more, the impact on the automatic release processing time is great, and might cause deterioration in the throughput and latency.

A.2 Considering the update frequency of the objects stored in an HTTP session

This subsection describes how to implement an application considering the update frequency of the objects stored in an HTTP session and how to check the implementation using the J2EE server log.

(1) Considerations for the implementation of applications

Smaller the update frequency of the objects stored in an HTTP session, better the memory efficiency of the Explicit memory block. For example, if you invoke multiple `setAttribute` methods with the same value in the first argument `name`, the objects in the second argument `value` are only updated for the number of times the method is invoked with the same value. This is the *update frequency* of the object.

If the objects stored in an HTTP session are updated, the objects before update are used, but that memory area is not released until the HTTP session is destroyed. Therefore the lower the update frequency the better the memory efficiency. Note that there is no problem if the object updates are repeated in short cycles (cycles of about 10 times or less of the copy GC interval), but if the updates are repeated in long cycles, the object becomes a long life span object and is moved to the Explicit memory block. The objects that are transferred to the Explicit memory block are not released until the HTTP session is destroyed even if the object has already been used. If the Explicit Memory Management functionality is not applied, such objects are released when a full garbage collection occurs. Typically, in systems where the survival duration of an HTTP session is long, for example the HTTP session is not destroyed during the day, the object updates might be easily repeated in long cycles.

From the application design point of view, if the update frequency of the objects stored in an HTTP session increases, consider the following points to determine the applicability of the Explicit Memory Management functionality:

- Spike in latency due to the full garbage collection when the Explicit Memory Management functionality is not applied
- Deterioration in memory efficiency when the Explicit Memory Management functionality is applied

(2) How to check using the J2EE server log

You check the update frequency of the objects stored in an HTTP session from the contents of the thread dump.

To check:

1. Start the J2EE server, and execute the business until just before the HTTP session is destroyed.

2. Execute the `eheapprof` command to obtain the thread dump.

3. Find the Explicit memory block where `<EM_NAME>` is `CCC#HttpSession`.

   The `CCC#HttpSession` Explicit memory block corresponds to the HTTP session. Compare the value of `used` (used size of the Explicit memory block) and the total size of the objects stored in the HTTP session, calculated and estimated during application development. If the update frequency of the objects is high, there is a big difference between the two values, and the value of `used` is greater.

   An example of output is as follows:

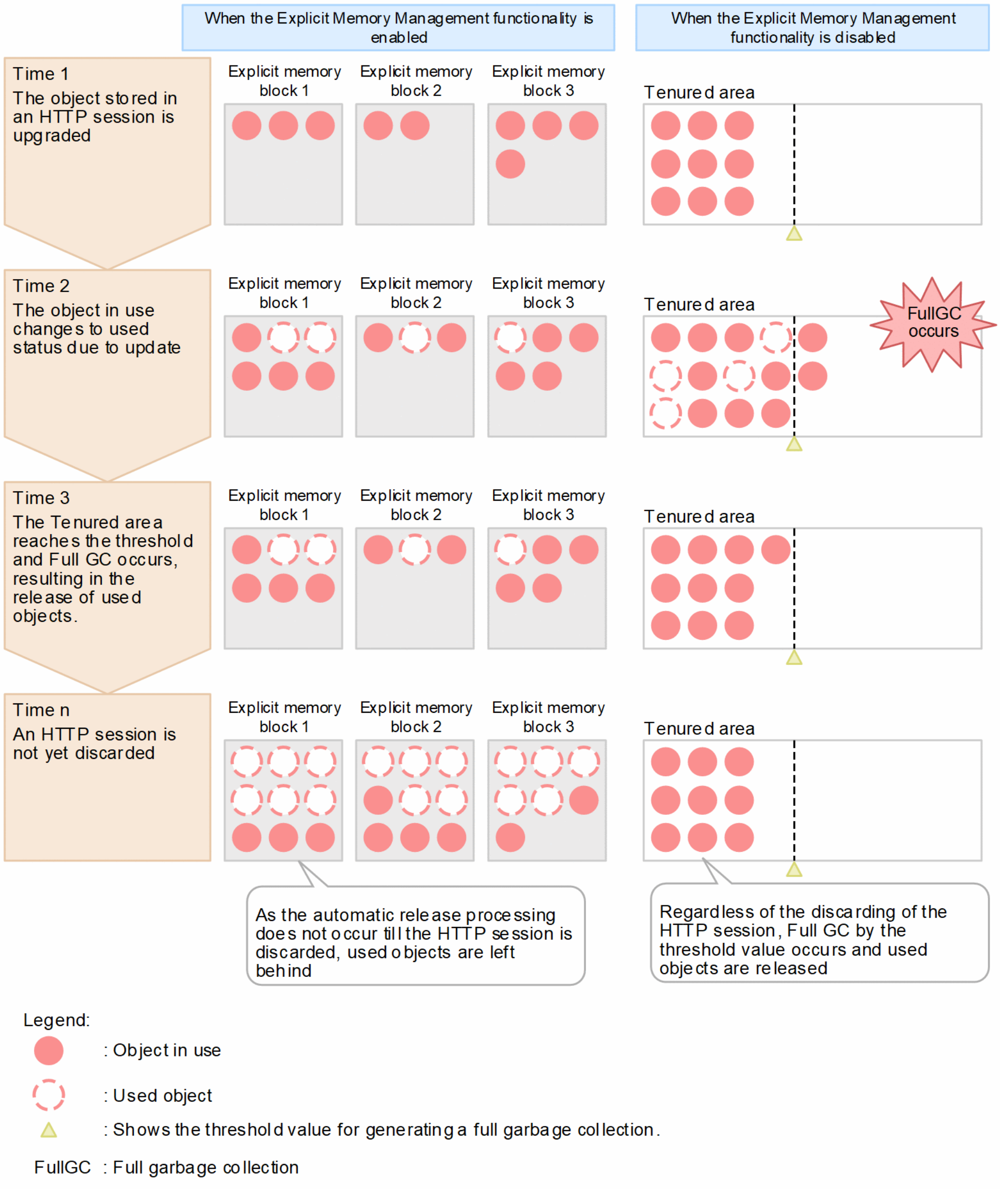
   ```
   "CCC#HttpSession" eid=97(0x02f25610)/R, total 16K, used 8K, garbage 0K (50
   .1% used/total, 0.0% garbage/used, 0 blocks) Enable
   ```

