SYBASE®

Performance and Tuning Guide

EAServer

6.0

DOCUMENT ID: DC20063-01-0600-01

LAST REVISED: July 2006

Contents

About This Book

Subject

This book contains information about configuring server and application settings to achieve the highest application performance. This book also describes implementation and design issues that affect performance.

Audience

This book is for advanced administrators and developers who are familiar with the basics of EAServer administration, development, and deployment.

How to use this book

Chapter 1, "Introduction," explains key performance concepts, describes tools to test and measure performance, and provides techniques for measuring performance and identifying areas where your tuning efforts will have the greatest impact on overall performance.

Chapter 2, "Server Tuning," describes how to configure server and system settings for best performance. These settings affect all applications, regardless of architecture.

Chapter 3, "Component Tuning," describes business component settings and coding practices that you can optimize for best performance. These settings affect applications that call business components from the Web tier or directly from base clients.

Chapter 4, "EJB CMP Tuning," describes how to tune the settings in the EAServer EJB CMP engine and EJB CMP component properties. These settings affect applications that use EJB entity beans with container managed persistence (CMP).

Chapter 5, "Web Application Tuning," describes tuning and coding best practices to create high performance Web sites hosted in EAServer. These settings affect applications that serve static content with EAServer and make use of servlets and JavaServer Pages (JSPs) deployed on EAServer.

Chapter 6, "Database Access Tuning," describes how to tune data source settings and the EAServer transaction manager, and provides coding best practices for interacting with remote databases. These settings affect applications that call remote database servers from business components, servlets, or JSPs deployed on EAServer.

Chapter 7, “Cluster Tuning,” describes how to tune application settings and code to obtain high performance and load balancing in a clustered (multi-server) deployment.

Chapter 8, “Message Service Tuning,” describes how to configure the messages service for maximum performance and explains best coding practices for high performance use of the JMS or message service APIs.

Related documents

Core EAServer documentation The core EAServer documents are available in HTML and PDF format in your EAServer software installation and on the SyBooks™ CD.

What’s New in EAServer 6.0 summarizes new functionality in this version.

The *EAServer API Reference Manual* contains reference pages for proprietary EAServer Java classes and C routines.

The *EAServer Automated Configuration Guide* explains how to use Ant-based configuration scripts to:

- Define and configure entities, such as EJB modules, Web applications, data sources, and servers
- Perform administrative and deployment tasks

The *EAServer CORBA Components Guide* explains how to:

- Create, deploy, and configure CORBA and PowerBuilder™ components and component-based applications
- Use the industry-standard CORBA and Java APIs supported by EAServer

The *EAServer Enterprise JavaBeans User’s Guide* describes how to:

- Configure and deploy EJB modules
- Develop EJB clients, and create and configure EJB providers
- Create and configure applications clients
- Run the EJB tutorial

The *EAServer Feature Guide* explains application server concepts and architecture, such as supported component models, network protocols, server-managed transactions, and Web applications.

The *EAServer Java Message Service User’s Guide* describes how to create Java Message Service (JMS) clients and components to send, publish, and receive JMS messages.

The *EAServer Migration Guide* contains information about migrating EAServer 5.*x* resources and entities to an EAServer 6.0 installation.

The *EAServer Performance and Tuning Guide* (this book) describes how to tune your server and application settings for best performance.

The *EAServer Security Administration and Programming Guide* explains how to:

- Understand the EAServer security architecture
- Configure role-based security for components and Web applications
- Configure SSL certificate-based security for client connections
- Implement custom security services for authentication, authorization, and role membership evaluation
- Implement secure HTTP and IIOP client applications
- Deploy client applications that connect through Internet proxies and firewalls

The *EAServer System Administration Guide* explains how to:

- Start the preconfigured server and manage it with the Sybase Management Console
- Create, configure, and start new application servers
- Define database types and data sources
- Create clusters of application servers to host load-balanced and highly available components and Web applications
- Monitor servers and application components
- Automate administration and monitoring tasks with command line tools

The *EAServer Web Application Programming Guide* explains how to create, deploy, and configure Web applications, Java servlets, and JavaServer Pages.

The *EAServer Web Services Toolkit User's Guide* describes Web services support in EAServer, including:

- Support for standard Web services protocols such as Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), and Uniform Description, Discovery, and Integration (UDDI)
- Administration tools for deployment and creation of new Web services, WSDL document creation, UDDI registration, and SOAP management

The *EAServer Troubleshooting Guide* describes procedures for troubleshooting problems that EAServer users may encounter. This document is available only online; see the EAServer Troubleshooting Guide at http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.eas_5.2.eastg/html/eastg/title.htm.

jConnect for JDBC documents EAServer includes the jConnect™ for JDBC™ 6.0.5 driver to allow JDBC access to Sybase database servers and gateways. The *jConnect for JDBC 6.0.5 Programmer's Reference* is available on the Sybase Product Manuals Web site at http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.jconnjdbc_6.05.prjdbc/html/prjdbc/title.htm&toc=/com.sybase.help.jconnjdbc_6.05/toc.xml.

Sybase Software Asset Management User's Guide EAServer includes the Sybase Software Asset Management license manager for managing and tracking your Sybase software license deployments. The *Sybase Software Asset Management User's Guide* is available on the Getting Started CD and in the EAServer 6.0 collection on the Sybase Product Manuals Web site at http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.eas_6.0/title.htm.

Conventions

The formatting conventions used in this manual are:

Formatting example	To indicate
commands and methods	When used in descriptive text, this font indicates keywords such as: • Command names used in descriptive text • C++ and Java method or class names used in descriptive text • Java package names used in descriptive text • Property names in the raw format, as when using Ant or jagtool to configure applications rather than the Management Console
variable, *package*, or *component*	Italic font indicates: • Program variables, such as *myCounter* • Parts of input text that must be substituted, for example: `Server.log` (with *Server* in italics) • File names • Names of components, EAServer packages, and other entities that are registered in the EAServer naming service
File | Save	Menu names and menu items are displayed in plain text. The vertical bar shows you how to navigate menu selections. For example, File | Save indicates "select Save from the File menu."

Formatting example	To indicate
`package 1`	Monospace font indicates: • Information that you enter in the Management Console, a command line, or as program text • Example program fragments • Example output fragments

Other sources of information

Use the Sybase Getting Started CD, the SyBooks CD, and the Sybase Product Manuals Web site to learn more about your product:

- The Getting Started CD contains release bulletins and installation guides in PDF format, and may also contain other documents or updated information not included on the SyBooks CD. It is included with your software. To read or print documents on the Getting Started CD, you need Adobe Acrobat Reader, which you can download at no charge from the Adobe Web site using a link provided on the CD.
- The SyBooks CD contains product manuals and is included with your software. The Eclipse-based SyBooks browser allows you to access the manuals in an easy-to-use, HTML-based format.

  Some documentation may be provided in PDF format, which you can access through the PDF directory on the SyBooks CD. To read or print the PDF files, you need Adobe Acrobat Reader.

  Refer to the *SyBooks Installation Guide* on the Getting Started CD, or the *README.txt* file on the SyBooks CD for instructions on installing and starting SyBooks.
- The Sybase Product Manuals Web site is an online version of the SyBooks CD that you can access using a standard Web browser. In addition to product manuals, you will find links to EBFs/Maintenance, Technical Documents, Case Management, Solved Cases, newsgroups, and the Sybase Developer Network.

  To access the Sybase Product Manuals Web site, go to Product Manuals at http://sybooks.sybase.com/nav/base.do.

Sybase certifications on the Web

Technical documentation at the Sybase Web site is updated frequently.

❖ **Finding the latest information on product certifications**

1. Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.
2. Select Products from the navigation bar on the left.

3 Select a product name from the product list and click Go.

4 Select the Certification Report filter, specify a time frame, and click Go.

5 Click a Certification Report title to display the report.

❖ **Creating a personalized view of the Sybase Web site (including support pages)**

Set up a MySybase profile. MySybase is a free service that allows you to create a personalized view of Sybase Web pages.

1 Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.

2 Click MySybase and create a MySybase profile.

Sybase EBFs and software maintenance

❖ **Finding the latest information on EBFs and software maintenance**

1 Point your Web browser to the Sybase Support Page at http://www.sybase.com/support.

2 Select EBFs/Maintenance. If prompted, enter your MySybase user name and password.

3 Select a product.

4 Specify a time frame and click Go. A list of EBF/Maintenance releases is displayed.

   Padlock icons indicate that you do not have download authorization for certain EBF/Maintenance releases because you are not registered as a Technical Support Contact. If you have not registered, but have valid information provided by your Sybase representative or through your support contract, click Edit Roles to add the “Technical Support Contact” role to your MySybase profile.

5 Click the Info icon to display the EBF/Maintenance report, or click the product description to download the software.

Accessibility features

EAServer has been tested for compliance with U.S. government Section 508 Accessibility requirements. The online help for this product is also provided in Eclipse help formats, which you can navigate using a screen reader.

The Web console supports working without a mouse. For more information, see “Keyboard navigation” in Chapter 2, “Management Console Overview,” in the *EAServer System Administration Guide*.

The Web Services Toolkit plug-in for Eclipse supports accessibility features for those that cannot use a mouse, are visually impaired, or have other special needs. For information about these features see the Eclipse help:

1 Start Eclipse.
2 Select Help | Help Contents.
3 Enter `Accessibility` in the Search dialog box.
4 Select Accessible User Interfaces or Accessibility Features for Eclipse.

Note You may need to configure your accessibility tool for optimal use. Some screen readers pronounce text based on its case; for example, they pronounce ALL UPPERCASE TEXT as initials, and MixedCase Text as words. You might find it helpful to configure your tool to announce syntax conventions. Consult the documentation for your tool.

For additional information about how Sybase supports accessibility, see Sybase Accessibility at http://www.sybase.com/accessibility. The Sybase Accessibility site includes links to information on Section 508 and W3C standards.

If you need help

Each Sybase installation that has purchased a support contract has one or more designated people who are authorized to contact Sybase Technical Support. If you cannot resolve a problem using the manuals or online help, please have the designated person contact Sybase Technical Support or the Sybase subsidiary in your area.

CHAPTER 1 Introduction

This document provides an overview of ways to improve performance for EAServer applications. There are many variables involved for application throughput and response times. In addition to tweaking the code in your application for optimum performance, you can tune EAServer based on application specifics as well.

This chapter describes key performance concepts, tools to test and measure performance, and techniques for measuring performance and identifying areas where your tuning efforts will have the greatest impact on overall performance.

The recommendations in this book are general guidelines. Results vary depending on the design of your application, hardware and network configuration, and other factors. For best results, you should monitor and measure performance as you fine-tune the configuration and application.

Determining factors

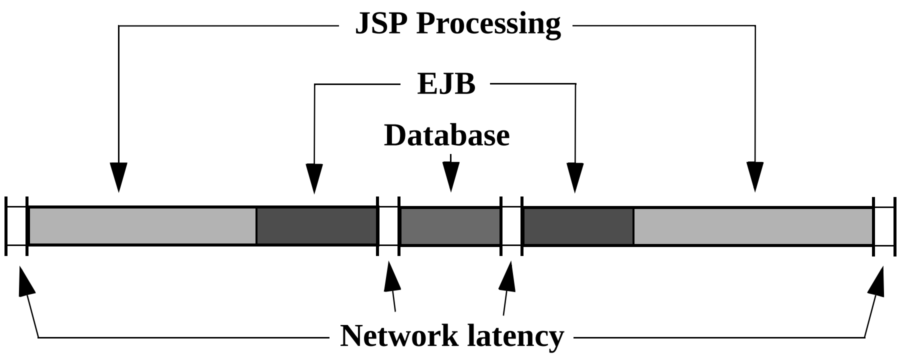
Several factors determine how well your application and server configuration perform.

Response time

Response time is the time required to execute a specified task, for example, to call an EJB method or submit a JSP form request. For end users, response time provides the key measurement of performance.

In client-side coding, you can minimize perceived response time by displaying partial results or status bars. However, in server-side coding, all you can do is minimize the in-server response time to an acceptable level. It helps to break down the response time into time spent in each component and subsystem. Figure 1-1 illustrates the processing of a Web form request to a JSP that calls an EJB component which in turn executes a remote database query. A slowdown can occur in any of these components. When tuning, you must isolate the part of your deployment that is causing the delay.

Figure 1-1: Response time breakdown



Scalability and throughput

Although a server configuration may perform well with a few users, response times can increase as the number of connected users increases. *Scalability* is a measure of how many simultaneous users your application and server configuration can support under prescribed use patterns before response times increase to unacceptable levels. *Throughput* is a measure of how many operations the server or application can process in a given time period; for example, database transactions per second or Web server page requests per second.

Throughput can be useful in comparing benchmark results for servers from different vendors, but scalability is a more useful measurement for tuning a given application deployment. You can directly measure the number of users and response times. End users are usually more concerned about how quickly their own work gets done than they are about overall server performance.

Memory use

Many performance optimizations in EAServer use *caching*: once created, objects such as component instances and database server connections are pooled for reuse, avoiding the overhead of re-creating the object. EAServer also caches servlet responses and static HTTP pages to avoid the overhead of running the servlet or reading files from disk, respectively. Caching reduces response time at the expense of increased memory use.

To maximize the performance gain from caching, Sybase recommends you run EAServer with as much memory as possible, from 1GB minimum for large deployments up to the limit of the machine architecture (2-3GB on most 32-bit address systems, since 1-2GB of a process's address space is reserved for use by the operating system).

Common performance problems related to memory use include:

- **Memory leaks** A *memory leak* occurs when code creates dynamically allocated objects but never releases them. In a Java or EJB component, you must set object references to null to release the memory associated with them. When using JDBC connections, you must release statement objects before releasing connections back to the data source pool (see “Clean up connections before releasing them to the data source” on page 87). Since EAServer pools and reuses component instances and data sources, a memory leak can slowly exhaust the available memory. You can diagnose and find memory leaks using a profiling tool—see “Profiling software” on page 7.

- **Swapping** Most operating systems support some form of virtual memory, which allows programs to address more memory than is physically available on the machine. Excess memory is mapped to data stored on disk. *Swapping* occurs when the system exchanges in-memory data for data stored on disk. Swapping should be avoided since the resulting disk I/O slows down the server. Memory leaks can cause swapping. If you have eliminated memory leaks, you can avoid swapping by ensuring that the machine has enough memory to support the EAServer configuration, and by making sure the system's per-process memory limit allows the server to use all of it. If you cannot increase physical memory, reduce the server's memory requirements by adjusting the parameters listed in "EAServer memory requirements" on page 31.
- **Object churning** Large, complex objects such as EJB components and database connections can take considerable time to allocate and construct. *Object churning* refers to repeated allocation and deallocation of the same object. For components, use instance pooling to avoid this phenomena, as described in "Instance pooling" on page 38. For database connections, use a data source. You can cache objects of other types within your component, servlet, or JSP class instance.

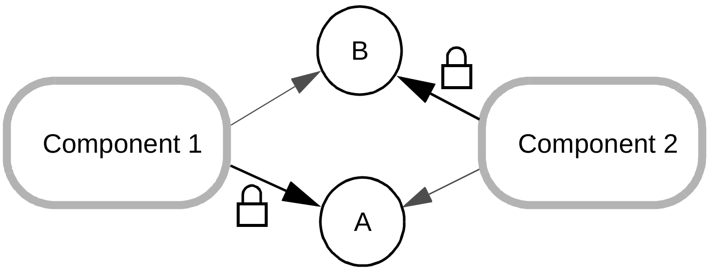
Threading

EAServer scales well, primarily through the use of native platform threads. Threading allows multiple components to execute concurrently with a minimum of context-switching overhead. Threading issues that affect performance include:

- **Number of threads** You can tune the number of EAServer threads for HTTP request handling. You can also assign *thread monitors* to components to limit the number of threads the component can be active in. More threads allow the server to handle more clients. However, if the number is too high, you may experience *thrashing*, which occurs when each thread gets so little execution time that more time is spent switching the thread context than running threads. You can avoid thrashing by reducing the number of threads, assigning thread monitors to components that cause thrashing, adding CPUs to a multi-CPU machine, or moving to a clustered EAServer deployment.

- **Concurrency** When different threads share data structures or resources, you must synchronize their execution so that access to the shared data or resource is *serialized*, that is, accessed by only one thread at a time. If access to the shared object is not serialized, you can cause *race conditions*, where overlapping modifications yield unpredictable results, often causing a crash due to the resulting nonsense data or resource state. However, excessive serialization can slow down the application by creating bottlenecks where many threads idle waiting to acquire synchronization locks. To avoid this problem, do not use design patterns that require synchronized code. When objects must be shared across threads, minimize synchronization and design carefully to avoid deadlock.
- **Deadlock** Deadlock occurs when two or more threads create recursive lock dependencies and wait indefinitely for each other to release the locks held. Figure 1-2 illustrates a deadlock scenario. Component 1 has locked object A while component 2 holds locks on object B. Now component 1 waits for B to be released while component 2 waits for A to be released.

Figure 1-2: Deadlock example



Deadlock is an extreme problem that can hang the server or at least the threads that are deadlocked. You can eliminate deadlock by carefully designing and following a locking protocol that avoids recursive dependencies when a component locks more than one object at once. For example, to lock the two objects in Figure 1-2, always lock A before locking B.

- **Thread binding** EAServer pools and reuses threads, allowing component instances to run on any thread rather than being tied to the same thread as a client connection or the thread that created the instance. Since most client connections have significant idle time, thread pooling allows fewer threads to serve more clients. However, if a component uses thread-local storage, each component instance must be bound to the thread that created it. Binding the thread significantly reduces scalability, since the thread cannot be used to run other instances and sits idle when the component is not running. For more information, see "Thread-related issues" on page 33.

Measurement and diagnosis tools

There are several tools available to measure the performance of your code and server configuration.

Instrumented code

In your code, add optional logic that you can enable to record timing information. Measure the execution time for major tasks such as:

- Component business method entry and exit
- Entry and exit of JSP or servlet service invocations
- Calls to other components or EJBs
- Database command execution and result-set processing
- Requests for cached connections
- JNDI lookups that return EJB proxies or JDBC data sources

EAServer includes built-in support for profiling and tracing of EJB business method invocations and Web component invocations—see "Component profiling and tracing" on page 9.

In your own Java code, you can record timings by calling System.currentTimeMillis(). Logging can degrade performance, so be sure to encapsulate the timing code in logic that allows you or your administrators to selectively enable tracing for areas where you are tuning. To allow configuration of the log options, you can use Log4j or the Java Logging package.

Profiling software

Profiling software measures the frequency of execution of each method or function in your code. Some profilers can also break down the execution time and memory use by each object.

EAServer includes built-in profiling support of public component methods—see "Component profiling and tracing" on page 9. You can also use third-party tools to gather profiling data for additional methods. Popular third-party software option include:

- OptimizeIt, from Borland at http://www.borland.com/. For detailed instructions on using OptimizeIt with EAServer, see Integrating OptimizeIt in Sybase EAServer, on the Sybase Web site at http://www.sybase.com/detail?id=1011357.
- JProbe, available from Quest Software at http://www.quest.com/jprobe/index.asp.

Load-testing tools

Load-testing software simulates multiple clients, allowing you to replicate real-world timings and server loads in your test environment. These tools typically allow you to run multiple scripted HTTP client sessions that simulate typical end user request patterns. Popular options include:

- OpenSTA, which is available on the Web at http://www.opensta.org/
- Segue silkperformer from Segue Software at http://www.segue.com
- Winrunner, Loadrunner, and other test tools from Mercury Interactive at http://www.mercuryinteractive.com
- e-TEST and other tools from Empirix at http://www.empirix.com/

Load-testing strategies
When setting performance goals, you must also specify a usage pattern that reflects real-world use of the application. For example, interactive users do not usually submit one request per second. A catalog shopper may download a part description, read it, download another, add it to the shopping cart, and so forth before checking out. To get accurate performance results, you must set up your test tools to mimic typical request patterns, including the "think time" between subsequent requests.