(3) Mechanism of retaining the used HTTP session-related objects in the Explicit memory block

This point describes the mechanism by which the used HTTP session-related objects remain in the Explicit memory block as a result of repeated updates. The objects described here are long life-span objects with survival duration shorter than the destruction of an HTTP session.

The following figure shows the differences in the object release processing when the Explicit Memory Management functionality is enabled and when the functionality is disabled.

Figure A–3: Object release processing when the Explicit Memory Management functionality is enabled and when the functionality is disabled


When the Explicit Memory Management functionality is enabled
When the Explicit Memory Management functionality is disabled
Time 1
The object stored in an HTTP session is upgraded
Explicit memory block 1
Explicit memory block 2
Explicit memory block 3
Tenured area
Time 2
The object in use changes to used status due to update
FullGC occurs
Time 3
The Tenured area reaches the threshold and Full GC occurs, resulting in the release of used objects.
Time n
An HTTP session is not yet discarded
As the automatic release processing does not occur till the HTTP session is discarded, used objects are left behind
Regardless of the discarding of the HTTP session, Full GC by the threshold value occurs and used objects are released
Legend:
: Object in use
: Used object
: Shows the threshold value for generating a full garbage collection.
FullGC : Full garbage collection

This figure is an example of an HTTP session that is not destroyed for a long time. With `Time 2`, four objects are used up due to an object update. With `Time 3`, when the Explicit Memory Management functionality is disabled, the Tenured area size exceeds the threshold value for the occurrence of a full garbage collection, so the full garbage collection (in the figure, Full GC occurs is applicable) occurs, and the used objects are released. When the Explicit Memory Management functionality is enabled, the full garbage collection does not occur, so the used objects remain behind and are not released. Furthermore, even after the lapse of `Time n`, the HTTP session is not yet destroyed. When the Explicit Memory Management functionality is disabled, the used objects of the Tenured area are released with the occurrence of the threshold value-based full garbage collection. When the Explicit Memory Management functionality is enabled, the automatic release processing does not occur until the HTTP session is destroyed, so the used up and un-released objects remain behind and the possibility of Explicit heap overflow increases.

By `Time n` in this example, there are `16` objects that are used up and have not been released, but in a system that requires an HTTP session that is not destroyed for a long time (for example, during a day), there might be a situation in which even more used objects are not released.

A.3 Considering the timing at which an HTTP session is created

This subsection describes how to implement an application considering the times at which an HTTP session is created and how to check the implementation using the J2EE server log.

(1) Considerations for the implementation of applications

If you specify settings so that only the HTTP sessions used in applications are created, the memory efficiency of the Explicit memory block improves. Note that a J2EE server has the functionality for using an HTTP session implicitly.

An Explicit memory block and an HTTP session have a one-to-one correspondence. Also, the minimum size of one Explicit memory block is 16 KB (default is 64 KB). If you create one HTTP session, the Explicit heap area uses 16 KB, even if no objects are stored in the HTTP session. Therefore, the memory efficiency becomes better if you create the HTTP sessions that are actually used. For example, assume an application that creates an un-used HTTP session whenever a servlet is accessed, and the HTTP session is not destroyed until the session times out. If you assume that the timeout is 30 minutes and that the servlets are accessed 10,000 times in 30 minutes, a `16 KB× 10,000 = 160 MB` (by default, 640 MB) Explicit heap area will be used.

Also, with the JSPs, an HTTP session is created for each access by default. Therefore, if an HTTP session uses a JSP with an unnecessary processing (for example, the health check of a J2EE server), an unnecessary HTTP session object is generated and the Explicit heap might overflow. With JSPs that do not need sessions, specify settings so that the `HttpSession` object is not created explicitly. You use the `session` attribute of the `page` directive in the settings.

If you cannot change the application, consider using the memory saving functionality of the Explicit heap to be used in the HTTP session. For details on the memory saving functionality of the Explicit heap to be used in the HTTP session, see *7.11 Reducing memory usage of the Explicit heap that is used in an HTTP session* in the *uCosminexus Application Server Expansion Guide*. If you use this functionality, even before the HTTP session is destroyed, the objects stored in an Explicit memory block with a low utilization rate are moved and consolidated to another area, and the Explicit memory block with a low utilization rate is automatically released. The relationship of the HTTP sessions and Explicit memory blocks in Application Server becomes many to one. One Explicit memory block can be shared by multiple HTTP sessions, so the utilization rate of the Explicit memory block improves. Due to this, you can reduce the memory usage of the Explicit heap allocated to the HTTP session.

(2) How to check using the J2EE server log

You check whether Explicit memory blocks are being created for unnecessary HTTP sessions, from the contents of the thread dump.

To check:

1. Start the J2EE server, and execute the business.

   An HTTP session might also be used implicitly in an application. Therefore, do not limit the business to one using the HTTP sessions.

2. During the execution of the business, execute the `eheapprof` command to obtain the thread dump.

   Check the values of `used/total` (Explicit memory block utilization rate) and `spaces exist` (number of enabled Explicit memory blocks) output to the Explicit heap information.
   An example of output is as follows:

   ```
   max 31415926K, total 162816K, used 150528K, garbage 10004K (0.0% used/max
   , 91.1% used/total, 6.6% garbage/used), 3 spaces exist
   ```

In this example, the value of `used/total` is 91.1%. The closer this value is to 100%, the better is the memory efficiency. Check whether this value is a single digit and whether the value of `spaces exist` greatly exceeds the assumed number of HTTP sessions.

B. Effect on the Explicit Memory Management Functionality Due to the Life Span of the Objects Allocated to the Explicit heap

The efficient usage of the Explicit Memory Management functionality is greatly affected by the life span of the objects allocated to the Explicit heap. When the Explicit memory block in an Explicit heap is automatically released, if all the objects in the Explicit memory block are already used, the automatic release processing time reduces and the memory efficiency improves.