Memory and CPU usage monitors

You can monitor memory and process CPU time using system tools such as top on UNIX systems or the Task Manager or Performance Monitor on Windows. Many profiling tools such as OptimizeIt track memory and can help you find the source of memory leaks.

You can monitor memory usage of a running application server using the Management Console, as described in "CheckMemoryUsageTask tab" in Chapter 11, "Runtime Monitoring," in the *System Administration Guide*. You can also configure the server to log memory usage statistics hourly or per-minute using the Dump60MinuteMemoryUsage or Dump60SecondMemoryUsage scheduled tasks described in "Scheduled tasks for statistics collection" on page 9.

In server and listener properties, you can configure memory thresholds for each server as described in "Threshold monitor settings" on page 15.

In Java code, you can log the amount of free memory reported by the methods freeMemory() and totalMemory() in the java.lang.RunTime class to track total memory use in the Java dynamic allocation heap.

EAServer monitoring and tracing tools

EAServer includes these monitoring and tracing tools.

Runtime monitoring with the Management Console

The Management Console includes a runtime monitor that shows component, Web application, and data source statistics. For more information, see Chapter 11, "Runtime Monitoring," in the *System Administration Guide*.

Scheduled tasks for statistics collection

EAServer provides several scheduled tasks to collect performance data, including:

- Dump60MinuteMemoryUsage, which records memory usage statistics every hour
- Dump60MinuteStatistics, which records performance metrics every hour
- Dump60SecondMemoryUsage, which records memory usage statistics once per minute
- Dump60SecondStatistics, which records performance metrics once per minute
- PbHeap_dumpSummary, which periodically records the PowerBuilder VM heap manager memory usage
- SybHeap_dumpSummary, which periodically records the EAServer heap manager memory usage
- TxRef, which logs a transaction cross reference—see "Transaction cross-reference logging" on page 94.

EAServer logs memory usage statistics and transaction cross-reference statistics to the server log file. The other tasks log data to files in the *logs/statistics* EAServer subdirectory. These tasks are not enabled by default. You must install them in the server's scheduled task list as described in Chapter 3, "Using Scheduled Tasks," in the *Automated Configuration Guide*.

Component profiling and tracing

EAServer generates profiling and tracing code for deployed business and Web components.

Tracing is off by default and must be enabled in server properties and in the code generation options for EJB or Web modules. To enable tracing in server properties, configure the Enable EJB Trace and Enable Web Trace options in the server properties as described in "Log/Trace tab" in Chapter 3, "Creating and Configuring Servers," in the *System Administration Guide*. To enable or disable tracing for EJB and Web components, configure the following properties in the user configuration script for the module:

- For EJB components, the ejb.enableTracing Ant property globally enables or disables generation of tracing code for all components in the module. To disable or enable for a specific component, override the <setProperties> task for the component in the user-configuration script and set the nested <tracePublicMethods> property. For example:

```
<target name="configure-user">
  <setProperties component="ejb.components.myjar.MyCompRemote">
     <tracePublicMethods enable="true"/>
  </setProperties>
</target>
```

- For Web components, the web.enableTracing Ant property globally enables or disables generation of tracing code for all components in the Web application. You can override the setting for individual Web components using the same syntax shown above for EJB components.

To enable profiling of deployed components, you must enable generation of profiling code and not disable statistics collection in the server properties. Generation of profiling code is enabled by default in deployed components. The following properties can be changed in the generated configuration scripts to enable or disable generation of profiling code:

- For EJB components, the ejb.enableProfiling Ant property globally enables or disables generation of profiling code for all components in the module. To disable or enable for a specific component, override the <setProperties> task for the component in the user-configuration script and set the nested <profilePublicMethods> property. For example:

```
<target name="configure-user">
  <setProperties component="ejb.components.myjar.MyCompRemote">
     <profilePublicMethods enable="true"/>
  </setProperties>
</target>
```

- For Web components, the web.enableProfiling Ant property globally enables or disables generation of profiling code for all components in the Web application. You can override the setting for individual Web components using the same syntax shown above for EJB components.

In server properties, the Disable Statistics (disableStatistics) property must be false to allow collection of profiling statistics.

If you change the profiling or tracing settings for an EJB or Web module, recompile the module and restart the server for the change to take affect. If you change the server Disable Statistics, Enable Web Tracing, or Enable EJB Tracing properties, restart the server for the change to take affect.

When tracing or profiling is enabled, you can view statistics in the Web console or with a Web connection from your spreadsheet software. For more information, see “Viewing server statistics” in Chapter 11, “Runtime Monitoring,” the *System Administration Guide*.

Other trace-logging and statistics collection options

You can configure some EAServer subsystems to log trace data to the server log file, including:

- Thread monitors, to log performance data for the components to which you have assigned the monitor. See "Thread monitors" on page 35 for more information. You can assign thread monitors that have no thread limit to track the number of threads that a component is running on.
- Tracing of SQL commands run through a data source, enabled in server properties. See "Log/Trace tab" in Chapter 3, "Creating and Configuring Servers," in the *System Administration Guide*.
- Tracing of JMS commands, enabled in server properties. See "Log/Trace tab" in Chapter 3, "Creating and Configuring Servers," in the *System Administration Guide*.

Runtime monitoring APIs

EAServer includes several APIs that you can use to create your own monitoring applications, including:

- Jaguar::Monitoring provides methods to monitor the server state, connected users, and performance statistics such as the number of active and pooled component instances.
- Jaguar::PerfMonitor provides performance statistics in a per-second, per-minute, and per-hour bucket model for systems that have a statistics provider component installed. EAServer includes statistics providers for the connection caching and HTTP protocol handler subsystems. You can implement additional statistics providers for your application code using the Jaguar::StatProvider and Jaguar::StatProviderController interfaces.
- The logPerfManagerStats method in the Jaguar::Management interface reports statistics for components and network listeners that have monitoring thresholds configured for the EAServer Performance Monitor. For an example program that calls this method, see "Obtaining performance monitor statistics" on page 12.

For additional documentation of these APIs, see the generated HTML reference documentation in the *html/ir* subdirectory of your EAServer installation.

Obtaining performance monitor statistics

To obtain performance monitor statistics, call the logPerfManagerStats method in the *Jaguar/Management* built in component. The code below is a sample Java client program to call this method:

```
// PerfDump.java
// This program is supplied on as is basis
// without any guarantees.
// This Program is not guaranteed to by
// Sybase to produce required results
// any or all of the time.
//
// Usage: java PerfDump iiop://<hostname>:<iiop port#>

import org.omg.CORBA.*;
import SessionManager.*;
import com.sybase.jaguar.system.*;

public class PerfDump
{
    public static void main(String[] str)
    {
        try {
            java.util.Properties props = new java.util.Properties();
            props.put("org.omg.CORBA.ORBClass", "com.sybase.CORBA.ORB");
            ORB orb = ORB.init((String[])null,props);
            Manager manager = ManagerHelper.narrow(orb.string_to_object(str[0
]));
            Session session = manager.createSession("jagadmin","");
            Management _mg = ManagementHelper.narrow(session.create("Jaguar/M
anagement"));
            try {
                 _mg.logPerfManagerStats();
            }catch (Exception ex)
            {
                ex.printStackTrace();
            }
        }catch (Throwable th)
        {
           th.printStackTrace();
        }
        System.out.println("Now Refer to EAServer's Log file for Performance
Monitor Information");
    }
}
```

EAServer writes statistics to the server log file. These include statistics for each component and stack traces for each thread. Statistics for each component include the component name, number of current active instances, number of instances waiting to execute and average response time. A response time value of -1.00 indicates that the component is not being monitored. Here is example output:

```
Apr 15 20:45:32 2004: [0000004692] ******** PERFORMANCE MONITOR STATISTICS START
********
Apr 15 20:45:32 2004: [0000004692] Name                          Active     Waiting
Response
Apr 15 20:45:32 2004: [0000004692]                           Instances   Instances
Time
Apr 15 20:45:32 2004: [0000004692] -------------------------------------------
---------------
Apr 15 20:45:32 2004: [0000004692] CosNaming/JNameService     00000      00000
-1.00
Apr 15 20:45:32 2004: [0000004692] Jaguar/HttpStatProviderCon 00000      00000
-1.00
Apr 15 20:45:32 2004: [0000004692] JaguarOTS/OtsService       00000      00000
-1.00
Apr 15 20:45:32 2004: [0000004692] CtsComponents/MessageServi 00032      00000
-1.00
Apr 15 20:45:32 2004: [0000004692] -------------------------------------------
---------------
```

The stack trace listing shows the execution stack for each thread. When you suspect a deadlock condition, the stack shows which entity is being blocked and the calling sequence that caused the block. As show in the stack trace below, the `***BLOCKED***` token is printed when the entity execution is blocked. In this example, the call is blocked when trying to execute the *j2eebookstore/customer* component. The stack sequence indicates that this component is recursive: *j2eebookstore/customer* has called *j2eebookstore/customer*. Also from the stack it is evident that the client is connected to the port defined by Jaguar_iiop listener:

```
Apr 15 20:45:32 2004: [0000004692] *************STACK TRACES START
***************
Apr 15 20:45:32 2004: [0000004692] Thread:134938976
Apr 15 20:45:32 2004: [0000004692] CtsComponents/MessageService
Apr 15 20:45:32 2004: [0000004692] CtsComponents/MessageThread
Apr 15 20:45:32 2004: [0000004692] -------------------------------------------
Apr 15 20:45:32 2004: [0000004692] Thread:134642792
Apr 15 20:45:32 2004: [0000004692] *******BLOCKED*******
Apr 15 20:45:32 2004: [0000004692] j2eebookstore/customer
Apr 15 20:45:32 2004: [0000004692] j2eebookstore/customer
Apr 15 20:45:32 2004: [0000004692] Jaguar_iiop
Apr 15 20:45:32 2004: [0000004692] -------------------------------------------
```

The tuning process

Tuning requires extensive testing to isolate bottlenecks and fix them. You must be systematic and test each potential fix as it is applied. Trying to fix multiple issues at once may introduce new problems. Use the tools described in this chapter to test and tune as described below.

❖ **The tuning process**

1. Load test under expected peak load conditions, using a tool configured to mimic the typical request timings expected in production.
2. Find and fix any memory leaks and deadlocks. These problems may be discovered now if you have not load-tested before.
3. Identify problem areas in your code or configuration.
4. Focus efforts on tuning the relevant EAServer settings or application code. After each code or configuration change, repeat your functional tests to verify that the application still returns correct results, then repeat the performance test to check for improvement.

   Try to identify where your tuning efforts will yield maximum gain. If tuning business logic or Web components, focus on the components and methods that are invoked most often. For example, it is better to shave 100 milliseconds from a method that is called twice a second than to shave 1 second from a method that is called once a minute. The latter optimization saves 60 seconds an hour, while the former saves 720.

C H A P T E R 2

Server Tuning

This chapter describes how to tune server, Java virtual machine, and system properties for the best server performance.

Threshold monitor settings

EAServer provides threshold-based memory and response time monitoring to prevent degradation of server performance under extreme load conditions. You can configure these settings to heuristically govern the processing requests to prevent performance degradation due to overuse of available resources.

How threshold monitoring works

Ideally, an applications request rate relates linearly to response rate as shown in Figure 2-1.

However, performance of any application depends on availability of resources like CPU, memory, network connections, and swap space. These resources are limited, and when they are exhausted, the response rate degrades. Due to resource limits, the response rate is expected to level off when the number of incoming requests reaches the point where resources are exhausted, as shown in Figure 2-2. However, in practice, an unlimited increase in incoming requests can cause performance to degrade; the response rate can drop in this case. In extreme cases, the application may run out of memory and abend or hang.

Figure 2-1: Ideal response rate curve

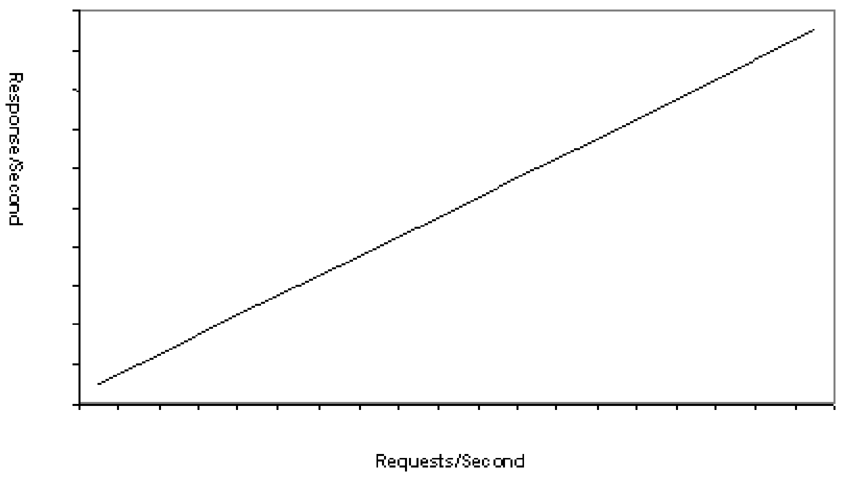
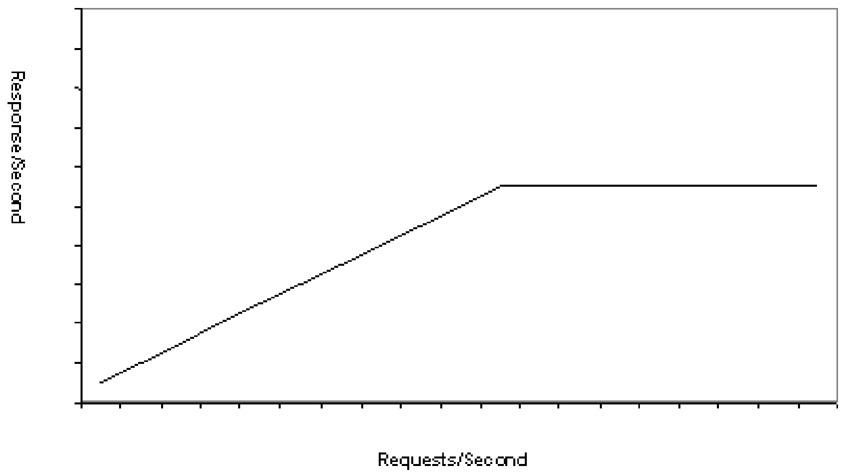


Figure 2-2: Expected response rate curve



Threshold monitoring allows you to configure the system to operate at a constant response rate and avoid out-of-memory conditions under high load conditions. EAServer uses these algorithms to heuristically govern the request rate when high load conditions are detected:

- **Memory monitoring** You can configure thresholds for memory usage. EAServer monitors the memory used and throttles external requests when the critical threshold is reached.
- **Response time monitoring** You can configure expected average response times for network requests and component method invocations. EAServer keeps a running average of the actual response time, and throttles handling of new requests when the average response time rises above the configured threshold. When the average response time drops below the threshold, the throttle is removed.

Threshold monitoring requires associating a thread monitor with the performance constraint to be monitored. Under stress conditions, the thread monitors Maximum Active Threads property throttles the load. When the performance threshold is crossed, new client requests must acquire the thread monitor before proceeding. If the thread monitor has no available threads, the request blocks temporarily until a thread is available. You can modify the thread monitor’s Maximum Active Threads property to tune the response rate curve.

Configuring threshold monitoring for servers or listeners

In server properties, or the properties for each listener, you can configure threshold monitors to queue handling of new requests when a response-time or memory-limit threshold is reached. For example, you can specify that when 90% of available memory is consumed, new requests must acquire the "memory" thread monitor before proceeding.

In the Management Console, configure the threshold monitoring properties on the Monitors tab of the server properties pages, or on the Performance tab of the listener properties pages. For details, see "Monitors tab" in Chapter 3, "Creating and Configuring Servers," in the *System Administration Guide*.

To be effective, threshold monitoring requires thread monitors with fixed limits on the Maximum Active Threads setting. The default thread monitor for response-time threshold monitoring is "performance." The default for memory threshold monitoring is "memory." To configure thread monitors, follow the instructions in "Monitoring threads" in Chapter 3, "Creating and Configuring Servers," in the *System Administration Guide*.

Configuring threshold monitoring for components

You can configure response-time monitoring for EJB, CORBA, PowerBuilder, and Web components.

For components, threshold monitoring requires a thread monitor with a hard thread limit. When the response-time threshold is reached, subsequent calls to the component must acquire the thread monitor before proceeding. After the average response time drops below the threshold, new invocations are allowed to proceed normally. To define a thread monitor, follow the instructions in "Monitoring threads" in Chapter 3, "Creating and Configuring Servers," in the *System Administration Guide*. Set the thread limit to a low number, for example 10. You can also define the thread monitor in an Ant configuration file, for example:

```
<project name="MyThreadMonitor">
  <import file="ant-config-tasks.xml"/>
  <target name="configure">
    <setProperties threadMonitor="MyThreadMonitor">
      <property name="maximumActiveThreads" value="10"/>
    </setProperties>
  </target>
</project>
```

After you have defined a thread monitor, threshold monitoring can be applied with the <performanceMonitor> Ant command run inside the <setProperties> task. For example:

```
<setProperties component="ejb.components.myjar.MyCompRemote">
  <performanceMonitor maxResponseTime="5000" threadMonitor="MyMonitor"/>
</setProperties>
```

The `maxResponseTime` value specifies the maximum allowed response time in milliseconds. The `threadMonitor` value specifies the thread monitor that must be acquired by new requests when the average response time exceeds the maximum. The `component` value specifies the name of the DJC component that runs the application component that you want to monitor. Set this depending on the application component type, as follows:

- For enterprise JavaBeans components, specify the DJC component that corresponds to the remote or local interface. If configuring an entity bean with finder methods that perform heavy processing, consider applying the same performanceMonitor configuration to the DJC component that corresponds to the home interface. You can read the DJC component names from the EJB module's Ant configuration file that was generated by deployment.
- For CORBA and PowerBuilder components, specify the DJC component that corresponds to the remote interface of the EJB session bean that wraps your component. This name is `ejb.components.`***`package`***`.`***`component`***`Remote`, where *package* is the CORBA package name, and *component* is the component name.
- For Web components, specify the DJC component that corresponds to the servlet or JSP that the settings apply to. You can read the DJC component names from the Web application's Ant configuration file that was generated by deployment.

Tuning response rate thresholds

Tune the response rate threshold settings when your performance testing indicates a severe degradation of server performance under load. When tuning, consider the following:

- **Deciding where to apply the settings** You can apply response time thresholds to components, servers, or listeners. Choose the entity that has the greatest affect on client load and that has the least unintended effects on other applications running in the same server. For example:

- Apply memory-threshold settings to the server to govern all client requests, regardless of type.
- If the application front end is a Web application, you can apply response-time or memory-threshold settings to your HTTP listener since all client requests pass through it.
- If clients connect to EJB session beans, apply the settings to the session bean components or the IIOP listener, so that the number of session bean instances governs the applied load.
- If a component accesses a database, and the client load tends to overwhelm the database, apply a response-time threshold to this component to throttle the database load to manageable levels. (You could also tune the data source size as described in “Data source settings” on page 89.)

Warning! When applying response rate thresholds to components, some configurations can introduce the possibility of deadlock—see “Avoiding deadlock scenarios” on page 20.

- **Choosing a response time threshold** Use performance testing under controlled client loads to determine a realistic value for response times under high load conditions. Apply this setting as the allowable Maximum Response Time for the component or listener. This setting prevents response times from growing exponentially under worst-case load conditions; it does not make the server run faster.

Avoiding deadlock scenarios

When you apply response time thresholds to components, the server may deadlock if the thread monitors used do not allow nesting.

The thread monitor Allow Nesting (allowNesting) property specifies whether thread monitor limits are applied when a thread already holds another monitor. For example, if a thread has already acquired monitor A, nesting allows the thread to acquire monitor B even if B's usage level is above the maximum. Nesting is allowed by default.

Allowing nesting prevents deadlock. Deadlock may occur in some intercomponent call scenarios if you disable thread monitor nesting for the thread monitor used in response-time threshold monitoring. If you suspect your components may be deadlocked due to response-time monitoring, analyze the stack traces in the performance monitor statistics.

Thread settings

The server threading properties affect the number of clients that can be served simultaneously and the memory used by each executing thread. In addition to the properties discussed below, you can also configure the size of the thread pool for HTTP request handling in listener properties—see “HTTP thread pool size” on page 27.

Thread stack size

In EAServer, the thread stack size property determines the amount of memory reserved for the call stack associated with each thread. The stack size must be sufficient to allow for nested intercomponent calls. However, if the stack size is too large, memory is wasted.

The default stack size is 256K on UNIX and on 32-bit Windows operating systems. This is appropriate for almost all situations, and provides adequate reserve memory for the largest case loads that have been tested by Sybase engineering and customers.

For production servers that see heavy use from large numbers of clients, you may want to decrease the stack size from the default value. Doing so can make the per-thread stack memory available for other uses. However, you must first run load tests on a test server to ensure that the stack size is adequate for the components running on the server. If the stack size is too small, client requests may fail with thread stack overflow errors, which are recorded in the server log.

Sybase recommends that you do not reduce the stack size if you run:

- Components that call third-party DLLs or shared libraries
- Java components that call native classes (including JDBC drivers that call out to native libraries)

In EAServer, the thread stack size is a Java virtual machine startup option. You can configure it as described in “JVM memory allocation parameters” on page 24.

Debug and trace settings

While useful for diagnosing configuration or code problems, debug and trace properties can reduce application performance when data is logged needlessly. Disable any debug or tracing properties unless you are actively diagnosing a related problem.

Java virtual machine tuning

These settings tune the Java virtual machine (JVM) that runs Java code in the server. These settings have a large effect on applications that are implemented with Java, EJB, or Web components. Since many of the server internal components are implemented in Java, these settings have some effect on applications that are implemented in other languages such as PowerBuilder.

CLASSPATH and BOOTCLASSPATH settings

Check the CLASSPATH and BOOTCLASSPATH for the server to ensure that they does not include unnecessary entries. You can check the runtime values in the server log file.

The server start scripts assemble the CLASSPATH from the required files in the EAServer installation and existing settings from your environment. The scripts then set BOOTCLASSPATH to include the CLASSPATH settings.

You can set these variables in the *bin\local-setenv.bat* file (for Windows platforms) or *bin/local-setenv.sh* script (for UNIX platforms). Create the file if it does not already exist. If you require no additional CLASSPATH or BOOTCLASSPATH entries, unset these variables. Otherwise, set them to include the minimum required settings.

Custom class lists

EAServer uses custom Java class loaders to allow refreshing the Web application classes and Java components, and to load classes from directories and JAR files that are not specified in the CLASSPATH environment variable. During the development cycle, this feature allows you to add or modify classes without restarting the server. However, duplicates in the custom class list for different components can waste memory by loading duplicate class instances. Chapter 10, “Configuring Java Class Loaders,” in the *System Administration Guide* describes how to configure common class lists for components and Web applications.

Java VM type and version

EAServer supports multile JDK versions, and each JDK version can support multiple VM types such as Server Hotspot, Client Hotspot, and Classic. You specify the JDK version and VM type when starting the server, or for servers that run as Windows services, with the command that you run to install the service. For details, see “Starting the server” in the *EAServer System Administration Guide*.

As a general rule, you should use the Server Hotspot VM in the latest supported JDK version. However, always consult the *EAServer Release Bulletin* for your platform for updated recommendations. For more information on Java Hotspot technology, see the Sun Microsystems white paper Java HotSpot Performance Engine Architecture at http://java.sun.com/products/hotspot/whitepaper.html.

Some applications may run significantly faster if compiled and run with JDK 1.5. JDK 1.5 provides the StringBuilder class as a faster alternative to StringBuffer. JSPs and other code that makes extensive use of string concatenation can run faster in JDK 1.5 due to the use of StringBuilder. To realize the performance benefits of JDK 1.5, you must compile your applications with JDK 1.5, and specify the `-jdk15` option when deploying them to EAServer. For more information on deploying, see the deploy reference page in Chapter 12, “Command Line Tools,” in the *System Administration Guide*.

Switching back to JDK 1.4 from JDK 1.5
Classes compiled with JDK 1.5 cannot be run in JDK 1.4 or earlier JDK versions. Since EAServer generates new classes for deployed applications, applications deployed with JDK 1.5 may not run if the server is restarted with JDK 1.4. To switch back to JDK 1.4 from JDK 1.5, undeploy and redeploy any applications that were deployed to the server running in JDK 1.5. Applications deployed to JDK 1.4 must also be compiled with JDK 1.4 (or with JDK 1.5 while specifying JDK 1.4 compatiblity with the `-source 1.4` option to javac).

Just-in-time compilation

The Java just-in-time (JIT) compiler converts Java bytecode into native machine code, which can run much faster than the interpreted bytecode. The JIT compiler is enabled by default.

JVM memory allocation parameters

The Java virtual machine uses its heap storage for dynamic allocation memory. In addition, each thread requires reserved memory for the stack used to pass method parameters. These parameters must be configured in the Java virtual machine startup options.

Table 2-1 describes the JVM memory allocation options. Configure these parameters for the server by setting the indicated environment variables in the *bin\local-setenv.bat* file (for Windows platforms) or *bin/local-setenv.sh* script (for UNIX platforms).

Table 2-1: JVM memory allocation parameters

Parameter	Description
`-Xmx`*`MaxHeap`*	Specifies the maximum heap size. *MaxHeap* is the heap size value specified using the syntax in Table 2-2. The JVM reserves this much memory at start-up. The memory used for object allocation cannot exceed this amount. If the heap size is exceeded, you see request failures accompanied by java.lang.OutOfMemoryError errors in the error log. To set this property, set the DJC_JVM_MAXHEAP environment variable to the heap size value specified using the syntax in Table 2-2.

Parameter	Description
`-Xms`*`MinHeap`*	Specifies the minimum, or initial heap size. *MinHeap* is the heap size value specified using the syntax in Table 2-2. While the maximum size is reserved at start-up, only the minimum size is monitored and allocated from by the JVM. On production servers, set this value to the same size as the maximum heap size. The maximum heap size is reserved at server start-up regardless of the minimum size, and using equal sizes avoids the CPU overhead of dynamically growing the heap. To set this property, set the DJC_JVM_MINHEAP environment variable to the heap size value specified using the syntax in Table 2-2.
`-Xss`*`StackSize`*	Configures the stack size for Java threads. *StackSize* is the amount of virtual memory reserved for the stack of each Java thread. To set this property, add it to the Java Startup Options server property in the Management Console (or the javaStartupOptions property if using Ant configuration); this property specifies other JVM startup options as they are passed on the Java command line.

The optimum heap size depends on your application and machine configuration. To tune the value, first verify that you have removed any memory leaks from your own code. Then test under expected peak load conditions to determine the minimum size that allows the application to run without errors. If the heap size is too large, it uses memory that could otherwise be used for the call stack required to run each thread. Large heap sizes can also incur a larger delay when the Java garbage collector runs. Never set the heap size larger than the machine's physical memory; if you do, the system will swap memory to disk. Set the minimum and maximum sizes to equal values, specified in bytes, kilobytes, or megabytes, as described in Table 2-2.

Table 2-2: Syntax for Java heap size values

Heap size value syntax	To indicate

`n``M`
or
`n``m`
	n megabytes, for example:
	`512M`

`n``K`
or
`n``k`
	n kilobytes, for example:
	`1024K`

Heap size value syntax	To indicate
`n`	*n* bytes, for example: `536870912`

Set the Java thread stack size to the smallest value that still allows the application to run. Usual values are 256K or 512K for the applications used for internal stress testing at Sybase. Most applications should never require more than 1Mb. The stack must be large enough to accommodate parameters passed in component dispatcher and intercomponent calls. However, if the value is too high, it limits the maximum number of threads that can be spawned. To run *N* threads, there must at least *N* x *StackSize* of free memory available.

Other Java VM settings and troubleshooting

You can configure additional Java VM options by adding them to the Java Startup Options server property in the Management Console (or the javaStartupOptions server property if using Ant configuration). This property specifies other JVM startup options as they are passed on the Java command line.

Alternatively, add Java startup options to the value of the DJC_USER_JVM_ARGS environment variable. Set this environment variable in the *bin\local-setenv.bat* file (for Windows platforms) or *bin/local-setenv.sh* script (for UNIX platforms).

To verify the Java VM options, check the server log file. The server logs all the options at start-up, including those that are configured internally and by the heap size settings discussed above.

Listener tuning

EAServer includes several preconfigured listeners, described in “Preconfigured listeners” in the *EAServer System Administration Guide*. Remove any listeners that you do not need. For example, if your application does not need to support RMI clients or SSL clients, remove these listeners. Unused listeners waste memory and network resources.

HTTP thread pool size

For HTTP listeners, the Maximum Threads property specifies the size of the thread pool used to run HTTP requests. This number can be less than the number of client connections. EAServer runs an HTTP requests > by pulling a thread from the pool to run the request. This happens when too many clients try to send requests at the same time. The default setting of -1 indicates that the thread pool size is the default of 256.

This setting does not apply to listeners for other network protocols.

If the thread pool is too small, you may see `[SocketListener] LOW ON > THREADS` errors in the log. To determine how many HTTP threads are required, check the request pattern in the request log for indications of a heavily loaded server. Adjust the maximum thread setting as necessary. Ideally, this setting should be 10 – 20% more than the number of simultaneous HTTP requests that you expect to handle. (The additional threads accommodate the use of threads in Web browsers to submit simultaneous requests for images and text). A value that is too low can increase HTTP response time by causing requests to block while waiting for a thread. A value that is too high wastes available threads that could be used for other purposes.

Connection request backlog pool size

You can configure the size of the pool used to handle outstanding connection requests as the Listen Back Log setting in the Management Console Listener Properties pages. If using Ant configuration, set the listenBackLog property for the socketListener entity. If this property is not set, the default is 10.

When the server is very busy, all available threads may be in use when a connect request arrives. These pending connect requests are pooled until they can be handled. If the pool size is too small, client connection requests may time out before the server can handle the request.

You can configure different request pool sizes for different protocols. For example, if the server is handling mostly HTTP requests, you can increase the request pool size for the HTTP listener while leaving the IIOP request pool size at a low value.

The connection request pool size affects the server memory requirements:

mem = entries * 20K

That is, each entry requires about 20K of memory reserved at server start-up.

Operating system settings

These operating system settings can affect EAServer performance. For additional information on system requirements, see the *EAServer Release Bulletin* for your platform.

UNIX file descriptors

On UNIX, concurrent client connections to EAServer are limited by the operating system limit for the number of file descriptors that can be opened in one process. Before you start the server, set the file descriptor limit in the shell where you will start the server as follows:

1. Use a text editor to open the *bin/local-setenv.sh* file. Create this file if it does not exist.
2. Add the following line to specify the number of descriptors:

```
    ulimit -n NNNN
```

Where *NNNN* is the number of descriptors. See your UNIX documentation for more details on the ulimit command. On some systems you may need to also adjust the system-wide limit. For example, to use more than 1024 descriptors on Solaris, you must modify the */etc/system* file and modify the rlim_fd_max setting, for example:

```
set rlim_fd_max = 4096
```

Per-process memory limits

On some systems, the default configuration limits the memory available to the server. You may need to raise the limit to make best use of memory intensive features such as caching or a large Java heap. For more information, see:

- Your operating system documentation for details on per-process limits
- “EAServer memory requirements” on page 31
- For AIX systems, the Sybase technical document Configuring Memory Parameters on AIX for EAServer at http://www.sybase.com/detail?id=1024625

Factors that affect start-up and shutdown time

These settings affect how long it takes to shut down and restart the server.

Start-up performance

These settings affect how long it takes the server to start, that is, the time between starting the process and when the server is ready to accept connections:

- **Message service initialization** If you are running the message service, it must initialize before the server can accept connections. At start-up, the message service reads unprocessed persistent messages into the in-memory cache. A large message backlog can delay server start-up.
- **Service components and scheduled tasks** All service components must return from their start methods before EAServer accepts client connections. Scheduled tasks that run at startup must also complete. Lengthy processing in the service start method or in a scheduled task can delay server start-up. For more information, see Chapter 3, “Using Scheduled Tasks,” and Chapter 4, “Creating Service Components,” in the *Automated Configuration Guide*.
- **Modules loaded at start-up** To run in a server, Web and EJB modules must be installed in the server’s start modules list. To speed up server startup, remove modules that you are not using by configuring the Modules tab settings in the server property pages in the Management Console. For details, see “Modules tab” in Chapter 3, “Creating and Configuring Servers,” in the *System Administration Guide*.

  If you do not use the Management Console, that is, you do all configuration with Ant, you can remove the Management Console from the server’s start module list by including application-console in the setting of the server systemExcludeModules property. For example:

```
<project name="MyApplicationServer">
  <import file="ant-config-tasks.xml"/>
  <target name="configure">
    <setProperties applicationServer="MyApplicationServer">
      <property name="systemExcludeModules"
                value="application-console"/>
    </setProperties>
  </target>
</project>
```

- **JSPs loaded at start-up** JSPs that are configured to load at start-up are compiled if necessary. Compilation of many JSPs can delay start-up. If you define all JSPs that need to be precompiled with their own “servlet” elements in your *web.xml* files, you can precompile them when deploying to EAServer. Otherwise, precompile them using the jagtool compilejsp command.
- **Servlets loaded at start-up** Servlets that are configured to load at start-up must return from their init method before the server continues. Lengthy processing in this method can delay start-up. (If the servlet does not load at start-up, lengthy processing in this method can delay the response to the first client request).

Shutdown performance

These settings affect how long it takes the server to shut down.

Pooled component destruction

EAServer explicitly destroys pooled component instances before the server shuts down. This allows you to perform cleanup operations in your component, such as closing database connections.

Servlet destruction

EAServer calls each servlet’s destroy method before shutting down or after you have refreshed or stopped the servlet using the Management Console. If service calls are still active, the servlet is not destroyed until they complete.

The Destroy Timeout setting specifies the number of seconds that the server should wait for the service calls to return before calling the destroy method. The default behavior specifies that the server wait indefinitely for service calls to return.

EAServer memory requirements

The following configuration settings affect EAServer's memory requirements. While exact memory requirements depend on your component and servlet implementations, this list tells you what options you can tune to affect the server's memory footprint:

- **Java heap sizes** The Java Virtual Machine (JVM) that EAServer uses to run Java code has parameters to size its dynamic memory allocation heap. For more information, see "JVM memory allocation parameters" on page 24.
- **The number of threads and thread stack size** Each thread requires a small amount of reserved memory to store the stack for code running in the thread. "JVM memory allocation parameters" on page 24 describes how to configure the thread stack size.
- **HTTP response cache sizes** EAServer supports several forms of HTTP response caching. For more information, see "Understanding HTTP response caching options" on page 71.
- **Entity bean instance and query caching** EAServer can cache instance data and finder-method results for EJB-CMP entity beans. See "Configuring object and query caching" on page 62 for more information.
- **Data source pool sizes** You can configure the number of connections stored in each cache as described in "Tuning the pool size" on page 89.
- **Component instance pool sizes** You can configure the pool size for component instances as described in "Instance pooling" on page 38. The memory required for each instance depends on your implementation.
- **Custom class lists** EAServer uses custom Java class loaders to allow you to refresh the Web application classes and Java components, and to load classes from directories and JAR files that are not specified in the CLASSPATH environment variable. During the development cycle, this feature allows you to add or modify classes without restarting the server. However, duplicate entries in the custom class lists for different components waste memory by loading duplicate class instances. Chapter 10, "Configuring Java Class Loaders," in the *System Administration Guide* describes how to configure common class lists for components and Web applications.

- **Use of the hot refresh feature** Refreshing PowerBuilder and C++ components loads additional copies of the implementation classes. EAServer leaves the previous implementation in memory for use by existing client sessions. For this reason, it is best to restart your production server after deploying a large number of PowerBuilder or C++ component updates. If you have a maintenance window when the server can be restarted, redeploy your changed code at this time to allow a restart of the server. When you do refresh, do so at the lowest level possible. For example, if you modified a component, refresh the package that it is installed in rather than the whole server.

CHAPTER 3 Component Tuning

Common component performance issues

These recommendations apply to all components, regardless of their type. Follow these suggestions for every component, in addition to the ones provided for specific component types.

Tracing and debugging settings

Tracing properties enable additional logging, which can be useful when debugging problems. However, tracing requires additional file I/O and computation. For best performance, disable all tracing and debugging properties.

Thread-related issues

EAServer is scalable because it is multithreaded and multiprocessor-safe, using a thread pooling model to run components invoked from the Web tier. Ideally, a component:

- Supports thread pooling, to run on any thread rather than being tied to the same thread as a client connection. Since most client connections have significant idle time, thread pooling allows fewer threads to serve more clients.

- Supports concurrent execution, allowing multiple instances of the implementation class to be invoked simultaneously to service different clients.

These settings affect the threaded execution of your component.

Bind thread

For CORBA and PowerBuilder components, the Bind Thread option specifies whether component instances must be bound to the thread that creates the instance.

This option decreases scalability. Twice as many threads are needed to run the component, since each instance requires the client thread plus another thread bound to the component. Also, while the thread is bound to the instance, it cannot be pooled and used to service requests involving other components.

Enable this option only for PowerBuilder components if required (see “PowerBuilder component performance” on page 44) and CORBA/Java and CORBA/C++ components that use thread-local storage. Otherwise, disable this feature so EAServer can run the component on any available thread.

Enterprise JavaBean components implemented according to the EJB specification do not require this setting.

Concurrency

For CORBA and PowerBuilder components, the Concurrency setting specifies whether component instances can execute concurrently on multiple threads.

Enable this option for any component that is thread-safe. Concurrent access can decrease the response time of client method invocations. If this option is disabled, EAServer serializes all method calls to the component. Concurrency applies to execution of all instances. With concurrency disabled, a call to one instance cannot overlap the execution of another instance.

If the Sharing setting is enabled for your PowerBuilder component, disable the Concurrency setting. PowerBuilder is thread-safe at the session level only. For other component types, concurrency requires that your implementation be thread-safe. The requirements depend on the value of the Sharing setting as described in Table 3-1.

Table 3-1: Coding requirements to support concurrency

Sharing enabled?	Coding requirements
No	Protect any static instance variables; synchronize access to them to prevent concurrent access from different threads. Exceptions to this rule are read-only static variables, such as those that include the final modifier (meaning it is a constant that cannot be changed), and static variables defined as a primitive datatype that is 32 bits or less.
Yes	Same as the above, but you must protect all instance variables since one instance is called by multiple threads.