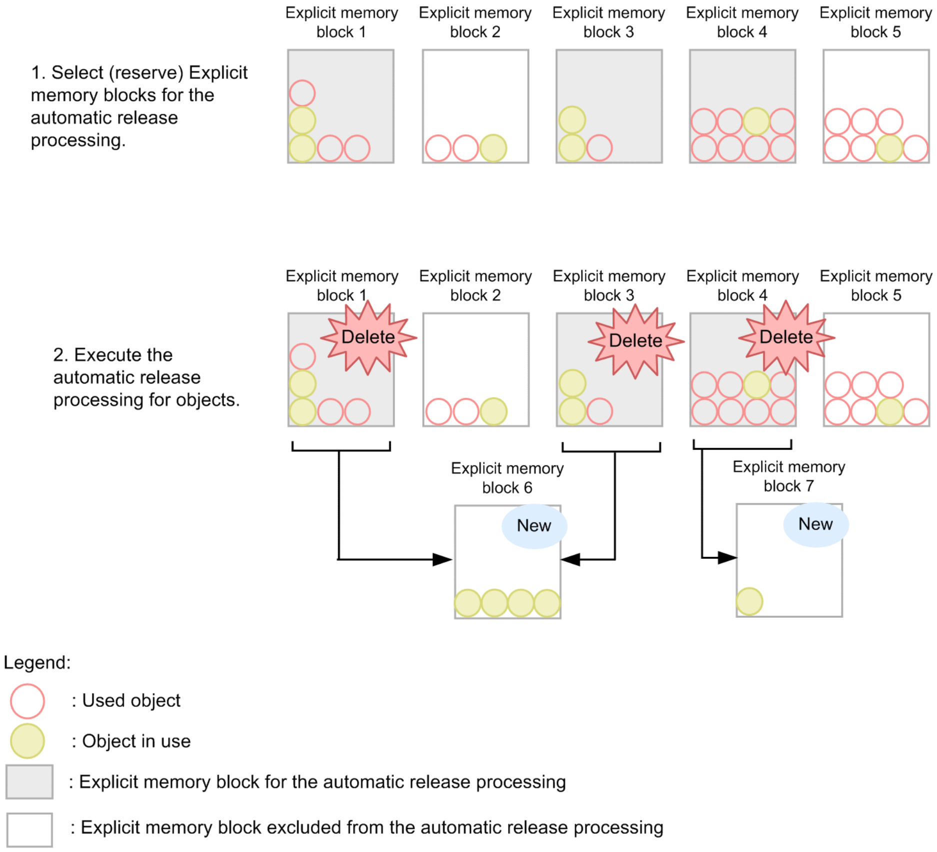
This appendix describes how to select the Explicit memory block subject to the automatic release processing of the Explicit Memory Management functionality, and the concepts of the memory usage of an Explicit heap and the life span of an object. After you understand the concepts, check the points to be considered in *Appendix A Efficient Usage of the Explicit Heap Used in an HTTP Session*, and determine the efficient usage of the Explicit Memory Management functionality.

B.1 Effect on the automatic release processing of the Explicit memory block

The efficient usage of the Explicit Memory Management functionality is greatly affected by the efficiency of the automatic release processing of the Explicit memory block. To execute the automatic release processing carefully so that a latency spike does not occur with the Explicit Memory Management functionality, divide and manage the memory areas (Explicit heap) into Explicit memory blocks, and execute the automatic release processing for the blocks. The automatic release processing of the Explicit memory block is executed during GC as and when required. Also, the execution of applications stops during the automatic release processing, as in the case of a full garbage collection. Therefore, the shorter the automatic release processing time, the better the throughput and latency.

The following figure shows the automatic release processing of the Explicit memory block.

Figure B–1: Automatic release processing of the Explicit memory block


Explicit memory block 1
Explicit memory block 2
Explicit memory block 3
Explicit memory block 4
Explicit memory block 5
1. Select (reserve) Explicit memory blocks for the automatic release processing.
2. Execute the automatic release processing for objects.
Delete
Delete
Delete
Explicit memory block 6
New
Explicit memory block 7
New
Legend:
○ : Used object
● : Object in use
▨ : Explicit memory block for the automatic release processing
□ : Explicit memory block excluded from the automatic release processing

With the automatic release processing, if the Explicit memory block subject to the automatic release processing has in-use objects, a new Explicit memory block is created and the objects are moved to this block. If there are multiple in-use objects, the automatic release processing time increases in proportion to that amount. Also, the automatic release processing must be re-executed for the new Explicit memory blocks (6 and 7 in this example), so the overall automatic release processing time also increases. Therefore, during the automatic release processing, the fewer the in-use objects in the Explicit memory block, the higher the efficiency of the automatic release processing. This also indicates that with the Explicit Memory Management functionality, the life span of the objects affects the efficiency.

Note that the following subsection describe how to select Explicit memory blocks for the automatic release processing.

How to select Explicit memory blocks for the automatic release processing

JavaVM selects (reserved for release) an Explicit memory block corresponding to one of the following blocks as the target of the automatic release processing.

1. Blocks corresponding to the HTTP sessions destroyed between the previous and the next automatic release processing
2. New blocks created during the automatic release processing
3. Blocks created by the automatic allocation functionality

However, for the blocks in 2 and 3, only the blocks selected by estimating the increment and the release ratio of the Explicit heap, or by the threshold value are subject to the automatic release processing.