If you enable the Concurrency setting for a component that does not meet these requirements, you may encounter hard-to-diagnose threading errors such as race conditions. In a *race condition*, multiple threads update the same data simultaneously. The outcome of conflicting updates is unpredictable and may cause crashes or incorrect results.

Sharing

For CORBA components, if the Sharing setting is enabled, a single instance serves all client requests.

For best performance, this option requires that you also enable the concurrency option. However, if your component has read-write static or instance variables, you must synchronize all access to them. This can create bottlenecks where threads wait to access synchronized data or methods. Also, in a cluster, the component is not a true singleton object: while one instance runs per server, multiple instances run in the cluster, one instance per server. Consider these limitations carefully before adapting the sharing/singleton pattern if your implementation has read/write static or instance variables.

You can use sharing and concurrency without synchronization if your implementation has no read/write static or instance variables. This can reduce memory use since only one instance is loaded. However, the effect is likely to be negligible unless the implementation class is very large.

Thread monitors

Thread monitors provide a means to limit the execution time devoted to specified components and component methods. You can assign components and methods to a thread monitor to ensure that no more than a specified maximum number of threads will be active at any point executing the methods and components assigned to the monitor.

You can also use thread monitors without a limit on the number of threads. Doing so allows you to use the monitor trace properties to record performance data.

Alternatives to thread monitors
As an alternative to configuring thread monitors to govern component load, you can configure response-time threshold monitoring for your application components or network listeners. For more information, see “Configuring threshold monitoring for components” on page 18

❖ **Creating or configuring a thread monitor**

- Follow the instructions in “Monitoring threads” in Chapter 3, “Creating and Configuring Servers,” in the *System Administration Guide*.

❖ **Assigning a component or method to a thread monitor**

- Create a user configuration script that configures the component properties and calls the <threadMonitor> Ant task. For example:.

```
<setProperties component="ejb.components.myjar.MyCompRemote"
merge="true”>
  <threadMonitor name="MyMonitor"/>
</setProperties>
```

The `threadMonitor` value specifies the thread monitor that must be acquired by new requests when the average response time exceeds the maximum. The `component` value specifies the name of the DJC component that runs the application component that you want to monitor. Set this depending on the application component type, as follows:

- For enterprise JavaBeans components, specify the DJC component that corresponds to the remote or local interface. If configuring an entity bean with finder methods that perform heavy processing, consider applying the same thread monitor configuration to the DJC component that corresponds to the home interface. You can read the DJC component names from the EJB module’s Ant configuration file that was generated by deployment.
- For CORBA and PowerBuilder components, specify the DJC component that corresponds to the remote interface of the EJB session bean that wraps your component. This name is `ejb.components.`***`package`***`.`***`component`***`Remote`, where *package* is the CORBA package name, and *component* is the component name.

- For Web components, specify the DJC component that corresponds to the servlet or JSP that the settings apply to. You can read the DJC component names from the Web application's Ant configuration file that was generated by deployment.

Stateful versus stateless components

A component that remains bound to a client instance between consecutive method invocations is called a *stateful component*. A component that can be unbound from the client after each method call is said to be a *stateless component*. Typically, an application built with stateless components offers the greatest scalability.

Performance benefits of the stateless model

Applications that use stateless components generally perform better. In the stateless model, each instance can serve different clients during the "think time" that is typically seen in interactive applications. In the stateful model, each client requires a dedicated component instance for the lifetime of the client session. Resources associated with the instance state remain tied up during the user's idle time.

To develop stateless Enterprise JavaBeans component, follow the stateless session bean model described in the EJB specification.

For CORBA and PowerBuilder components, you can either configure stateless behavior, or code the component to call the appropriate lifecycle control method to unbind the component instance from the client reference in each business method. For more information, see Chapter 2, "CORBA Component Life Cycles and Transaction Semantics," in the *CORBA Components Guide*.

Note the stateless model requires an implementation that supports stateless execution. For example, if your component requires two subsequent invocations to compute a result for the client, it will break if you change the component properties to enable stateless behavior.

Passivation timeout for stateful components

If you use stateful CORBA components or EJB stateful session beans, configure a passivation timeout and removal timeout for the component. These settings limit the time that an instance can be bound to a client session. Doing so ensures that if a client crashes or an end user leaves their workstation, transactions do not remain open indefinitely if the component is transactional. The timeout also prevents component instances from tying up other server resources indefinitely.

To configure a timeout for stateful session beans, set the ejb.passivateTimeout and ejb.removeTimeout properties as described in Chapter 2, "Deploying and Configuring EJB Components," in the *EJB Users Guide*.

To configure a timeout for stateful CORBA or PowerBuilder components, set the Passivation Timeout as described in Chapter 4, "Managing CORBA Packages and Components," in the *CORBA Components Guide*.

Instance pooling

Instance pooling allows a single component instance to service multiple clients. Rather than creating a new instance for each client, EAServer maintains a pool of instances for reuse to service multiple clients. Instance pooling increases performance by eliminating the overhead of creating new instances for each client session. EAServer supports pooling of EJB stateless session bean and entity bean components by default. Chapter 2, "CORBA Component Life Cycles and Transaction Semantics," in the *CORBA Components Guide* describes how you can implement CORBA and PowerBuilder components that support pooling.

To enable pooling for CORBA and PowerBuilder components, set the Pooling option in the Management Console Component Properties pages. You can also programmatically enable pooling using the canBePooled transaction primitive method—see Chapter 2, "CORBA Component Life Cycles and Transaction Semantics," in the *CORBA Components Guide*.

To prevent idle pooled components from needlessly consuming memory, configure an instance pool timeout by setting the ejb.poolTimeout Ant property in the EJB module's user-configuration script—see "Commonly configured properties" in Chapter 2, "Deploying and Configuring EJB Components," in the *Enterprise JavaBean User's Guide*. For CORBA and PowerBuilder components, configure this setting for the EJB module that contains the EJB wrappers for your CORBA package.

Optimizing intercomponent calls

If your components make many intercomponent calls to EJB components, you can use local interfaces or call-by-reference. See “Optimizing in-server EJB calls” on page 42 for more information.

Using method results caching

For stateless components, you can configure method results caching. In this configuration, EAServer caches method return codes using a key composed of the input parameter values. When handling an incoming request, EAServer checks for a valid cached entry. If the cache contains a valid result, EAServer returns the cached return code rather than invoking the business method. To use this feature, your component must satisfy these requirements:

- The component must be an EJB stateless session bean, or a CORBA component wrapped by one. For CORBA and PowerBuilder components, the Stateful Session Bean component property must be disabled.
- The business method must return values (have a return type other than void) and have the same semantics whether it is called once for a given set of input values or multiple times. For example, you cannot enable results caching for a method that increments counters or creates new database rows every time it is called, regardless of input. Doing so would change the behavior of the application. On the other hand, if multiple calls with the same set of input values result in the same database end state, you can use method results caching.
- The business method must perform database lookups or other time-consuming processing. Methods that do simple calculations are unlikely to be worth caching, since the overhead of caching is likely equal to or greater than calling the method in the first place.
- The business method must operate on small input values. EAServer stores input values in memory as part of the cache key. Large input values will consume too much memory when the method results are cached.
- If the method results depend on data that can change independently of method execution, your application must include some mechanism to prevent the use of stale cache data. Use one of the following options:

- Configure and use a database *version table*. The version table contains a single row containing a an integer column that changes when data that affects the method outcome has changed. For example, if the method queries a database, you can configure database triggers on the tables that affect the method result to update the version table row.
- Configure a finite cache timeout. Cached values older than the timeout are discarded. This option can result in a larger performance gain since it removes the overhead of querying the database version table. However, it should not be used if stale data is never acceptable.

To configure method caching, run the <cacheResult> property task in the Ant user-configuration file for the EJB module that contains the stateless session bean to be configured. To configure a CORBA or PowerBuilder component, perform this configuration on the EJB module that contains the EJB wrapper components for the CORBA package. Here is an example configure-user target that configures a version table:

```
<target name="configure-user">
  <setProperties component="ejb.components.myjar.MyCompRemote">
    <cacheResult method="getProductInfo(java.lang.Integer)"
                 cacheSize="100"
                 dataSource="myDataSource"
                 tableVersion="ref_tv.version"
    />
  </setProperties>
</target>
```

Here is another example that configures a cache timeout:

```
<target name="configure-user">
  <setProperties component="ejb.components.myjar.MyCompRemote">
    <cacheResult method="getProductInfo(java.lang.Integer)"
                 cacheSize="100"
                 cacheTimeout="60"
    />
  </setProperties>
</target>
```

The table below describes the attribute values used in the examples:

Attribute	Description
`component`	Specifies the name of the DJC component that runs the application component that you want to monitor. Set this depending on the application component type, as follows: • For enterprise JavaBeans components, specify the DJC component that corresponds to the remote or local interface. You can read the DJC component names from the EJB module's Ant configuration file that was generated by deployment. • For CORBA and PowerBuilder components, specify the DJC component that corresponds to the remote interface of the EJB session bean that wraps your component. This name is `ejb.components.package.componentRemote`, where *package* is the CORBA package name, and *component* is the component name. • For Web components, specify the DJC component that corresponds to the servlet or JSP that the settings apply to. You can read the DJC component names from the Web application's Ant configuration file that was generated by deployment.
`method`	Specifies the Java name and signature of the business method. If omitted, the cache configuration applies to all non-void methods in the component.
`cacheSize`	The maximum number of results that will be held in cache. To avoid overflow, rows are removed from cache using a least-recently-used (LRU) discard strategy. If not specified, the default is 1000.
`cacheTimeout`	The maximum number of seconds that a result stored in cache will be considered valid. This property should not be used to attempt to control the cache size, because invalid results are only discarded from cache when an attempt is made to access them. A value of zero is interpreted as an infinite timeout. The default is zero.
`dataSource`	If a version table is required, specifies the name of the data source to connect to the database in which the table is located.
`tableVersion`	If a version table is required, specifies the table name and the name of the column that contains the version number. For example, `ref_tv.version` specifies the version column in table ref_tv.

Java/CORBA component performance

Java/CORBA and EJB components benefit from most of the settings described in "Common component performance issues" on page 33.

You can improve the performance of Java/CORBA and EJB components by configuring common class loaders for the application's components (including Web components). EAServer uses custom class loaders to allow you to refresh implementation classes without restarting the server. Loading multiple copies of the same class uses memory unnecessarily. You can eliminate redundant class loading by configuring an application-level or server-level class list, as described in Chapter 10, "Configuring Java Class Loaders," in the *System Administration Guide*.

EJB component performance

EJB components benefit from most of the settings described in "Common component performance issues" on page 33. You can also configure common class loaders as described in "Java/CORBA component performance" on page 41. You can configure the settings below to further tune EJB components.

The techniques described below can also be applied to PowerBuilder and CORBA components. Apply the settings to the EJB module containing the EJB wrappers for your CORBA package.

If you use EJB CMP entity beans, you can further tune the persistence settings described in Chapter 4, "EJB CMP Tuning."

Optimizing in-server EJB calls

Most J2EE applications have EJB components that are called from other EJB components and the Web tier (servlets and JSPs). You can use the following techniques to optimize these calls:

- Local interfaces, introduced in the EJB 2.0 specification, improve performance by eliminating the marshalling of parameter data; in other words, parameters in component calls are passed as local data references rather than copying the object.
- You can also enable Pass-by-reference semantics in EAServer to achieve the same benefits as local interfaces when code invokes components through the remote interface.

Local interfaces

Beginning in EJB version 2.0, clients can also execute EJB components using local interfaces if the client and component execute in the same virtual machine. Using the local interface can improve performance. You can use local interfaces for intercomponent calls, and for component invocations made from servlets and JSPs hosted by the same server as the component. For more information, see Chapter 3, "Developing EJB Clients," in the *Enterprise JavaBeans User's Guide*.

Pass-by-reference semantics

EAServer supports the proprietary EJB pass-by-reference mechanism supported by most other J2EE vendors. To enable pass-by-reference for EJB component, set the ejb.copyValues Ant property in the EJB module user-configuration file to false. For details, see "Commonly configured properties" in Chapter 2, "Deploying and Configuring EJB Components," in the *Enterprise JavaBeans User's Guide*.

When this setting is enabled, EJB stubs and skeletons for the component and its home and remote interfaces use the same parameter passing mode that EAServer normally uses for EJB 2.0 local interfaces. After changing the value, you must recompile the EJB module.

Stateful session beans

Stateful session beans are more resource intensive than stateless session beans. The stateful implementation remains bound to the client that creates them until the client calls the remove method or EAServer removes the instance because it has timed out. You can minimize the performance impact of using stateful session beans by following these recommendations:

- **Always call the remove method** Code your clients to call remove so that EAServer knows when the instance is no longer needed.
- **Configure a passivation timeout** Configure this setting as described in "Passivation timeout for stateful components" on page 38 The timeout ensures removal or passivation of component instances when the client crashes or the end user walks away from their desk.
- **Configure the maximum allowed instances** If clients activate too many instances at once, the server can run out of memory. To prevent this, set a limit on the number of instances that can be active using a thread monitor. See "Thread monitors" on page 35.

C++ component performance

Most C++ component performance issues are related to memory leaks. When using C++ types that are mapped from IDL, follow the memory management recommendations in “Using mapped IDL types” in Chapter 7, “CORBA/C++ Overview,” in the *CORBA Components Guide*.

The suggestions in “Common component performance issues” on page 33 apply to C++ components.

PowerBuilder component performance

To run PowerBuilder components in EAServer, use the most recent certified PowerBuilder versions. For details on which PowerBuilder versions Sybase recommends for EAServer on your platform, see the *EAServer Release Bulletin* for your platform. In addition, you can tune the following settings for improved performance.

Settings that affect system resource use

The system resource use of your PowerBuilder components determines how many instances can run on a given system, which in turn determines how many simultaneous clients the application can serve. Tune the settings in Table 3-2 to minimize resource use.

Table 3-2: PowerBuilder component settings that affect resource use

Setting	Description
Component class loader	By default, the PBVM uses per-component class loaders to run components. You can configure components to share class loaders to reduce the memory footprint required to run components. Doing so can improve scalability by allowing more component instances to run in the available memory. For details, see the Sybase white paper Reducing Memory Requirements When Using PowerBuilder Components in EAServer at http://www.sybase.com/detail?id=1019042.
DataStore resource footprint	DataStore objects used in components can consume system resources such as memory and Windows user and kernel object handles. When many DataStore objects are instantiated, they can exhaust the available resources unless you have tuned the DataStore settings to minimize resource use. For details on tuning DataStore settings, see the Sybase white paper Operating System Constraints Affecting the Scalability of PowerBuilder DataStores in EAServer at http://www.sybase.com/detail?id=1019174.

Setting	Description
DataWindow memory management	For large retrievals or imports into a DataWindow object, set the datawindow.storagepagesize property to LARGE. Setting this property allows the DataWindow to most efficiently use the available virtual memory. While the setting LARGE is recommended, a setting of MEDIUM is also available. For more information, see the *DataWindow Reference* manual in the PowerBuilder documentation.
Bind thread	Disable Bind Thread for PowerBuilder components deployed to EAServer unless instructed otherwise by the documentation that accompanies your PowerBuilder version or Sybase technical support. For more information on threading issues that affect PowerBuilder components, see the *Application Techniques* manual in the PowerBuilder documentation.
Garbage collection	The PBVM uses a garbage collection model to free memory used by unreferenced and orphaned objects. For more information on how garbage collection happens in PowerBuilder, see the *Application Techniques* manual in the PowerBuilder documentation.

DataStore row height size

If you are retrieving a lot of rows of data, try setting the datawindow.detail.height.autosize property for the DataStore object to false. Depending on the number of rows being retrieved, this setting can have a significant impact on performance. If you have autosize enabled for the height or width of any specific objects, try disabling those settings as well. For more information, see the *DataWindow Reference* manual in the PowerBuilder documentation.

Web DataWindow settings

If you use Web DataWindows, tuning these settings can improve performance.

Tuning code generation settings You can tune the Web DataWindow settings to minimize the size of the generated JavaScript code. Doing so improves the client response time by avoiding the generation and network transport of unneeded code. If you do not use a feature such as display formatting, validation rules, or client-side scripting, disable code generation for the unused feature. You can also cache client-side methods in JavaScript files to reduce the size of the generated code and increase performance on both the server and the client. Without JavaScript caching, each time a Web DataWindow is rendered in a client browser, JavaScript code for DataWindow methods is generated on the server and downloaded to the client. However, there is no performance gain if the client Web browser settings prevent caching. The *DataWindow Programmer's Guide* in the PowerBuilder documentation describes techniques for controlling the size of generated code.

Using custom containers For improved performance, use a custom component as the Web DataWindow container rather than the predefined HTMLGenerator component. When using a custom component, you can configure additional settings to reduce the number of method calls required to configure the component. Doing so can result in improved performance, maintainability, and scalability. Specifically, you can set the source file and DataWindow object on the server so that the DataWindow object is loaded when the component instance is created, resulting in fewer method calls from server-side scripts in the Web page. You can also improve performance by having your custom component maintain its state. For more information, see the *DataWindow Programmer's Guide* in the PowerBuilder documentation.

Changing the Web target default behavior By default, the PSJaguarConnection methods for using a Web DataWindow make several trips to the server. You can set the *bOneTrip* argument to make one trip to the server instead. Doing so improves performance by reducing network traffic. For more information, see the PSJaguarConnection reference pages in the *Web and JSP Target Reference* manual in the PowerBuilder documentation.

CHAPTER 4 EJB CMP Tuning

For EJB CMP entity beans, EAServer implements the persistence engine that manages the mapping between the entity bean's container-managed fields and the underlying database. This chapter describes how to tune the persistence settings for best performance.

CMP tuning concepts and terminology

To understand the CMP tuning options and strategy, it is helpful to understand the concepts and terminology used.

Concurrency control

Concurrency control prevents overlapping updates from entity instances running in different threads or different servers, or from applications running outside of EAServer. There are two approaches for concurrency control:

- In the *Pessimistic concurrency control (PCC)* model, data rows are locked when read, for the duration of the EAServer transaction. This method can introduce database deadlocks and usually reduces the scalability of the application.

- In the *Optimistic concurrency control (OCC)* model, data rows are not locked when read. Timestamps are used for concurrency control; the timestamp can be a timestamp column in the database that is updated every time the row is modified, or it can be the row data itself. At the end of the transaction, the in-memory timestamp value is compared to the timestamp value in the database, and the transaction rolls back if the values do not match.

Object and query caching

Many optimizations rely on in-memory caching of database query result sets. EAServer has two cache mechanisms for caching container managed field data and query results:

- The *object cache* stores results from findByPrimaryKey method invocations and the associated entity bean instances.
- The *query cache* stores results returned from other finder methods and query methods.

EAServer creates and manages these caches internally. There are no configuration entity types to configure them explicitly; rather, they are configured implicitly by your CMP table and field mapping configuration settings.

Caching can improve performance by minimizing the number of database select queries required for ejbLoad operations, finder method invocations, and ejbSelect method invocations. Most database applications are governed by the 80:20 rule: 80% of users access 20% of the data. Object caching increases performance and scalability by allowing faster access to the most recently used data.

Assuming that the database access is the principal bottleneck, the expected performance gain falls in these ranges, depending on the ratio of update to read-only transactions:

- 1.5 to 2 times faster for applications where most transactions are updates.
- 3 to 30 times faster for applications where most transactions are read-only.

Besides the transaction mix, the actual performance gain depends on:

- The size of the database table
- The size of the object and query caches
- The cache time out value

In summary, the best use case for caching is data that is static. If the data changes often, the overhead of updating caches can outweigh the performance benefits of caching. If the data is updated too frequently, soft locking or hard locking may yield better performance. Furthermore, the data consistency requirements dictate how cached data can be used. Decide how much consistency you require, then optimize within those constraints.

Just-in-time JDBC wrapper drivers

EAServer includes customized JDBC drivers for use by CMP entity beans. Many performance optimizations require code run in a wrapper driver. For example, just-in-time (JIT) creation of semi-temporary stored procedures to run queries and updates. The wrapper drivers offer better performance by allowing updates to be deferred to the end of each transaction and sent together as a command batch. Doing so improves performance by reducing network round trips between the database server and EAServer.

Note The wrapper does not replace the underlying JDBC driver - it merely permits CMP tuning at the level of JDBC prepared statements. All calls to the database go through the underlying JDBC driver.

Logical isolation level

In standard SQL using PCC, the *transaction isolation level* defines the degree to which data can be accessed by other users during the transaction. For example, the isolation level “repeatable reads” locks all rows or pages read during the transaction. After one query in the transaction has read rows, no other transaction can update or delete the rows until the repeatable-reads transaction completes.

When using OCC, EAServer supports a *logical isolation level* that provides the same semantics as the corresponding PCC isolation level, while allowing safe use of OCC optimizations such as avoiding row locks and query results caching.

Logical isolation levels supported by EAServer are listed in the description of the ejb.isolationLevel Ant property—see “Commonly configured properties” in Chapter 2, “Deploying and Configuring EJB Components,” in the *Enterprise JavaBean User’s Guide*.

Ant configuration for CMP entity beans

Before reading this chapter, you should be familiar with the deployment settings described in Chapter 2, “Deploying and Configuring EJB Components,” in the *Enterprise JavaBean User’s Guide*. In particular, review the descriptions of these Ant properties listed under the section “Commonly configured properties”:

- sql.dataSource, which specifies the data source (and thus the target database)
- sql.createTables, which specifies whether to create tables automatically
- ejb.isolationLevel, which specifies a logical isolation level for queries and transactions run by the entity beans
- sql.isolationLevel, which specifies the database isolation level setting to request from the JDBC driver
- ejb.transactionBatch, which allows you to specify a default transaction batch configuration for the entity beans
- ejb.transactionRetry, which enables automatic transaction retry globally for the entity bean methods when using optimistic concurrency control.

When you deploy EJB CMP entity beans, EAServer generates Ant configuration commands to define default database mappings and query method configurations for the entity beans. To modify these settings, you can copy the generated commands to your user-configuration file and modify them.

Table and field mapping configuration

The <persistentObject> and <persistentField> Ant task configure the object-relational mapping for the entity beans. For each entity bean, <persistentObject> commands specify the data source to use, which table is queried, and other settings such as caching options and transaction isolation level. For each container-managed field, <persistentField> maps fields to database table columns.

For example:

```
<target name="configure-user">
  <setProperties component="ejb.components.example.CustomerInventory">
    <persistentObject
        table="cust_inv"
        isolationLevel="RepeatableRead"
        dataSource=”myDB”
    />
    <persistentField field="id"
        column="c_id"
    />
    <persistentField field="name"
        column="c_name"
        maxLength="40"
    />
  </setProperties>
</target>
```

Many optimizations require you to modify or add parameters in the <persistentObject> invocations.

Finder and query method configuration

The <queryMethod> Ant task configures the EJB-QL queries for finder and query methods and optionally configures other settings such as an isolation level that applies to this query only. For example:

```
 <target name="configure-user">
    <setProperties component="ejb.components.example.Product">
      ...
      <queryMethod
        method="findByName(java.lang.String name)"
        isolationLevel="RepeatableReadWithCache"
        tableVersion="ref_tv.tv"
        />
      <queryMethod
        method="findByTitle(java.lang.String title)"
        isolationLevel="RepeatableReadWithCache"
        tableVersion="ref_tv.tv"
        />
    </setProperties>
  </target>
```

Many optimizations require you to modify or add parameters in the <queryMethod> invocations.

Finding persistence bottlenecks

You can generate a transaction cross reference log for your application as described in “Transaction cross-reference logging” on page 94. Review the cross-reference log files to identify persistence bottlenecks to which the various batching and caching options may be applicable.

Creating and tuning database tables

While EAServer automatically creates entity bean tables for supported databases, this feature is provided as a development time convenience. For deployment to production servers, you or your DBA should create the tables, using an optimized index model and any other necessary optimizations, such as enabling row-level locking. You can also add tuning parameters to the SQL and DML syntax that is configured in the table mapping properties for the entity bean. For example, you might optimize the select query to force the use of an index by adding proprietary DBMS keywords.

For more information, see the description of the sql.createTables Ant property in Chapter 2, “Deploying and Configuring EJB Components,” in the *Enterprise JavaBean User’s Guide*.

Configuring the logical isolation level

Logical isolation levels supported by EAServer are listed in the description of the ejb.isolationLevel Ant property—see “Commonly configured properties” in Chapter 2, “Deploying and Configuring EJB Components,” in the *Enterprise JavaBean User’s Guide*. Set this Ant property in your user-configuration script to specify an option used globally (in all <persistentObject> invocations that reference `${ejb.isolationLeve}` as the isolationLevel value). You can also configure the value explicitly for individual components.

Tuning data source settings for CMP entity beans

The data source used by EJB CMP entity beans is specified by the sql.dataSource Ant property—see “Commonly configured properties” in Chapter 2, “Deploying and Configuring EJB Components,” in the *Enterprise JavaBean User’s Guide*.

Tune the data source pool size and database type

Tune the cache size parameters as described in “Data source settings” on page 89. For EJB CMP entity beans, make sure you specify a Database Type that matches your database server. The database type definition allows the persistence engine to make use of database-specific features such as stored procedures and statement batches. You can create additional database type definitions as described in “Configuring database types” in Chapter 4, “Database Access,” in the *System Administration Guide*.

Use JIT JDBC wrapper drivers

EAServer includes customized JDBC drivers for use by CMP entity beans to implement the performance optimizations described in “Just-in-time JDBC wrapper drivers” on page 49. Wrapper drivers are provided for Sybase and Oracle database drivers.

- **Sybase** This driver is a wrapper around the Sybase jConnect driver. To use the driver, specify the following class names in the Data Source properties:

  - For JDBC Data Source Class, specify com.sybase.djc.sql.jit.SybaseDataSource3
  - For JDBC/XA Data Source Class, specify com.sybase.jdbc3.jdbc.SybXADataSource.

  Other data source properties for this wrapper driver are the same as are used by com.sybase.jdbc2.jdbc.SybDriver, plus those listed in Table 4-1.

- **Oracle** This driver is a wrapper around the Oracle JDBC driver. To use the driver, specify the class name com.sybase.djc.sql.jit.OracleDriver as the driver in your data source. Data source properties for this wrapper driver are the same as are used by oracle.jdbc.driver.OracleDriver, plus those listed in Table 4-1.

Oracle driver classes not included
The Oracle JDBC driver classes are not included in the EAServer installation. Before using the Oracle wrapper driver, make sure these classes are deployed to EAServer and that direct Oracle data sources can be pinged successfully.

Table 4-1 lists the additional properties supported by the wrapper drivers. You can configure these properties on the Advanced tab in the Management Console Data Source Property pages, or by running <setProperties> in an Ant configuration script and setting them with nested <configProperty> commands.

Table 4-1: Sybase JIT JDBC wrapper driver properties

Property	Legal Values	Default Value	Description
jit:printWarnings	true/false	true	Enables all database warning messages received by wrapper driver to be printed in server log.
jit:maximumBatchParameters	0 or positive	99 (subject to change)	Maximum number of parameters in a batch.
jit:maximumBatchStatements	0 or positive	8 (subject to change)	Maximum number of statements in a batch. Any value less than 2 effectively disables batching. Larger values will give better performance as long as memory is available. Setting this too high may result in too many stored procedures being created, and the database server may run out of procedure cache.

Automatic key generation settings

EAServer supports several mechanisms for automatic key generation. If automatic key generation is enabled, keys are created automatically for every row inserted in the table. In

Java key type for beans that use automatic key generation

There are two options for the primary key type when using automatic key generation in EJB-CMP entity beans:

- **java.lang.Object** The EJB 2.0 specification requires this type for entity beans that have automatically generated keys. However, using java.lang.Object makes client coding difficult, particularly if the home interface has finder methods that take key values as input. In this case, you do not know what the actual Java key type is until after deploying the component.
- **java.lang.Integer or other integer types** EAServer allows you to use an integer type with automatic key generation configured. You can also use other integer types, as long as you specify the wrapper class name, such as java.lang.Long or java.math.BigDecimal with scale of zero.

Configuring automatic key generation

EAServer supports three mechanisms for key generation:

- **Using the Sybase identity datatype** If you are using Sybase Adaptive Server Enterprise or Adaptive Server Anywhere, the mapped table uses the identity datatype for the primary key. The database manages the creation of new keys.
- **Using the Oracle sequence datatype** If you are using an Oracle database, the main table uses an Oracle sequence for the primary key. The database manages the creation of new keys.
- **Using a key lookup table** For any SQL database, you can use a single-row, single-integer-column table to generate key values. EAServer increments the key lookup value to generate new keys. If other processes or applications insert to the table, they must also use the key lookup table.

To configure the key generation mechanism and enable automatic key creation, set the keyGenerator attribute in the <persistentObject> Ant command that maps the database table to the entity bean. For example:

```
<target name="configure-user">
  <setProperties
component="ejb.components.example.CustomerInventory">
    <persistentObject
        table="cust_inv"
        isolationLevel="RepeatableRead"
        dataSource=”myDB”
        keyGenerator=”key-gen-value”
    />
  </setProperties>
</target>
```

The the keyGenerator attribute takes the values listed in Table 4-2.

Table 4-2: keyGenerator attribute values

Value	To specify
`sequence` *`seq-name`*	Oracle sequence *seq-name*
`select @@identity`	Sybase style identity column
`key-table.key-column` `+=` *`batch-size`*	Key generator table using the specified table, column name, and key increment batc size. For example, to specify a table named cust_key, using column next_id, and incrementing through 100 keys at a time, set the value to: `cust_key.next_id += 100` Create the key lookup table if it does not exist in the database. EAServer does not crea e it automatically.

Tuning settings related to automatic key generation

If your component uses the Adaptive Server Enterprise with the Sybase identity column type, make sure the relevant database and table options are tuned, such as the identity burning set factor database option or the identity_gap table creation parameter.

If your component uses automatic key generation with a key-lookup table, tune the key batch size value. To prevent different threads from creating duplicate keys, EAServer uses a semaphore to synchronize the key increment operation. Each thread reserves the specified number of key values per increment. The key use rate can be tuned to reduce inter-thread contention for locks on the key table. A value of 100 results in good performance for most applications. Large batch increment values can result in large gaps between key values. Gaps in the key sequence are possible if the key use rate is greater than 1.

Configuring concurrency control options

You can configure OCC or PCC for a bean as a whole or for transactions started by individual finder or business methods. By default, EAServer uses OCC with an all-values comparison for timestamp checking.

OCC allows greater scalability than PCC for most CMP entity beans. However, when using OCC, you must code your application to retry rejected updates, or you must enable automatic transaction retry for the application components.

PCC can perform better than OCC when your beans are mapped to tables with very high update contention. In these cases, the overhead of retrying transactions that fail due to update collisions can outweigh that caused by using database locks. If you have configured OCC, and see many "TRANSACTION_ROLLEDBACK: Optimistic Concurrency Control" messages in the server log, you should try PCC on the component identified in these messages.

Enabling PCC

To configure pessimistic concurrency control, you can do one of the following:

- Set the <persistentObject> selectWithUpdateLock or selectWithSharedLock attribute to true.

  The selectWithUpdateLock setting requests an exclusive database lock be obtained at select time to avoid deadlocks during lock promotion. If you use this setting, also consider configuring the database table for row-level locking.

  For databases such as Sybase Adaptive Server Enterprise that do not support select for update locking syntax, EAServer locks rows by issuing a no-change update statement. The <persistentObject> touchColumn attribute specifies which column to update. If you do not set this property, EAServer uses the first non-key column. For best performance, specify the column with the datatype that can be updated most quickly. For example, int columns can be updated more quickly than varchar columns. If no existing column is suitable, consider adding an int column that defaults to 1.

- Configure the EJB finder and select methods queries and add "holdlock" or the appropriate lock syntax for your database. For more information, see "Finder and query method configuration" on page 51.

Enabling OCC

When using OCC, each update statement contains SQL logic that determines if the last-read timestamp matches the stored value, and rolls back the transaction if the timestamp does not match. In other words, updates based on stale data are rejected. There are several options for using timestamps:

- Use a timestamp column: each table contains a timestamp column, which can be a database timestamp type (if supported) or an integer column that is incremented for every update. This option provides good performance if your database and table schema can support it.
- Use all-values comparison: on update, all row values are compared to the last-read values to detect update collisions. OCC with all-values comparison is the default concurrency control model. Performance with this option is worse than when using a single timestamp column, particularly if the table contains many columns or wide columns (such as Sybase text or image columns). Whenever possible, the use of a timestamp column is recommended in these cases.
- Use a table-level timestamp: the timestamp is a single integer counter that is incremented for every update, insert, or delete in the main table. This option provides the best performance for CMP entity beans that are mapped to read-mostly (or read-only) tables when verified results are required to meet transaction isolation requirements. For best results, use table-level timestamps with a Sybase JIT wrapper driver to allow verification queries to be batched with other deferred operations. See "Use JIT JDBC wrapper drivers" on page 53 for more information.

Configuring OCC options

To enable OCC, first verify that PCC is disabled, then configure the timestamp mechanism of your choice.

To ensure that PCC is disabled, verify that the <persistentObject> configuration does not set the selectWithSharedLock or selectWithUpdateLock attributes. Both attributes default to false.

To to configure an OCC timestamp mechanism, set one of the <persistentObject> or <queryMethod> configuration attributes described in Table 4-3.

Table 4-3: OCC timestamp configuration options

<persistenObject> or <queryMethod> attribute	Description
None	If you specify no timestamp or lock-related attributes in the <persistentObject> configuration, the default behavior is OCC with all-values comparison for version control.

<persistenObject> or <queryMethod> attribute	**Description**
timestampColumn Applies only to <persistenObject>	Set the timestampColumn attribute in the <persistentObject> configuration to specify the name of a column that holds a row timestamp which is automatically set (or changed) by the database server whenever a row is inserted (or updated). This column is usually the database timestamp type – you can use the timestamp datatype if using Sybase Adaptive Server Enterprise or Adaptive Server Anywhere version 7.0 or later.
versionColumn Applies only to <persistenObject>	Set the versionColumn attribute in the <persistentObject> configuration to specify the name of an integer-typed column that holds a row version which is automatically set (or changed) by the persistence manager whenever a row is inserted (or updated). If, due to triggers or other database-specific mechanisms, this version column is also updated when ad-hoc updates are made to the database table by a client which is not performing version management, then the trigger or other mechanism must be disabled when the persistence manager is in use. This disabling is needed to prevent an update issued by the persistence manager resulting in a double increment of the version column (one time by the persistence manager and one time by the database server). To disable triggers, set the Disable Triggers property for the data source.
tableVersion	Set the tableVersion attribute in the <persistentObject> or <queryMethod> configuration to specify the use of a timestamp table. Specify a table and column name, in the form `ts_table.ts_column`, where *ts_table* specifies the timestamp table and *ts_column* specifies the name of the timestamp column in the timestamp table. The specified timestamp table must be separate from the main table. The timestamp tables can contain multiple columns, to allow use of one timestamp table by multiple entity beans. Timestamp tables are automatically created if they do not exist. A timestamp table can be shared among multiple components even when only one column is present in the timestamp table. In other words, a single timestamp value can be shared by multiple tables. This helps further improve performance for a group of read-mostly tables. However, any insert, delete, or update on any of the tables results in all cache entries being discarded. If, due to triggers or other database-specific mechanisms, this “table version” column is also updated when ad-hoc updates are made to the database table by a client which is not performing version management, then the trigger or other mechanism must be disabled when the persistence manager is in use. This disabling is needed to prevent an update issued by the persistence manager resulting in a double increment of the table version column (one time by the persistence manager and one time by the database server). To disable triggers, set the Disable Triggers property for the data source. You cannot use a timestamp table if external processes outside of EAServer can update the versioned data without updating the timestamp table.

<persistenObject> or <queryMethod> attribute	**Description**
parentVersion	Set the parentVersion attribute in the <persistentObject> or <queryMethod> configuration to specify version control using a particular version of a "parent" object to allows cached entities to be associated with a particular version of a "parent" object, while guaranteeing a high level of transactional consistency. The syntax of this property is: *ParentEntity*[*ForeignKey*].*version* The parent entity refers to another entity which uses the versionColumn attribute as the timestamp mechanism. The foreign key is a persistent field of the current entity which is assumed to be the primary key for the corresponding parent object. This is particularly useful in combination with a logical isolation level of RepeatableReadWithCache or SerializableWithCache. As an example, a foreign key field custId declared on an Order entity could be associated with a parent Customer entity by using the parent version expression: `Customer[custId].version`

Enabling automatic transaction retry

EAServer can automatically retry transactions that are rolled back—method calls back to the last transaction boundary are retried by the stub code. This feature is useful for EJB CMP entity beans that use optimistic concurrency control.

Auto-retry is not appropriate for all applications. For example, an end user may want to cancel a purchase if the item price has risen. If auto-retry is disabled, clients must be coded to retry or abort transactions that fail because of stale data. The exception thrown is CORBA::TRANSIENT (for EJB clients, this exception is the root cause of the java.rmi.RemoteException thrown by the EJB stub).

Auto-retry must be enabled for the component that initiates the transaction, which is typically a session bean in EJB applications.

To configure automatic retry globally for beans in an EJB module, set the ejb.transactionRetry Ant property to true in the user configuration script. To configure this setting for individual components, set the <transaction> command in a <setProperties> task that configures the component's remote or local interface, or a specific method in that interface (whichever is used in the call sequence that initiates the container-managed transaction). For details, see the description of the ejb.transactionRetry Ant property in "Commonly configured properties" in Chapter 2, "Deploying and Configuring EJB Components," of the *Enterprise JavaBean User's Guide*.

Using soft locking

You can configure in-server soft locking of database rows used by EJB CMP entity beans that use the isolation level `repeatable_read_with_cache`.

If you enable soft locking for a component, EAServer applies a soft lock to each row selected by an instance, which prevents other component instances running in the server from updating the row. Soft locking must be used with optimistic concurrency control (OCC). The soft lock prevents update collisions between instances in the same server, while OCC prevents update collisions with external applications and instances running in another server.

As an alternative to pessimistic locking, OCC with soft locking may improve performance if there is heavy update contention among entity bean instances running in a single-server deployment or in small-to-medium sized clusters. In clusters, if you see excessive OCC update failures, you may need to switch to pessimistic database locking as described in "Enabling PCC" on page 57.

You can enable soft locking by setting the following <persistentObject> attributes:

- cacheLock – a value of true enables soft locking. The default of false disables soft locking.
- dcacheLockTimeout – the timeout period for soft-locked rows, specified in seconds. Soft locks use a timeout mechanism to avoid deadlock. The default is 5. If too many "soft lock timeout" errors are reported in the server log, increase the timeout.

For example:

```
<target name="configure-user">
  <setProperties component="ejb.components.example.CustomerInventory">
    <persistentObject
        table="cust_inv"
        isolationLevel="RepeatableRead"
        dataSource="myDB"
        cacheLock="true"
        cacheLockTimeout="10"
    />
    ...
</target>
```

Configuring object and query caching

For components, object caching is enabled if you have configured an isolation level that requires caching. You can further customize the caching parameters as described in "Enabling object caching" on page 63. Query caching must be configured for each finder and ejbSelect method—see "Enabling query caching" on page 64. Query caching is disabled by default.