This subsection describes how to select a block by estimating the increment and the release ratio of the Explicit heap, or by the threshold value.

Selection by estimating the increment and the release ratio of the Explicit heap

Select multiple blocks based on the following information. Normally, this method is used.

- Increment in the Explicit heap size after the previous automatic release processing

- Release ratio of the past automatic release processing (ratio of objects that could be released automatically)
- Estimated value of the object usage rate in each block

First, from "increment in the Explicit heap size after the previous automatic release processing", set up the target value (hereafter called the *target automatic release size*) of the Explicit heap size to be released automatically. The size released automatically must be more than the increment in the Explicit heap size so that the Explicit heap does not increase monotonically, and the greater the increment, the larger the target automatic release size. To automatically release the target automatic release size, calculate the size that might be selected from the total of multiple blocks (hereafter called the *selection size*). Even if the automatic release processing is executed for a particular block, all the objects in that block cannot necessarily be released. Therefore, based on the "release ratio of the past automatic release processing", estimate the release ratio of the objects to be released automatically. With the estimated release ratio, calculate the selection size required for reaching the target automatic release size. The lower the "release ratio of the past automatic release processing", the greater the selection size.

Next, sort all the blocks with the "estimated value of the object usage rate in each block", if the percentage of the in-use objects is low, select the blocks sequentially from the estimated blocks. If the total size of the selected blocks reaches the selection size, discontinue the selection process. The blocks selected here are subject to the automatic release processing.

Selection by the threshold value

The blocks that always have a high percentage of in-use objects, and the blocks with size exceeding the selection size are not selected with "Selection by estimating the increment and release ratio of the Explicit heap". Therefore, if the size of the Explicit memory block exceeds a fixed percentage of the overall Explicit heap size (threshold value), the block is forcefully selected as a target for automatic release processing. Note that the blocks selected with "Selection by the threshold value" are not selected with "Selection by estimating the increment and the release ratio of the Explicit heap".

B.2 Effect on the memory usage of the Explicit heap

The memory usage of the Explicit heap greatly affects the efficient usage of the Explicit Memory Management functionality. Until the block is subject to the automatic release processing, the more the number of used objects in the Explicit memory block, the lower the memory efficiency of the Explicit heap. For example, for the Explicit memory block 2 in *Figure B-1*, two out of the three objects are used up (66% are used up). However, the used objects in the Explicit memory block 2 are not released automatically until the block is selected for the automatic release processing. This indicates that if there are many in-use objects in the Explicit memory block before the automatic release processing, the memory efficiency improves.

B.3 Relation between the reference relationship and the life span of objects allocated to the Explicit heap

When GC occurs, objects referenced from the objects allocated to an Explicit heap are transferred from the Java heap to the Explicit memory block. This processing is *Moving the objects from the Java heap to the Explicit memory block based on a reference relation*. This processing is applied recursively, and the objects until the end of the reference relationship are transferred to the Explicit memory block. Therefore, from the objects in the Explicit memory block, the fewer the references to the objects with different life spans, the more similar are the life spans of the objects in the Explicit memory block. By making the life spans identical, the objects are automatically released concurrently, and the memory efficiency improves. In other words, you must implement the applications in such a way so that the reference relationship of the objects closes between the objects with as identical life spans as possible. For example, the efficiency improves if there is no reference relationship between a management table object with a long life span and a data object used for one business with a short life span. For details on transferring the objects based on a reference relationship from the Java heap to the Explicit memory block, see *7.6.5 Moving the objects from the Java heap to the Explicit memory block based on a reference relation* in the *uCosminexus Application Server Expansion Guide*.

C. Tuning Parameters for Performing the Performance Tuning with Methods other than the Recommended Procedures

This appendix describes the tuning parameters for tuning the items described in *8. Performance Tuning (J2EE Application Execution Platform)* and *9. Performance Tuning (Batch Application Execution Platform)* with methods other than the recommended procedures. The methods other than those recommended include the method for the settings using the management portal and the method for the settings by editing files.

For details about the concept of tuning, see *8. Performance Tuning (J2EE Application Execution Platform)* and *9. Performance Tuning (Batch Application Execution Platform)*.

C.1 Tuning parameter for optimizing the number of concurrent executions (methods other than the recommended procedures)

This subsection describes the methods and locations for setting up the tuning parameters used to optimize the number of concurrent executions.