Cache coherency and transaction consistency

When data is maintained in the object cache as well as the source database, you must take steps to ensure these transactional constraints are satisfied:

- *Read consistency*, to ensure that data read from the cache matches data in the source database.
- *Update consistency*, to ensure that updates are not committed if the source data has changed since it was last read.

Read consistency If your application requires read consistency, choose an isolation level that requires it, such as `ReadCommittedVerifyUpdates` or `ReadCommittedWithCacheVerifyUpdates`. When read consistency is required, caching should be used only when the data changes infrequently. Caching volatile data can make your application perform worse because the added overhead of retrying queries that roll back because the data changed.

Update consistency When using caching, transactional update consistency is ensured by:

- **The timing of cache updates** Cache entries are never modified or deleted until the transaction associated with the change has committed.
- **Optimistic Concurrency Control (OCC)** At commit time, EAServer runs a verification query to check whether the data has changed since it was originally selected. If data has been changed by another user, EAServer rolls back the transaction.

  You should not disable OCC when using object caching, and you should use a timestamp or version column rather than using the default value-comparison technique of concurrency control. Version columns are preferable to database timestamp columns when using caching. When using the database timestamp column type, EAServer must requery after updates to get the new timestamp value, which partly defeats the advantages of using a cache (particularly if there are frequent updates and you have enabled the cacheLock setting).

- **Read consistency using timeouts** For applications that have a more lax requirement for read consistency, you can configure cache timeouts to minimize the use of stale data. The cache timeout sets a time limit on how long cached data remains valid. Stale entries are refreshed from the source database before the data is used in the component.

Enabling object caching

The object cache is enabled implicitly if you configure an isolation level that requires the use of the cache—see "Configuring the logical isolation level" on page 52. To configure the cache settings explicitly, set the <persistentObject> attributes listed in Table 4-4.

Table 4-4: <persistentObject> attributes to configure object cache settings

Attribute	Description
cacheSize	The maximum number of database rows that will be held in cache. To avoid overflow, rows are removed from cache using a least-recently-used (LRU) discard strategy. If not set, the default is 1000.
cacheTimeout	The maximum number of seconds that a row stored in cache will be considered valid. This setting should not be used to attempt to control the cache size, because invalid rows are only discarded from cache when an attempt is made to access them. The default value of zero is interpreted as an infinite timeout.
cacheLock	When the name of the selected isolationLevel ends with "Cache," this setting determines whether exclusive locks should be used to ensure that only one transaction at a time can access any cache entry. The purpose of this setting is to permit the use of caching with entities that have some level of update contention. This property should not be relied upon to ensure exclusive access to the underlying database table rows. It is best used in combination with optimistic concurrency control.
cacheLockTimeout	The number of seconds that a transaction will wait when trying to obtain a lock. If a lock is not acquired in this period, the transaction is rolled back.
cacheChildren	When query methods refer to this entity using the parentVersion attribute, the query methods must also be explicitly referenced by this entity using the cacheChildren attribute. Specify a comma-separated list of `ChildEntity.QueryName` entries such as: `Order.findByCustomer,Order.findByItem`

Enabling query caching

Query caching allows EAServer to cache the values returned by finder and ejbSelect method queries. When caching is enabled for a query, the key values returned by each invocation are cached in memory, with the method input parameter values serving as the cache key. Together with entity object caching, query caching can reduce the number of unnecessary database reads.

To enable caching for a finder or ejbSelect query, configure the <queryMethod> command attributes listed in Table 4-5.

Table 4-5: <queryMethod> attributes to configure query cache settings

Attribute	Description
isolationLevel	To enable query caching for a method, you must explicitly specify an isolation level in the <queryMethod> configuration that includes "Cache" in the name. See "Logical isolation level" on page 49.
cacheSize	The maximum number of database rows that will be held in cache. To avoid overflow, rows are removed from cache using a least-recently-used (LRU) discard strategy. If not set, the default is 1000.
cacheTimeout	The maximum number of seconds that a row stored in cache will be considered valid. This setting should not be used to attempt to control the cache size, because invalid rows are only discarded from cache when an attempt is made to access them. The default value of zero is interpreted as an infinite timeout.
preloadCache	If the logical isolation level for the associated persistent object indicates the use of a cache, and if this query has no parameters, then this property (if true) requests preloading of results into the cache.

CHAPTER 5 Web Application Tuning

This chapter describes the settings that you can tune to optimize the performance of your Web applications.

Tuning server and Web application settings

These Web application and server settings can affect the performance of your Web-based application.

In most cases, you modify your Web application using the development tool of your choice, then deploy (or redeploy) the Web application in to EAServer. Refer to the *Web Application Programmers Guide* for additional information.

Tracing properties

Tracing properties enable additional logging, which can be useful when debugging problems. However, tracing requires additional file I/O and computation. For best performance, disable these trace properties unless you are troubleshooting a related issue:

- **Server tracing** There are several options for tracing server events (RMI-IIOP, JMS, SQL), which can be set from the Log/Trace tab of the Server Properties dialog box. See the *EAServer System Administration Guide* for more information.

- **Web application tracing** The Web application property <property name="web.enableTracing" value= "true"/> enables tracing of the Web application. You must also set the Web Trace property from the Log/Trace tab of the Server Properties dialog box to enable Web application tracing.
- **HTTP request log** The server property enableHttpRequestLog generates an HTTP request log. You can set this property from the Log/Trace tab of the Server Properties dialog box. See the *EAServer System Administration Guide* for more information.

Session timeouts

Servlets and JSPs can use sessions to store temporary data required to maintain a Web user's session. EAServer also uses sessions internally in the Web application security implementation. The Web application Session Timeout property specifies how long a session can remain inactive, with no requests issued from the client. Since sessions consume memory resources, you should tune this setting to balance memory requirements against the possibility of users losing their session.

To configure this property, set the Session Timeout property for the Web application by setting the <session-config> property in the *web.xml* file using your Web application development tool.

Class loader settings

EAServer uses custom class loaders to allow you to refresh implementation classes without restarting the server. Loading multiple copies of the same class uses memory unnecessarily. To avoid this issue, configure a common class loader for use by the Web application and the components that it calls. To do this, configure an application-level or server-level class list, as described in Chapter 10, "Configuring Java Class Loaders," in the *EAServer System Administration Guide*.

Clustered deployments

If you deploy your Web application in a cluster, tune the settings described in "Web application settings" on page 97.

HTTP and HTTPS listener configuration

The HTTP listener parameters can affect the performance of your application. "Listener tuning" on page 26 describes how to tune these settings.

SSL and performance

You can configure Web pages to require SSL as described in Chapter 3, "Using Web Application Security," in the *EAServer Security Administration and Programming Guide*. SSL encryption can protect critical client data, such as passwords and credit card numbers. However, SSL adds overhead to the network transfer phase. Use SSL only when the extra security is required.

Tuning servlet and JSP settings and code

Use these tips to tune the implementation of servlets and JSPs and their deployment properties in EAServer.

Use local interfaces for EJB calls

If the Web application calls EJB components, local interface invocations offer the best performance since they pass parameters on the stack rather than marshalling parameter values into an IIOP stream. For information on using local interfaces, see these sections in the *EAServer Enterprise JavaBeans User's Guide*, from Chapter 3, "Developing EJB Clients":

- "Instantiating remote or local interface proxies"
- "Calling local interface methods"

Threading

Avoid using servlets that must be single-threaded. One instance of a single-threaded servlet can serve only one client at a time, while thread-safe servlets can serve all clients with one instance.

If you cannot avoid using a single-threaded servlet, configure the number of instances to minimize blocked client requests (requests block if there are more requests than available instances). For more information, see “Threading settings” in the “Creating Java Servlets,” chapter in the *EAServer Programmer’s Guide*.

Preloading classes

If your servlets that take a long time to load and initialize, configure them to load when the server starts. Otherwise, the first client that calls the servlet experiences poor response time when the servlet is loaded to satisfy the request. You can also configure JSPs to load at start-up. If a JSP is loaded at start-up, it is compiled if necessary.

Define the `<load-on-startup>` property for the servlet or JSP in the Web application’s *web.xml* file.

JSP compilation options

JSP runtime compilation is expensive. While this feature is convenient during the development phase, you should precompile JSPs when deploying them to production server. If you precompile JSPs, you can further improve performance by disabling runtime timestamp checking. If your application design requires runtime compilation, you can tune the JSP compilation settings to reduce compile time.

Precompiling JSPs

There are two ways to precompile JSPs:

- Configure the JSP to load at start-up, as described in “Preloading classes” on page 68. The JSPs are compiled when the server starts up. This technique requires that you have Web components defined for each JSP in your application.
- Compile the JSPs using the `jagtool compilejsp` command. You can do this from deployment scripts or batch files, or in an Ant build file. For more information, see Chapter 6, “Using jagtool and jagant,” in the *Automated Configuration Guide*.

Tuning distributed HTTP session settings

Web applications manage state information via HTTP sessions. Distributed HTTP sessions are required to support various features, such as:

- Clusters – HTTP session information is distributed to support failover of HTTP sessions.
- Load balancing – the same HTTP session information must be available on multiple servers simultaneously.

Types of distributed HTTP sessions include:

- In-memory persistence – Provides a faster way of replicating and maintaining HTTP session state information in a single server instance.
- Database persistence – Provides a reliable way to save session state information in persistent storage. While In-memory persistence is faster, some applications require a more permanent storage method or the ability to share session data among servers running in a cluster.
- A combination of in-memory and database persistence – This is the method used in EAServer. It combines the speed of in-memory persistence with the reliability of database persistence. For this reason, it is used in EAServer. each node maintains a local in-memory copy of the data. When a client requests an HTTP session, it sends a version number indicating which version it requires. If the version number in the local in-memory copy is the same, then the data from memory is used, otherwise it is obtained from the database. After each request is processed, the version number increments (only if the data has changed) and the database is updated. This method reduces the number of database accesses in half.

Distributed HTTP sessions meet these requirements:

- Optimized – clients that do not frequently update the session simply read the HTTP session information.
- Portable – not tied to any one Web server implementation.
- Reliable – database persistence provides reliable storage for session information.
- Versioning – changes only when the session state changes.

How it works

Each request made to a Web server is inspected to determine from where the session information is retrieved (in-memory or database). After processing, session information is updated and saved to a database. Web application filters and the SessionManager interface are used to intercept and process requests before the response is sent back to the client. The filters determine if:

- A new session is created – if there was none before invoking the servlet/JSP but there is one afterwards.
- The existing session is modified – If there was a session before the servlet/JSP and there is one afterwards, then a comparison is made to see if any attributes change. If so, the database and version cookie are updated.
- The session is removed – when there was a session before the servlet/JSP but none afterwards the session is deleted from the database.

Sessions

Session versioning

The version is tracked using a cookie. The cookie is sent to the client for each response. For each subsequent request, the client sends the cookie back indicating what version it requires. If the browser does not send back a cookie, the Web application operates correctly, but performance is reduced.

Session Manager

For authentication, the Web server creates a new session if necessary before the filter is invoked. See "Filter deployment" on page 71.

Session versioning allows use of a local copy of the session information instead of delegating it to a database. Without an associated version, session information would have to be read from the database to guarantee the data is not stale. Each session has an associated version. The version starts at one and changes each time a change is made to the session. To detect whether a session has changed, a cloned copy of the existing session is saved with which session information is compared. After processing, the request is compared to the cloned session to see if they are different.

Custom HTTP sessions

A custom version of an HttpSession object that contains an additional version parameter is created and used to maintain version information.

Distributable attribute

The distributable attribute is set automatically at the package level for any Web application marked as distributable. This attribute inserts a method call after the super.service() method, which updates the version cookie before the container sends the response to the client. This attribute can be set in the *web.xml* descriptor file of your Web application

Filter deployment

One filter is installed for each Web application, which has a path filter mapping of “/*”. This filter is automatically installed into your Web application during deployment if the Web application is marked as distributable.

Local in-memory cache

In order to serve the HTTP sessions more efficiently, the in-memory version of the session data managed by the Web server is used if appropriate. In order to determine whether or not the in-memory version is appropriate, the Web server compares the version that the client requires against the stored version.

Database persistence

Each session is saved to a user defined database. All database operations are handled by the persistence manager. One table, web_session, is required to hold all of the sessions for the server. This table has a single primary key whose value is the sessionId. The table consists of four VARBINARY columns and one BLOB. It also has a column containing the session version. The database only needs to be updated if the session has changed at the end of the request. Database persistence uses the *session.db* data source, which is an alias for the “default” data source in the as-installed configuration.

Understanding HTTP response caching options

EAServer supports caching of static content and servlet responses.

Servlet response caching

When caching is enabled for servlets and JSP Web components, EAServer checks for a cached response before calling the Web component. For servlets and JSPs that are called often, caching improves performance by skipping the processing required to produce the response. EAServer supports three mechanisms for response caching:

- **Dynamic response caching** Responses are cached in a hash table, using a multi-part key. By default, the key includes the request path, but you can configure additional key parameters such as request or session attributes. “Dynamic response caching” on page 73 describes how to configure this mechanism.
- **Partial response caching** allows you to cache parts of a response. This mechanism is useful when pages contain volatile content, such as calculation results, but otherwise have static content such as headers and footers. The response cannot be cached effectively using other mechanisms because of the volatile content, but partial response caching allows you to cache only the static parts of a response. Partial response caching is supported by a tag library for use in JSPs, and a public API for use in servlets. “Using partial response caching” on page 76 describes how to use this mechanism.

Which components should use caching?

Not all Web components should be cached. Caching the output of seldom-called Web components can sometimes reduce performance. If the cache is full, the rarely accessed output can bump more frequently accessed data out of the cache. On the other hand, if a servlet takes a long time to execute, you may still benefit from caching a servlet that it is not called as frequently as others, as long as there is sufficient space to cache the servlet. When the cache is too full to add or refresh a response, EAServer removes enough entries to make room, removing entries in least-recently-used order.

There is some overhead required to create and remove cache entries. If a Web component runs quickly, you may get better results with caching disabled, thus avoiding the overhead of maintaining additional cache entries and reserving more memory for the caching of other Web components.

To decide which Web components should be cached, review your request log patterns and Java profiling data (or timing trace data) to answer the following questions:

- How often is the Web component invoked?
- How long does it usually take?

- How often can requests use the cached data? If not always, can you define a key based on request and session parameters to allow the correct response to be cached and reused? Does a timeout suffice to satisfy the requirements for accurate data?

Based on these answers, you can determine which Web components are appropriate to cache and estimate the time that can be saved by caching them. For example, if you specify a timeout of 1 minute, the response takes 5 seconds to process, and the matching request occurs 4 times per minute, you can eliminate up to 15 seconds of processing time per minute (based on the fact that there are 3 cache hits per minute before the matching entry times out and must be recalculated).

JSPs must be listed as Web components in the deployment descriptor

To enable caching, you must define EAServer Web components for JSPs as well as servlets. Although a JSP can run when it is not installed as a Web component, you cannot enable caching unless you have defined a Web component that is mapped to the JSP. To ensure JSPs are defined as Web components when deploying to EAServer, add configuration for them to the *web.xml* deployment descriptor before deploying to EAServer.

Dynamic response caching

Dynamic response caching decreases a servlet's or JSP's response times by caching the output with a multi-part, user-configured key value. This caching mechanism stores responses in their entirety. For pages that return both volatile content and content that rarely changes, use partial response caching instead.

When response caching is enabled for a servlet or JSP, EAServer checks the cache before invoking the Web component, looking for an entry that matches the key that you have defined for the servlet or JSP. If an appropriate response is found in the cache, EAServer returns the contents of the cache, instead of calling the servlet. If the cache contains no matching key, EAServer invokes the servlet, and caches the response and response headers while returning them to the client.

You can define the key that EAServer uses to store and retrieve cached entries. By default, a key consists of only the servlet's or JSP's location on disk. You can further refine key values by adding up to five optional parameters. Doing so allows caching of separate responses from the same servlet or JSP, based on request characteristics such as locale or HTTP session ID. In addition, you can configure a timeout for cache entries associated with the JSP or servlet to prevent the use of stale data.

Configuring response caching for servlets and JSPs

Configuration of this caching is done in the *webapp.xml* configuration file. You can create a *sybase-webapp-config.xml* using the development tool of your choice that adds the necessary caching attributes to the Web application's configuration file.

❖ **To configure response caching for a servlet or JSP:**

1. Modify the *sybase-webapp-config.xml* file for the corresponding Web application, located in the *deploy\webapps\web_app_name\WEB-INF* subdirectory of your EAServer installation. For example, if the Web application is named *testwebapp*, then you would edit the *deploy\webapps\testwebapp\WEB-INF\sybase-webapp-config.xml* file.
2. Add the cacheResponse attribute to the XML file, and the parameters that you want to use in the key as listed in Table 5-1.

Table 5-1: response caching properties

Parameter name	Description
timeOut	Specifies the timeout in seconds for a cache entry; the default is 60; a value of 0 indicates no timeout. This property is cacheTimeout when specified in the <cacheResponse> Ant property.
size	Specifies the number of entries that can be cached. If not specified, the default is 1000.
sessionLocal	Includes the session ID from the request as part of the key. This allows session specific items such as shopping carts to be differentiated from each other. To enable the sessionLocal parameter, set it to true. The default is false.
localeSensitive	Allows you to include which languages are accepted as part of the key. The key is constructed from the list of locales in the request. The ability to include locale information is also included in the requestHeaders option, but for ease of use, it has been added as a separate option. Enable this option by setting it to true (the default).
requestParameters	Allows certain parameters from the request to be included in the key. The request contains a table of key-value pairs that are accessible from within a servlet or JSP from which the servlet developer can base the output of the servlet. This means that the same servlet, when given different parameters, may produce different output. Hence, two separate entries in the cache. You can specify as part of the cacheResponse attribute which attributes are to be included in the key. Since the attributes are stored as key-value pairs, both the key name and key value affect the key. The included parameters are listed in the requestParameters parameter of the cacheResponse attribute. To include all parameters, use an asterisk "*" (the default) in place of the list.

Parameter name	Description
sessionAttributes	Allows certain session attributes from the session to be included in the key. Session attributes are stored in the key as key-value pairs. The attributes are specified by the sessionAttributes parameter of the cacheResponse attribute. To include all attributes, use an asterisk "*" in place of the list. In order to guarantee that the cache can store a session attribute, the type being stored needs to be serializable. The default is not to use sessionAttributes.
requestHeaders	Allows certain request headers from the request to be included in the key. For example, you could specify the date header to be included so that only entries in the cache whose date header matches the current date header are considered. Headers are case insensitive so the key is converted to lowercase. Headers are stored in the key as key-value pairs. The included headers are listed in the requestHeaders parameter of the cacheResponse attribute. To include all headers, use an asterisk "*" in place of the list. The default is not to use requestHeaders.

3 Deploy (or redeploy) the Web application.

Caching an entire tree

To use response caching for a JSP that forwards request to, or dynamically includes, other JSPs or static files, consider these factors. By default, when you enable response caching for a JSP Web component, only its content is cached. If a JSP includes, or forwards requests to, other pages or static files, their output is not cached. To include the output of all the pages or files that are invoked, you can select to cache a Web component's entire tree. This example illustrates portions of three JSP files; two use the <jsp:include> tag to include other JSPs:

```
// page1.jsp
<HTML>
<H1>This is page 1</H1></p>
<jsp:include page="/page2.jsp" />
</HTML>

// page2.jsp
<HTML>
<H2>This is page 2</H2></p>
<jsp:include page="/page3.jsp" />
</HTML>

// page3.jsp
<HTML>
<H3>This is page 3</H3></p>
</HTML>
```

If you enable response caching for the Web component mapped to *page1.jsp and choose to cache the entire tree*, the cached entry displays this in the browser:

```
This is page 1
This is page 2
This is page 3
```

If you enable response caching for *page1.jsp* but not for the entire tree, the cached entry displays this in the browser:

```
This is page 1
```

When a client requests the Web component mapped to *page1.jsp* and it is configured to cache the entire tree, the output from *page1.jsp*, *page2.jsp*, and *page3.jsp* is cached as a single entry. EAServer creates a separate cache entry for a single page when:

- A client requests the page directly, and
- The Web component is configured to not cache the entire tree.

For example, if the Web component "Page2" is mapped to *page2.jsp* and it is not configured to cache the entire tree, its output is cached as a separate entry when a client specifically requests Page2.

Changes from EAServer 5.*x*

Caching in EAServer 6.0 does not support the messageTopics key parameter that was supported in EAServer 5.*x*. This parameter is ignored when you migrate your EAServer 5.*x* JSPs and servlets.

The default value for localeSensitive has changed from false to true. The new default is applied when you migrate your EAServer 5.*x* JSPs and servlets.

In EAServer 6.0, the caching parameters are configured in the Web application's XML configuration file as opposed to the *.props* file used in EAServer 5.*x*. Also, There is no longer a Web application level default for the parameters. Each parameter has its own default value.

Using partial response caching

There are two methods for caching dynamic content from JSP and Servlets:

1 The entire response

2 Part of the response

Fragment caching, also known as partial response caching allows you to cache parts of a response. This allows you to create “template pages” that contain static and dynamic data separated into individual fragments. As with regular dynamic response caching, you control different caching parameters. By configuring each fragment to be sensitive to different page parameters, the page designer ensures that you do not get stale or incorrect data.

Fragment caching is more effective than caching entire responses when pages contain volatile content. For example, you can use fragment caching when the response contains volatile content, such as calculation results, mixed with static content such as headers and footers.

Partial response caching is supported by a tag library for use in JSPs, and a public API for use in servlets. “Using the caching tag library” on page 77 describes how to use the tag library. Class CacheManager on page 80 describes the API for use in servlet code.

Using the caching tag library

The tag library implementation is provided in *CacheTags.jar*, installed in the *lib/default/ext* subdirectory of your EAServer installation. To use the library in a JSP, add the following directive:

```
<%@ taglib uri="http://www.sybase.com/EAServer/cachetags.tld" prefix="ct"%>
```

The library includes the tags described below.

The cache tag

To cache a portion of a page, surround it with this tag, as in:

```
<prefix:cache attributes>
... page content ...
</prefix:cache>
```

Where *prefix* is the tag prefix that you assigned the tag library when declaring it in the taglib directive in your page source, and *attributes* is a list of attribute-value pairs to set the attributes described in Table 5-2.

Table 5-2: Cache tag attributes

Attribute	Comments
parameters	A comma-delimited list of request parameters to include in the key. A value of "*" includes all parameters in the key. If not specified, all parameters are included in the key.
attributes	A comma-delimited list of session attributes to include in the key. A value of "*" includes all session attributes. If not specified, no session attributes are included in the key.
localeSensitive	Set this attribute to true if locale-sensitive headers are to be included as part of the key. The default is false, which omits locale-sensitive headers from the key.
headers	A comma-delimited list of request headers to include in the key. The default is to include no headers in the key.
timeout	Specifies how long, in seconds, an entry in the cache remains valid. The default value is 600.
name	Allows you to specify a unique name, so that a cache can be shared across multiple pages. If you do not specify a name, the default value is computed so that each page has one cache for all the tags within that page, and each occurrence of the cache tag is assigned an ID that is unique within the page. You can specify a name to cache parts of a response that occur on several pages: data computed on one page can be read from cache and used in another page.
scope	Specifies the scope in which data is stored in the cache. Can be either session or application. The value session indicates that only pages in the same session can view the cached data. The default, application, indicates that all pages in the Web application have access to the cached data.
maxEntries	Specifies the a size (number of entries) in the cache. If not specified, the default is 1000.

When recompiling a JSP, EAServer flushes any cache entries that are used in the page. When refreshing the Web application, EAServer refreshes all caches that are scoped to the application. You can also flush caches programmatically using the flushCacheByKey or flushCacheByScope tags.

Flushing caches

The flushCacheByKey and flushCacheByScope tags allow you to flush key entries from a cache:

- flushCacheByKey – causes the specified entries to be erased from the cache, and can only be used on caches that specify the name attribute.

- flushCacheByScope – removes entries from all the caches in a specific scope.

The flushCacheByKey tag

You can use this tag to flush caches for which you have specified a name. You can specify a name, scope, and key parameters as described in Table 5-3. The entry that matches the specified key values and scope is flushed when the tag executes.

Table 5-3: flushCacheBykey tag attributes

Attribute	Required	Comments
name	Yes	The cache name.
scope	No	The cache scope. If not specified, the default is application.
parameters	No	Same as for the cache tag. The default is to include all request parameters in the key.
attributes	No	Same as for the cache tag. The default is to not include any session attributes in the key.
localeSensitive	No	Same as for the cache tag. The default is false.
headers	No	Same as for the cache tag. The default is to not include headers in the key.

The flushCacheByScope tag

You can use this tag to flush all entries from all caches in the specified scope.

Table 5-4: flushCacheByScope tag attributes

Attribute	Required	Comments
scope	Yes	Specifies which scope to refresh the caches in. Values are session, application, and page. Specify application to flush all caches in the application. Specify session to flush all caches that are scoped to the user's session. Specify page to flush all cache entries that are used in the current page.
uri	No	The URI from which the caches are flushed. The default is the URI of the current page.

Using the caching API

You can call the caching API to cache response parts in servlets. The API is implemented by class CacheManager, described below.

Class CacheManager

Description

```
package com.sybase.djc.web.util.jsptags
public class CacheManager
```

Allows you to cache responses or parts of a response in Java servlets.

Constructors

None. Call the CacheManager.getInstance(ServletContext) method.

CacheManager.getInstance(ServletContext)

Description

Gets the instance of the CacheManager for a given servlet context. Each context has a single CacheManager instance.

Syntax

Package	com.sybase.djc.web.util.jsptags
Class	CacheManager

```
public static CacheManager getInstance(ServletContext context)
```

Parameters

context
The servlet context.

Return value

The CacheManager instance for the context, or null if the specified context is not a valid EAServer servlet context.

CacheManager.createCache(String)

Description

Creates a new cache.

Syntax

Package	com.sybase.djc.web.util.jsptags
Class	CacheManager

```
public void createCache(String cacheName) throws CacheNameException
```

Parameters

cacheName
The name of the cache to create. The method throws CacheNameException if the name is invalid.

CacheManager.getData(String, PageCacheKey)

Description

Retrieves data from the cache.

Syntax

Package	com.sybase.djc.web.util.jsptags
Class	CacheManager

public String getData(String cacheName, PageCacheKey key)
throws CacheNotFoundException, CacheNameException

Parameters

cacheName
The name of the cache to use.

key
The key for the entry. Call getCacheKey(HttpServletRequest, String, String, String, String, String, boolean, int) to get a key instance.

Return value

The cached text, or null if no entry matches the key. Throws CacheNotFoundException if there is no cache with the specified name. Throws CacheNameException if the name is invalid.

CacheManager.putData(String, PageCacheKey, String, int)

Description

Places data in the cache.

Syntax

Package	com.sybase.djc.web.util.jsptags
Class	CacheManager

public void putData(String cacheName, PageCacheKey key, String data, int timeout) throws CacheNotFoundException, CacheNameException

Parameters

cacheName
The name of the cache to use.

key
The key for the entry. Call getCacheKey(HttpServletRequest, String, String, String, String, String, boolean, int) to get a key instance.

data
The text to cache.

timeout
The timeout for the entry, in seconds.

CacheManager.flushCacheByKey(String, PageCacheKey)

Description

Flushes the entry for the specified key from the cache.

Syntax

Package	com.sybase.djc.web.util.jsptags
Class	CacheManager

public void flushCache(String cacheName, PageCacheKey key) throws CacheNotFoundException, CacheNameException

Parameters

cacheName
The name of the cache to flush from.

key
The key for the entry to flush. Call getCacheKey(HttpServletRequest, String, String, String, String, String, boolean, int) to get a key instance.

CacheManager.flushCacheByScope(HttpServletRequest, String, String)

Description

Flushes caches associated with the specified scope.

Syntax

Package	com.sybase.djc.web.util.jsptags
Class	CacheManager

public void flushCacheByScope(HttpServletRequest request, String scope, String uri)

Parameters

request
The request associated with the page that you are caching content for.

scope
A value from the following table:

Value	**To indicate**
application	To flush all caches in the Web application.
session	To flush all caches whose session ID matches the session ID of the current session. If there is not an active session, nothing is flushed.
page	Flush all caches for the specified page.

uri
The URI of the page from which to flush the cache.

CacheManager.getCacheKey(HttpServletRequest, String, String, String, String, String, boolean, int)

Description

Creates a cache key for the specified inputs.

Syntax

Package	com.sybase.djc.web.util.jsptags
Class	CacheManager

```
public PageCacheKey getCacheKey(HttpServletRequest request, String
parameters, String attributes, String headers, String scope, Boolean
localeSensitive, String tagID)
```

Parameters

request
The request associated with the page that you are caching content for.

parameters
A comma-separated list of request parameters to include in the key. Specify "*" to include all parameters.

attributes
A comma-separated list of session attributes to include in the key. Specify "*" to include all attributes.

headers
A comma-separated list of request headers to include in the key. Pass as null to omit all headers from the key.

scope
Controls the scope in which data is stored in the cache. Pass a value from the following table:

Value	To indicate
application	All pages in the Web application have access to the cached data.
session	Only requests in the same session can view the cached data.

localeSensitive
Pass as true if locale-sensitive headers are to be included in the key, and false otherwise.

tagID
A string containing the unique tag ID for the tag.

Return value

A com.sybase.djc.web.util.jsptags.PageCacheKey instance containing the input data.

CHAPTER 6 Database Access Tuning

This chapter describes how to tune data sources and the settings that affect the performance of the EAServer transaction manager.

Component design and implementation

The design and implementation of your code to access databases can have a significant effect on performance. Be sure to understand how the server manages transactions for your component model. For EJB components, EAServer implements the standard container-managed transaction semantics defined in the EJB specification. For CORBA and PowerBuilder components, EAServer implements the transaction semantics described in Chapter 2, “CORBA Component Life Cycles and Transaction Semantics,” in the *CORBA Components Guide*.

Keep transactions short

Avoid component designs that require the use of long-running transactions. For each transaction that your application runs, the database server may lock tables, rows, indexes, and other resources required to guarantee the required transaction outcome. Long-running transactions reduce the scalability of the application, since the required locks may be held for the duration of the transaction and other users must wait for them to be released.

Many design patterns that depend on long-running transactions can be easily modified to use optimistic concurrency control and short transactions. That is, rather than running all the database work in one transaction, select the initial values and perform all computations without starting a transaction. Use a timestamp or value comparisons before updates to verify that data has not been modified since it was first selected.

For EJB components, minimize the use of bean-managed transactions. If you do use bean-managed transactions, avoid implementations that allow the transaction to remain open when a method returns.

For CORBA and PowerBuilder components, avoid stateful components that are transactional and call the continueWork or disallowCommit state primitives, and other designs that require transactions to span client method invocations. If the transaction remains open when the business method returns, it can remain open if the client hangs or the user changes their mind.

Note Beginning in version 6.0, you can disable support for long-running transactions started by CORBA stateful components. For details, see “Long versus short transactions” in Chapter 2, “CORBA Component Life Cycles and Transaction Semantics,” in the *CORBA Components Guide*.

Minimize result set size

Tune your queries and schemas to ensure that you do not waste network resources and memory by selecting unneeded data. For example, do not select 100 rows, then search them in your component to find the one row that you need. Use the query language to direct the database to find and return only the data you need. In EJB entity beans, implement additional finder methods to allow results filtering with database query language rather than returning many rows to be filtered on the client.

In CORBA components that return large result sets to the client, you may get better performance by batching the result set into smaller groups of rows, then reassembling them on the client. Doing so avoids the need to construct large TabularResults.ResultSet objects in memory.

Minimize use of two-phase commit

Multiple database transactions require two-phase commit, and consequently execute more slowly than those that use only a single database. Review your application design and component transaction settings to make sure that two-phase commit is used only when the component work involved must be part of the same atomic unit of database work.

If a component inherits a transaction in an intercomponent call involving two or more database connections, EAServer uses two-phase commit. The component's transaction attribute determines whether transactions can be inherited through intercomponent calls. For example, two-phase commit is required if the component's transaction attribute is "Supports," the component has been called from another component that has attribute "Requires," and the components use different data sources.

To avoid use of two-phase commit for a component's database work, set the transaction attribute to "Requires New" after verifying that the work can be commit or rollback independently of the calling components transaction outcome. If a component performs updates to a noncritical database you can choose "Not Supported" as the component's transaction attribute to eliminate the overhead of using EAServer transactions at all. For example, the component may log usage statistics to a remote database.

Clean up connections before releasing them to the data source

Many JDBC programs do not explicitly clean up java.sql.Statement objects. Instead, they rely on the JDBC driver to clean up Statement objects when the connection is closed. This strategy does not work with pooled connections; you must explicitly clean up Statement objects before releasing a connection back into the pool. To clean up Statement objects, call Statement.close() and set the Statement reference to null.

Warning! To prevent memory leaks, you must explicitly clean up a connection's Statement objects before releasing the connection back into the pool. Do not release the same connection more than once.

Avoid unnecessary database work

For PowerBuilder and CORBA components that participate in transactions, you can call isRollBackOnly to test if the transaction is doomed before the method executes more logic that would have to be rolled back. For more information, see “Using transaction state primitives” in Chapter 2, “CORBA Component Life Cycles and Transaction Semantics,” in the *CORBA Components Guide*.

Server and component transaction settings

These server properties affect the performance of the EAServer transaction manager and components that use server-managed transactions.

Stateful component idle timeout

If your application includes transactional stateful CORBA or PowerBuilder components or EJB stateful session beans, configure an idle timeout property for the component. Doing so ensures that if a client crashes or an end user leaves their workstation, transactions do not remain open indefinitely.

To configure a timeout for stateful session beans, set the ejb.passivateTimeout property as described in Chapter 2, “Deploying and Configuring EJB Components,” in the *EJB Users Guide*.

To configure a timeout for stateful CORBA or PowerBuilder components, set the Passivation Timeout as described in Chapter 4, “Managing CORBA Packages and Components,” in the *CORBA Components Guide*.

Long transaction support

You can configure EAServer so long transactions cannot be run from CORBA and PowerBuilder stateful components. Doing so decouples the transaction outcome and duration from the component's invocation of transaction state methods. Long transactions are enabled by default. Before disabling long transactions, verify the change does not compromise the transactional integrity of your application.For more information, see "Long versus short transactions" in Chapter 2, "CORBA Component Life Cycles and Transaction Semantics," in the *CORBA Components Guide*.

Data source settings

Data source use increases performance by allowing reuse of database connections, eliminating the overhead of repeatedly creating and destroying connections to the same database. For general information on data sources, see:Chapter 4, "Database Access," in the *EAServer System Administration Guide*.

The following sections describe how to tune data source settings for the best performance.

Tuning the pool size

Data sources have 10 connections by default. For applications with many clients, this number is often too small. For lightly used data sources, you can lower the size to free up memory and network connections that would be wasted by rarely used database connections. To tune the pool size, monitor the data source statistics as described in Chapter 11, "Runtime Monitoring," in the *EAServer System Administration Guide*. Tune the pool size by setting the properties listed in Table 6-1.

Table 6-1: Properties to configure data source pool size

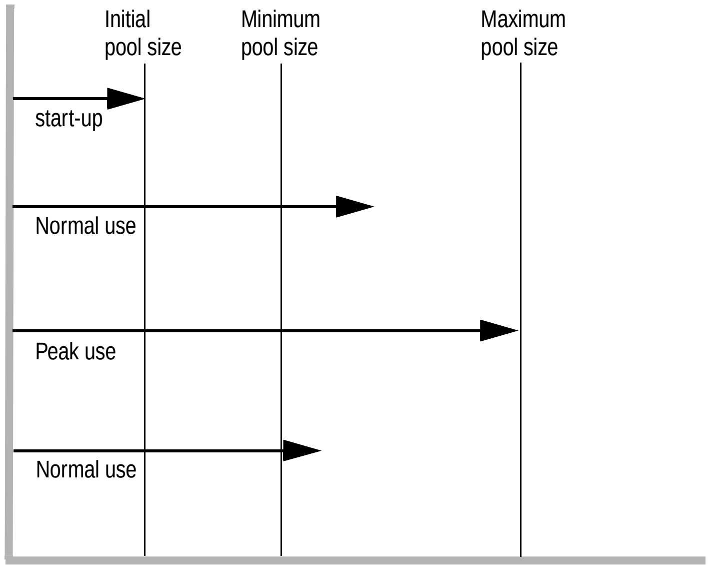
Property	Description
Initial Pool Size (initialPoolSize)	The initial number of pooled connections, allocated at server start-up. If not set, the default is 0.

Property	**Description**
Minimum Pool Size (minPoolSize)	The minimum number of connections in the pool. When open connections are idle, the pool is pruned to this size. The default is zero. If no minimum size is specified, EAServer opens connections as-needed to fill the pool up to the maximum size.
Maximum Pool Size	The maximum number of connections allocated to the pool for this data source. If the maximum is exceeded, and cannot be resolved by waiting; for example, if deadlock occurs, you may see “ResourceMonitorTimeoutException” errors in your log file, and some transactions may fail. A value of zero sets no limit to the connection pool size.
Maximum Wait Time	The maximum number of seconds to wait for a connection before the request is cancelled.
Maximum Idle Time	Specifies the number of seconds an idle connection remains in the pool before it is dropped. The default is 60 seconds. Idle connections are dropped until the minimum pool size is reached. **Note** If the idle timeout is set to 0, connections are not dropped and can remain in the pool until the server shuts down.
Maximum Cached Statements	The maximum number of JDBC prepared statements that can be cached for each connection by the JDBC driver. The value of this property is JDBC-driver specific.

Set the pool size so the majority of database connections are taken from the data source. You can tune the minimum pool size and idle time parameters to reduce the number of database connections that are held during off-peak hours. You can raise the maximum size if you see many failed connection requests or waits.

Figure 6-1 illustrates how these settings affect the growth of the data source pool size.

Figure 6-1: Data source pool growth patterns



When the server starts, it preallocates the specified inital number of connections, allowing faster response times to the initial client requests that require a database connection.

If all connections are in use simultaneously, but the pool size is below the maximum, the pool manager creates new connections to satisfy demand. When released, these connections are added to the pool, causing it to grow towards its maximum pool size.

During peak use, requests may arrive when the pool contains the maximum number of connections, but all connections are in use. When this happens, the requesting component waits. If the wait time exceeds the configured maximum (Maximum Wait Time), the request fails with exception `com.sybase.djc.util.ResourceMonitorTimeoutException`.

When the activity level drops, the pool manager removes idle connections if you have configured an idle connection timeout (Maximum Idle Time). The pool size drops back down to the minimum.