(1) Number of request-processing threads when using an NIO HTTP server

The following table describes the methods and locations to set up the tuning parameters for the number of request-processing threads when using an NIO HTTP server.

For details about the tuning parameters when Cosminexus HTTP Server is used for Web server integration, see the *uCosminexus Application Server HTTP Server User Guide*.

Table C–1: Tuning parameters for the number of request-processing threads when using an NIO HTTP (methods other than the recommended procedures)

Setup item	Method of setup	Location of setup
Number of request-processing threads generated when the J2EE server is started	Edit file	`webserver.connector.nio_http.min_threads` key in `usrconf.properties`
Maximum number of connections to the Web client or reverse proxy	Edit file	`webserver.connector.nio_http.max_connections` key in `usrconf.properties`
Maximum value (backlog) of the TCP/IP Listen queue used when the maximum number of connections to the Web client or reverse proxy is exceeded	Edit file	`webserver.connector.nio_http.backlog` key in `usrconf.properties`
Maximum number of request-processing threads	Edit file	`webserver.connector.nio_http.max_threads` key in `usrconf.properties`

(2) Number of concurrent executions in a Web application

Setup for each URL group, Web application, or Web container.

(a) Number of concurrent executions for a URL group

The following table describes the methods and locations to set up the tuning parameters for the number of concurrent executions of a URL group:

Table C–2: Tuning parameters for the number of concurrent executions of a URL group (methods other than the recommended procedures)

Setup item	Method of setup	Location of setup
Maximum number of concurrently executed threads in each Web container	Edit file	`webserver.connector.nio_http.max_servlet_execute_threads` key in `usrconf.properties`
Default pending queue size	Edit file	`webserver.container.thread_control.queue_size` key in `usrconf.properties`

The following setup items are the same as specified for the recommended procedures:

- Maximum number of concurrently executed threads in each Web application
- Number of dedicated threads for the Web application
- Size of the pending queue in a Web application
- Definition name of concurrently executed thread control of a URL group
- Maximum number of concurrently executed threads in a URL group
- Number of dedicated threads for a URL group
- Size of the pending queue in a URL group
- URL pattern to be controlled by a URL group

(b) Number of concurrent executions for a Web application

The following table describes the methods and locations to set up the tuning parameters for the number of concurrent executions in a Web application:

Table C–3: Tuning parameters for the number of concurrent executions in a Web application (methods other than the recommended procedures)

Setup item	Method of setup	Location of setup
Maximum number of concurrently executed threads in Web container unit	Edit file	`webserver.connector.nio_http.max_servlet_execute_threads` key in `usrconf.properties`
Default pending queue size	Edit file	`webserver.container.thread_control.queue_size` key in `usrconf.properties`

The following setup items are the same as specified for the recommended procedures:

- Maximum number of concurrently executed threads in each Web application
- Number of dedicated threads of the Web application
- Size of the pending queue in a Web application

(c) Number of concurrent executions for a Web container

The following table describes the methods and locations to set up the tuning parameters for the number of concurrent executions in a Web container:

Table C–4: Tuning parameters for the number of concurrent executions in a Web container (methods other than the recommended procedures)

Setup item	Method of setup	Location of setup
Maximum number of concurrently executed threads in each Web container	Edit file	`webserver.connector.nio_http.max_servlet_execute_threads` key in `usrconf.properties`

(3) Number of concurrent executions in an Enterprise Bean

Set up the number of concurrent executions in an Enterprise Bean for each Enterprise Bean.

The tuning parameters for the number of concurrent executions in an Enterprise Bean are the same as specified for the recommended procedures.

(4) Number of concurrent executions controlled by CTM

The following table describes the methods and locations to set up the tuning parameters for the number of concurrent executions controlled by CTM. The items are set up in the CTM daemon, application, and Stateless Session Bean.

The following table describes the methods and locations to set up the tuning parameters for the number of concurrent executions controlled by CTM:

Table C–5: Tuning parameters for the number of concurrent executions controlled by CTM (methods other than the recommended procedures)

Setup target	Setup item	Method of setup	Location of setup
CTM daemon	Maximum value of threads controlled by CTM, and number of requests registered in each queue	Edit file	Argument `-CTMDispatchParallelCount` of the `ctmstart` command

The setup items of the application or Stateless Session Bean are the same as specified in the recommended procedures.

C.2 Tuning parameters for optimizing the method of invoking the Enterprise Bean (methods other than the recommended procedures)

The location of tuning parameters used to optimize the method of invoking the Enterprise Bean are as follows:

(1) Using the local interface

You use the local interface defined in J2EE for creating an application.

The method is the same as specified for the recommended procedures.

(2) Using the local invocation functionality of remote interface

The following table describes the methods and locations to set up the tuning parameters for the local invocation functionality of the remote interface:

Table C–6: Tuning parameters for the local invocation functionality of the remote interface (methods other than the recommended procedures)

Setup item	Method of setup	Location of setup
Scope of local invocation optimization functionality	Edit file	`ejbserver.rmi.localinvocation.scope` key in `usrconf.properties`

(3) Using the pass by reference functionality of remote interface

The following table describes the methods and locations to set up the tuning parameters for the pass by reference functionality of the remote interface:

Table C–7: Tuning parameters for the pass by reference functionality of the remote interface (method other than the recommended procedures)

Setup item	Method of setup	Location of setup
Usage of the pass by reference functionality of the remote interface (for each J2EE server)	Edit file	`ejbserver.rmi.passbyreference` key in `usrconf.properties`

The usage of the pass by reference functionality of the remote interface (for each Enterprise Bean) is the same as specified in the recommended procedures.

C.3 Tuning parameters for optimizing the methods of accessing the database (methods other than the recommended procedures)

The tuning parameters used for optimizing the method of accessing the database are the same as specified for the recommended procedures.

C.4 Tuning parameters for specifying the timeout (methods other than the recommended procedures)

This subsection describes the locations to set up the tuning parameters used for specifying the timeout.

(1) Timeout specified in the Web server for receiving requests from the client and sending data to the client

When integrating with Web servers, set the tuning parameters at the Web server level. Only specify these tuning parameters when integrating with Web servers.

Table C–8: Tuning parameters for the timeout to be specified in the Web server for receiving requests from the client and sending data to the client (method other than the recommended procedures)

Method of setup	Location of setup
Management portal	Timeout directive of "Additional directives" of "Perform the setup for each item" in "Web Server Setup" window.
Edit file#	The Timeout directive in httpsd.conf

#
Set up the tuning parameters by editing httpd.conf, a definition file of Cosminexus HTTP Server.

(2) Timeout specified in the reverse proxy of the HTTP Server for sending and receiving data to and from the Web container

The following table describes the tuning parameters for the timeout specified in the reverse proxy of the HTTP Server. You can specify these tuning parameters only for the Web server integration.

Table C–9: Tuning parameters for the timeout specified in the reverse proxy of the HTTP Server (methods other than the recommended procedures)

Method of setup	Location of setup
Management portal	--
Edit file	timeout key of the ProxyPass parameter in httpsd.conf

(3) Timeout specified in the Web container for receiving data from the reverse proxy or Web client

Set up the tuning parameter for each J2EE server. The following table describes the tuning parameters for the timeout to be specified in the Web container:

Table C–10: Tuning parameters for the timeout specified in the Web container (methods other than the recommended procedures)

Method of setup	Location of setup
Edit file	webserver.connector.nio_http.receive_timeout key in usrconf.properties

(4) Timeout specified in the Web container for sending data to the reverse proxy or Web client

Set up the tuning parameter for each J2EE server. The following table describes the tuning parameters for the timeout to be specified in the Web container:

Table C–11: Tuning parameters for the timeout specified in the Web container (methods other than the recommended procedures)

Method of setup	Location of setup
Edit file	webserver.connector.nio_http.send_timeout key in usrconf.properties

C.5 Tuning parameters for optimizing the operations of the Web application (methods other than the recommended procedures)

This subsection describes the locations to set up the tuning parameters used for optimizing the operations of the Web application.

(1) Tuning parameters for separating the deployment of static contents and Web application

Specify the separation of the deployment of static contents and Web application as parameter of the file that defines the operations of the Web server.

Table C–12: Tuning parameters for separating the deployment of static contents and Web application (methods other than the recommended procedures)

Method of setup	Location of setup
Management portal	--
Edit file	`ProxyPass` directive# of `httpsd.conf`

#

For details about `httpsd.conf`, see the *uCosminexus Application Server HTTP Server User Guide*.

(2) Tuning parameters for caching static contents

The tuning parameters for caching static contents are explained in this subsection. You set up these tuning parameters for each Web container or Web application.