Monitoring data source pool activity

You can monitor the data source activity in the Management Console to determine how effective the settings are. See “DataSource tab” in Chapter 11, “Runtime Monitoring,” in the *EAServer System Administration Guide* for more information.

Tuning data sources used by EJB CMP entity beans

For EJB CMP entity beans, EAServer provides wrapper drivers that improve performance by using statement batches and stored procedures. For more information, see “Use JIT JDBC wrapper drivers” on page 53.

Remove unused data sources

Remove unused data sources from the server’s resource startup list, or set the Initial Pool Size setting to 0. EAServer allocates the minimum pool size for each data source, and unused connections waste memory and network resources.

Data source ping

By default, the data source manager runs a stock query to verify that connections are ready for use before placing them back in the data source pool. Sanity checking prevents errors that occur when components release a connection that is not ready for use by another component. For example, there may be pending results on the connection, causing an error when the next component to use the connection tries to send a command.

If you have debugged the results handling in your application, you can improve performance by disabling Ping Pooled Connections in the data source properties.

If you have enabled the Set Session Authorization property so database work runs under the effective user ID of the client, you can also enable the Ping With Set Session Auth property (pingAndSetSessionAuth). Doing so causes the ping and session-authorization commands to be run in a single command batch and may improve performance.

Using the caching APIs

In Java/CORBA and CORBA/C++components, you must use the EAServer APIs to obtain a data source reference and retrieve connections from it. For best performance, use by-name lookup to obtain data source handles and pass the data source handle when obtaining and releasing connections. Doing so avoids internal table searches in the data source manager. For information on using these APIs, see the *API Reference Manual*.

Dynamic prepare on jConnect data sources

Ensure that data sources that utilize a com.sybase driver class are defined with the DYNAMIC_PREPARE property set to FALSE for optimal performance. In EAServer 4.1.1 and later, this property is set to FALSE by default. However, it was set to TRUE by default in some earlier versions.

Database driver specific settings

See the documentation for your database and the connectivity driver or library for performance tuning recommendations. For example, if you are using Sybase jConnect for JDBC, the *Programmer's Reference* includes a chapter on performance and tuning. This document is available in the jConnect documentation on the Sybase Web site at http://sybooks.sybase.com/jc.html.

Unless you are actively debugging problems, ensure that the trace and debug settings are disabled for your connectivity driver or library.

Database tuning

Consult your database performance and tuning documentation for information on tuning your queries to minimize database response time. Take advantage of any performance features available in your database, such as stored procedures if using Sybase Adaptive Server Enterprise. Consult your database performance and tuning documentation. If you are using Sybase Adaptive Server Enterprise, the *Performance and Tuning Guide* is available on the Product Manuals Web site at http://sybooks.sybase.com/nav/summary.do?prod=9938&lang=en&prodName=Adaptive+Server+Enterprise.

Transaction cross-reference logging

You can configure EAServer to create a log of transaction cross references. For each distinct container managed transaction (identified by the business method that initiates the transaction), the log includes information about SQL statements and JMS operations that were executed in the transaction.

To create the transaction cross reference, enable Generate Transaction Cross Reference in the server property pages in the Management Console, or set the txRef property to true if using Ant configuration. This setting causes EAServer to collect the necessary data and write it to the log files using the TxRef scheduled task.

The transaction cross reference output is recorded in the EAServer *logs/transactions* directory, in file *my-server-name.transactions*, where *my-server* is the server name.

CHAPTER 7 Cluster Tuning

This chapter describes the performance benefits of EAServer clusters and tells you what settings to tune for the best performance when your application runs in a cluster.

When to use clusters

An EAServer *cluster* is a group of servers that share replicated repository information to run the same components and Web applications. A clustered deployment provides load balancing and high availability, at the cost of slightly increased overhead to replicate client session information between servers in the cluster. If you are not familiar with these concepts, see these chapters in the *EAServer System Administration Guide*:

- Chapter 6, “Clusters and Synchronization”
- Chapter 8, “Load Balancing, Failover, and Component Availability”

If your application cannot support the required number of clients running on one machine, moving to a cluster allows EAServer to balance the load across several machines. Depending on the hardware you choose, a cluster of low priced machines may be less expensive than upgrading to a single machine with multiple CPUs. Clusters also provide failover support when you run servers on multiple machines: no single machine failure takes your application offline.

Clusters incur a slight overhead increase due to the need to store client session data to a remote database for access by multiple servers. You can minimize the performance impact by minimizing your use of stateful components and HTTP session storage, and by tuning the database and data source used for session storage.

Cluster partitioning

A cluster partition allows you to divide the application load among the servers in a cluster. You can configure individual components, message queues, and topics to be serviced only by the servers in a specific partition. For details, see "Cluster partitions" in Chapter 6, "Clusters and Synchronization," in the *System Administration Guide*.

IIOP client settings for clustered applications

When deploying clients in a cluster, these settings can affect the distribution of server load and the client's ability to fail over if a server goes off line.

Socket timeout for Java clients

For EJB and CORBA/Java clients, this setting specifies a time limit to receive a server response before the connection fails over to try another server in the cluster. Setting this property ensures that failover happens without an unreasonable delay. Specify the timeout period in seconds. The default of 0 indicates no time limit. The following table describes how to set this property for each client type:

Client type	Property name
Java	com.sybase.CORBA.socketTimeout
EJB	com.sybase.ejb.socketTimeout
C++ and PowerBuilder	Not supported.

Idle connection timeout for C++ and PowerBuilder clients

In a cluster, EAServer's load balancing policy can evenly distribute the initial client connections. However, long running C++ and PowerBuilder clients can create unbalanced loads by building an affinity for the server that they are initially directed to by the name service.

To avoid this problem, configure the ORBidleConnectionTimeout C++ ORB property. This property specifies the time, in seconds, that a connection is allowed to sit idle. When the timeout expires, the ORB closes the connection. The default is 0, which specifies that connections can never timeout. The connection timeout does not affect the life of proxy instance references; the ORB may close and reopen connections transparently between proxy method calls. Specifying a finite timeout for your client applications can improve server performance. If many instances of the client run simultaneously, a finite client connection timeout limits the number of server connections that are devoted to idle clients. A finite timeout also allows rebalancing of server load in an application that uses a cluster of servers. You can also configure this property by setting the environment variable JAG_IDLECONNECTIONTIMEOUT.

Web application settings

Web applications in a clustered deployment can provide better performance since multiple machines can handle more load than one. Clusters also provide high availability: if one machine goes off-line, clients can connect to another server in the cluster. To run your Web application in a cluster, you must configure a mechanism to support load balancing of HTTP requests, and optionally failover. For more information, see "Deploying Web applications to a cluster" in Chapter 8, "Load Balancing, Failover, and Component Availability," in the *EAServer System Administration Guide*.

Chapter 5, "Web Application Tuning," describes the Web application settings that you can tune for single-server deployments. In a clustered deployment, the the HTTP session replication mechanism can also be tuned.

In a clustered deployment, EAServer replicates user session data stored in HTTP sessions so that the same data is available on other servers in the cluster. Replication uses the remote database server specified by the session.db data source. All servers in the cluster store session data in a remote database. Tune the data source settings as described in "Data source settings" on page 89.

Component settings

Chapter 3, “Component Tuning,” describes the component settings and implementation techniques that you can tune for single-server deployments. In a clustered deployment, these additional settings affect performance.

Automatic failover

To allow load balancing when deployed in a cluster, your components must be configured to support automatic failover.

Enterprise JavaBeans can be configured by setting the ejb.automaticFailover Ant property—see “Commonly configured properties” in Chapter 2, “Deploying and Configuring EJB Components,” in the *Enterprise JavaBean User’s Guide*.

CORBA and PowerBuilder components can be configured by setting the Automatic Failover property in the Component Properties pages in the Management Console.

Response and request logs

For components deployed to a cluster, you can configure request and response logging to avoid duplicate calls to business methods. These repeated calls might occur when the client has failed over from one server to another, and is uncertain whether the call was successfully executed on the first server before failure.

Request logging can prevent repeated calls to methods that return void. Enable request logging by calling the <requestLog> command inside a <setProperties> command that configures your component.

Response logging can prevent repeated calls to methods that return a value (not void). Enable request logging by calling the <responseLog> command inside a <setProperties> command that configures your component.

Request and response logging uses the af_response_log database table in the cluster.db data source. Request and response logging will normally degrade performance due to the use of a database. However, there use is appropriate if possible duplication of method calls is unacceptable.

The following example shows how to configure automatic failover, request logging, and response logging from an Ant user-configuration script:

```
<setProperties component="ejb.components.myjar.MyCompRemote">
  <automaticFailover enable="true"/>
  <requestLog enable="true"/>
  <responseLog enable="true"/>
</setProperties>
```

The <setProperties> component attribute specifies the name of the DJC component that runs the application component that you want to monitor. Set this depending on the application component type, as follows:

- For enterprise JavaBeans components, specify the DJC component that corresponds to the remote interface. You can read the DJC component names from the EJB module's Ant configuration file that was generated by deployment.
- For CORBA and PowerBuilder components, specify the DJC component that corresponds to the remote interface of the EJB session bean that wraps your component. This name is `ejb.components.`***`package`***`.`***`component`***`Remote`, where *package* is the CORBA package name, and *component* is the component name.

C H A P T E R 8

Message Service Tuning

About the message service

EAServer supports the Java Message Service APIs and implements the required server functionality. Before reading this chapter, you should be familiar with message service configuration and programming, as described in these chapters in the *Java Message Service User's Guide*:

- Chapter 1, "Message Service Overview"
- Chapter 2, "Setting up the Message Service"
- Chapter 3, "Developing JMS Clients"

Best practices for design and coding

For best performance, follow these recommendations when designing and implementing JMS applications:

- **Consider storage types carefully** EAServer supports two options for message storage and delivery, transient versus persistent:

- Transient messages are stored in memory only. Applications based on transient messages run faster, however messages can be lost if the server goes offline or restarts.
- Persistent messages are stored in a remote database and also cached in memory. The database interaction affects performance. However, since the messages are stored in a database, they are not lost when the server goes offline or restarts (reliable delivery depends on the settings described "Queue and topic settings" on page 104).

Use transient messages if your application requirements allow the possibility of lost messages. Use persistent messages if lost messages are never acceptable. For example, transient storage may suffice if the message is intended to notify retail customers of new catalog items. On the other hand, if the message represents a change to a customer's account balance, use persistent storage for the most reliable delivery.

Transient messaging yields much better performance than persistent messaging for high-volume publish/subscribe applications such as stock or currency trading systems. Sybase recommends that you design these applications to work reliably with non-persistent messages. If messages might have been are lost, your application receives a JMSException error. If the applications takes appropriate action to cope with the possible loss of messages, you can use transient messaging.

- **Use message selectors** If using the publish/subscribe model, message selectors save bandwidth by preventing the delivery of messages that the subscriber does not need. Do not scan and delete messages in your client code. Instead, create a selector so that the server filters messages before they cross the network.
- **Start consumers before producers** Messages that you send before a consumer is available can create backlogs in the queue. If you can control the timing, make sure the consumer starts first.
- **Set the expiration time** If appropriate, set the message time-to-live property. Doing so allows EAServer to free resources associated with the message when it expires. For example, in an automated trading application, you might set the time-to-live to 10 seconds for price-change messages, assuming this is the acceptable window for execution of trades that result from message receipt.
- **Set message priority** If some messages must be delivered ahead of others, set the message priority property. Priority values are application assigned relative values ranging from 0 to 9. You must use them consistently in your application.

- **Minimize message size** Longer messages consume more network bandwidth and take longer to process in memory. Design your message formats to eliminate unnecessary data.
- **For large message values, consider using compression** Message compression can reduce network overhead and, for persistent messages, reduce database overhead.

  In JMS clients, compress data sent over the network by setting the com.sybase.jms.dataCompression InitialContext property. Alternatively, use a JMS Provider instance and set the dataCompression property. For details, see Chapter 3, "Developing JMS Clients," in the *Java Message Service User's Guide*.

  For persistent messaging, you can also configure the queue or topic properties to enable compression before persisting the message. See "Queue and topic data compression" on page 107.

  Test system performance to determine whether enabling network and persistent message compression increases throughput. While compression can reduce database and network overhead, it increases CPU usage by EAServer and your client application.
- **Clean up unused JMS resources** Close resources such as connections, sessions, queues, and topics as soon as you no longer need them.

Data source and database tuning

You can specify the data source and database table name used for persistent storage in the message queue and topic properties. A database is required even for applications that use transient messaging. Among other things, EAServer requires the data source to store persistent messages, to store transient messages when the in-memory queue overflows, for temporary storage of transient message queues that are idle.

The default data source for messaging is the `message.db` data source. You can reconfigure or redefine this data source. To support development use, the default message service configuration connects to the Adaptive Server Anywhere database server runtime that is included with EAServer. For large scale production use, Sybase recommends that you use an enterprise-grade database server such as Sybase Adaptive Server Enterprise.

If your application uses messaging in transactions involving other components, such calling EJB entity bean method in the same transaction, consider running the message service against the same server that hosts the application data. Doing so allows you to avoid the added overhead of two-phase commit transaction management. When transactions enlist multiple data sources, all the data sources must be configured for two-phase commit to achieve atomicity between message queue operations and other resource access.

Tune the `message.db` data source as described in “Data source settings” on page 89.

The default database table in message queue and topic properties is jms_pm. You may get better performance specifying dedicated tables for each queue and topic. EAServer creates the specified table with appropriate indexes. For large-scale use, the table properties may benefit from further tuning by your database administrator.

Queue and topic settings

These settings affect the performance of the application’s message consumers and message producers. You can configure them in the message queue and connection factories that you create in the Management Console or from an Ant configuration script.

For details on configuring these settings, see Chapter 2, “Setting up the Message Service,” in the *Java Message Service User’s Guide*.

Database storage and table names

You can configure the data source and table name used to manage messages for the queue or topic by setting the Data Source and Database Properties. See “Data source and database tuning” on page 103 for performance considerations.

The default table name for queues and topics is jms_pm, which results in all queues and topics using a single table. If a single table becomes a bottleneck when using multiple queues and topics, specify different tables for each queue and topic. The database server may be better able to handle the load using multiple tables. You can also use different data sources to run queues and topics from different database servers.

In memory message limits and overflow handling

Set the queue or topic Maximum Messages In Memory property to constrain memory used to manage messages. The value is the number of messages that can be held in memory at once.

Persistent messages are always stored in the database, and some will be held in memory to improve performance. This property affects the size of the in-memory cache if the queue or topic is used only for persistent messaging.

EAServer manages non-persistent messages primarily in memory. You can configure the Store Transient Messages property how transient message overflow is handled. A value of false causes EAServer to discard overflow messages. A value of true enables storage of overflow messages in the database.

Keeping more messages in memory generally improves performance, unless the memory used leaves too little for other server tasks. Tune the message limit and overflow handling properties to balance performance requirements and memory constraints against the possibility of lost messages. Persistent messages that are discarded from memory can be retrieved from the database. Transient messages are lost when discarded from the queue unless you enable the Store Transient Messages property.

You can also control the memory used by queues and topics by:

- Configuring the idle timeout setting
- Setting the time-to-live message property in your application code
- Minimizing the message size

Idle timeout setting

In queue or topic properties, the Idle Connection Timeout setting specifies number of seconds that the message queue remains in memory when it is not being accessed by a consumer and has no registered listener.

For transient messaging, the Store Transient Messages property determines the fate of transient messages when an idle timeout occurs. If this property is true, EAServer saves transient messages to the database. Otherwise an idle timeout causes EAServer to discard transient messages.

For persistent messaging, the idle timeout setting acts like a cache timeout. Messages likely to be accessed soon are kept in memory. An idle timeout flushes the cached messages from memory.

If your application uses temporary queues or topics, configure a JMS Factory and specify the queue and topic template properties. Set the timeout in the queue and topic that are used as templates.

Transient message storage

EAServer may store transient messages to the database in two scenarios:

- The in memory message limit for the queue or topic has been exceeded
- The queue sits idle for longer than the specified idle timeout, causing EAServer to flush messages from memory

The Store Transient Messages property determines whether messages are discarded or stored to database in these scenarios. If lost messages are acceptable, set this property to false for better performance.

Duplicate key detection

The Ignore Duplicate Messages specifies whether EAServer checks for messages with duplicate keys before sending them. If duplicate messages are acceptable in the application, disabling this setting can improve performance.

If duplicate key detection is enabled, the Duplicate Detection Protocol property determines how EAServer handles duplicate keys. A setting of `optimistic` will optimize the detection of duplicates by using deferred inserts and checking for duplicate key exceptions from the duplicate detection table. As a result, when duplicates are detected, transaction rollbacks will occur, and automatic transaction retry adds non-deferred duplicate checking. A setting of `pessimistic` will always use non-deferred duplicate checking. When there are few duplicates, `optimistic` is more efficient. When there are many duplicates, `pessimistic` may be more efficient.

To check for duplicate keys, EAServer logs key values to the database table specified by the Duplicate Detection Table property. To prevent this table from growing to an unreasonable size, set the Duplicate Detection Timeout property. EAServer deletes entries that remain in the table past this time limit.

Queue and topic data compression

For persistent messaging, you can also configure the Compressed Stored Messages queue or topic property to enable compression before persisting the message. You can also enable compression on the network connection between the JMS client and EAServer—see “For large message values, consider using compression” on page 103.

Compression may increase performance if the application uses large messages. Test system performance to determine whether enabling network and persistent message compression increases throughput. While compression can reduce database and network overhead, it increases CPU usage by EAServer and your client application.

Abbreviated queue and topic identifiers

If using persistent messaging and queues or topics with long names, set the Queue ID property to specify an abbreviated identifier. By default, EAServer stores the full queue or topic name with the persisted message data. Using an abbreviation can reduce database and network overhead.

JMS connection factory settings

Connection factory settings configure how JMS clients interact with the message service. For details, see “Connection factories” in Chapter 2, “Setting up the Message Service,” in the *Java Message Service User’s Guide*. The command batching options can be tuned to affect performance. For details, see the property descriptions.

To ensure that the idle timeout property is set for queues that are created programmatically by clients, set the topic and queue template properties to specify a preconfigured topic and queue that have appropriate timeout values.

Using store-and-forward messaging

You can configure EAServer to perform store-and-forward messaging by defining a message queue with the Pull Messages From property set to specify another message source, or the Push Messages To property set to specify another message consumer. When using store-and-forward messaging, tune the Push Batch Size or Pull Batch Size. The default of 1 causes EAServer to send messages individually. A larger batch size may improve performance if there is high message traffic.

Message driven bean tuning

A message driven bean (MDB) is a variation of the EJB stateless component model designed to run as a message consumer. For more information, see "Message-driven beans" in Chapter 2, "Setting up the Message Service," in the *Java Message Service User's Guide*.

If your MDB implementation can run concurrently on multiple threads, this configuration can significantly increase performance. EAServer by default delivers messages using a single worker thread. The default configuration guarantees first-in-first-out (FIFO) processing of messages in the queue, based on message priority: EAServer delivers messages serially to one component instance. If you do not require FIFO message ordering, customize the MDB thread settings to increase the thread count. EAServer creates several instances of the MDB, each running on a different thread to process messages concurrently.

In the Management Console, modify the MDB thread count by modifying the properties of the EJB Module that contains the MDB. For details, see Chapter 2, "Deploying and Configuring EJB Components," in the *EJB Users Guide*. In a user configuration file, you can configure the thread count using the Ant <setProperties> and <messageListener> commands, for example:

```
<setProperties component="ejb.components.myjar.MyListener">
    <messageListener queue="MyQueue" threadCount="5"/>
</setProperties>
```

Index