The following table describes the methods and locations to set up the tuning parameters specified in each Web container:

Table C–13: Tuning parameters for caching static contents (items to be set up for each Web container) (methods other than the recommended procedures)

Setup items	Method of setup	Location of setup
Select whether static contents cache is to be used	Edit file	`webserver.static_content.cache.enabled` key in `usrconf.properties`
Setup of maximum memory size for each Web application	Edit file	`webserver.static_content.cache.size` key in `usrconf.properties`
Setup of maximum file size of the static contents for cache	Edit file	`webserver.static_content.cache.filesize.threshold` key in `usrconf.properties`

The setup of the tuning parameters for each Web application is the same as that specified in the recommended procedures.

C.6 Tuning parameters for optimizing the operation of CTM (methods other than the recommended procedures)

This subsection describes the locations to set up the tuning parameters used for optimizing the operation of CTM.

(1) Tuning parameters for setting up the monitoring interval of the operating state of CTM domain managers and CTM daemons

The following table describes the parameters for tuning the monitoring interval of the operating state of the CTM domain manager:

The monitoring interval is the value after multiplying the send interval with the coefficient.

Table C–14: Parameters for tuning the monitoring interval of the operating state of CTM domain manager (methods other than the recommended procedures)

Target	Setup item	Method of setup	Location of setup
CTM domain managers existing in the same network segment	Send interval	Execute command	Argument `-CTMSendInterval` of the `ctmdmstart` command
	Coefficient	Execute command	Argument `-CTMAliveCheckCount` of the `ctmdmstart` command
CTM domain managers existing in a different network segment	Send interval	Execute command	Argument `-CTMSendHostInterval` of the `ctmdmstart` command
	Coefficient	Execute command	Argument `-CTMAliveCheckCount` of the `ctmdmstart` command

The parameter for tuning the monitoring interval of the operating state of the CTM daemons is as follows:

Table C–15: Parameter for tuning the monitoring interval of the operating state of CTM daemons (methods other than the recommended procedures)

Setup item	Method of setup	Location of setup
Timeout during transfer between CTM daemons	Execute command	Argument `-CTMDCSendTimeOut` of the `ctmstart` command

(2) Tuning parameters for setting up the monitoring interval of the load status

The tuning parameter for the monitoring interval of the load status is as follows:

Table C–16: Tuning parameter for the monitoring interval of the load information (methods other than the recommended procedures)

Setup item	Method of setup	Location of setup
Timeout during transfer between CTM daemons	Execute command	Argument `-CTMLoadCheckInterval` of the `ctmstart` command

(3) Tuning parameters for setting up the timeout lock for CTM daemon

The timeout lock is executed by setting up the timeout occurrence frequency and the monitoring interval.

The tuning parameters for the timeout lock of the CTM daemon are as follows:

Table C–17: Tuning parameters for the timeout lock of a CTM daemon (methods other than the recommended procedures)

Setup item	Method of setup	Location of setup
Timeout occurrence frequency	Execute command	Argument `-CTMWatchRequest` (first optional argument) of the `ctmstart` command
Monitoring interval	Execute command	Argument `-CTMWatchRequest` (second optional argument) of the `ctmstart` command

(4) Tuning parameters for setting up a priority order for the requests distributed by CTM

The setting of the priority order of requests distributed by CTM is different for an EJB client application and for a J2EE server. Moreover, for a J2EE server, the location of setup differs depending on how the system is built. The tuning parameters for setting up the priority order of requests distributed with CTM, are as follows:

Table C–18: Tuning parameters for setting up the priority order of requests distributed by

Setup unit	Method of setup	Location of setup
J2EE server	Edit file	`ejbserver.client.ctm.RequestPriority` key of `usrconf.properties`

The tuning parameters for each EJB client application are the same as specified in the recommended procedures.

C.7 Tuning parameters for setting up the threshold value that causes full garbage collection of the batch server (methods other than the recommended procedures)

This subsection describes the tuning parameters used to set up the threshold value for executing full garbage collection of the batch server.

Determine the tuning of this item when you want to control the execution of full garbage collection in the batch server.

Table C–19: Tuning parameters for setting up the threshold value to execute full garbage collection of the batch server

Setup item	Method of setup	Location of setup
Threshold value	`usrconf.properties`	`ejbserver.batch.gc.watch.threshold` key

D. Glossary

Terminology used in this manual

For the terms used in the manual, see the *uCosminexus Application Server and BPM/ESB Platform Terminology Guide*.

Index