Performance and Tuning Guide

EAServer

5.0

DOCUMENT ID: DC20063-01-0500-01

LAST REVISED: December 2003

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

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 connection cache 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.

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 jagtool to configure applications rather than EAServer Manager
variable, *package*, or *component*	Italic font indicates: • Program variables, such as *myCounter* • Parts of input text that must be substituted, for example: *Server*.log • 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.”
`package 1`	Monospace font indicates: • Information that you enter in Jaguar Manager, a command line, or as program text • Example program fragments • Example output fragments

Related documents

Core EAServer documentation The core EAServer documents are available in HTML format in your EAServer software installation, and in PDF and DynaText format on the *Technical Library* CD.

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

The *EAServer Cookbook* contains tutorials and explains how to use the sample applications included with your EAServer software.

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 System Administration Guide* explains how to:

- Start the preconfigured Jaguar server and manage it with the EAServer Manager plug-in for Sybase Central™
- Create, configure, and start new application servers
- Define connection caches
- 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 jagtool

The *EAServer Programmer's Guide* explains how to:

- Create, deploy, and configure components and component-based applications
- Create, deploy, and configure Web applications, Java servlets, and JavaServer Pages
- Use the industry-standard CORBA and Java APIs supported by EAServer

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 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 using the Security Manager plug-in for Sybase Central
- 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 API Reference Manual* contains reference pages for proprietary EAServer Java classes, ActiveX interfaces, and C routines.

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://www.sybase.com/detail?id=1024509.

Message Bridge for Java™ Message Bridge for Java simplifies the parsing and formatting of structured documents in Java applications. Message Bridge allows you to define structures in XML or other formats, and generates Java classes to parse and build documents and messages that follow the format. The *Message Bridge for Java User's Guide* describes how to use the Message Bridge tools and runtime APIs. This document is included in PDF and DynaText format on your *EAServer 5.0 Technical Library* CD.

Adaptive Server Anywhere documents EAServer includes a limited-license version of Adaptive Server Anywhere for use in running the samples and tutorials included with EAServer. Adaptive Server Anywhere documents are available on the Sybase Web site at http://sybooks.sybase.com/aw.html.

jConnect for JDBC documents EAServer includes the jConnect™ for JDBC™ driver to allow JDBC access to Sybase database servers and gateways. The *Programmer's Reference jConnect for JDBC* is available on the Sybase Web site at http://sybooks.sybase.com/jc.html.

Other sources of information

Use the Sybase Getting Started CD, the Sybase Technical Library CD and the Technical Library 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 Technical Library CD. It is included with your software. To read or print documents on the Getting Started CD you need Adobe Acrobat Reader (downloadable at no charge from the Adobe Web site, using a link provided on the CD).
- The Technical Library CD contains product manuals and is included with your software. The DynaText reader (included on the Technical Library CD) allows you to access technical information about your product in an easy-to-use format.

  Refer to the *Technical Library Installation Guide* in your documentation package for instructions on installing and starting the Technical Library.

- The Technical Library Product Manuals Web site is an HTML version of the Technical Library CD that you can access using a standard Web browser. In addition to product manuals, you will find links to EBFs/Updates, Technical Documents, Case Management, Solved Cases, newsgroups, and the Sybase Developer Network.

  To access the Technical Library Product Manuals Web site, go to Product Manuals at http://www.sybase.com/support/manuals/.

Accessibility features

EAServer 5.0 has been tested for compliance with U.S. government Section 508 Accessibility requirements. The online help for this product is also provided in HTML, JavaHelp, and Eclipse help formats, which you can navigate using a screen reader.

EAServer Manager supports working without a mouse. For more information, see "Keyboard navigation" in Chapter 2, "Sybase Central Overview," in the *EAServer System Administration Guide*.

The WST 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 refer to 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 might 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.

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. Enter user name and password information, if prompted (for existing Web accounts) or create a new account (a free service).
  3. Select a product.
  4. Specify a time frame and click Go.
  5. Click the Info icon to display the EBF/Maintenance report, or click the product description to download the software.

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

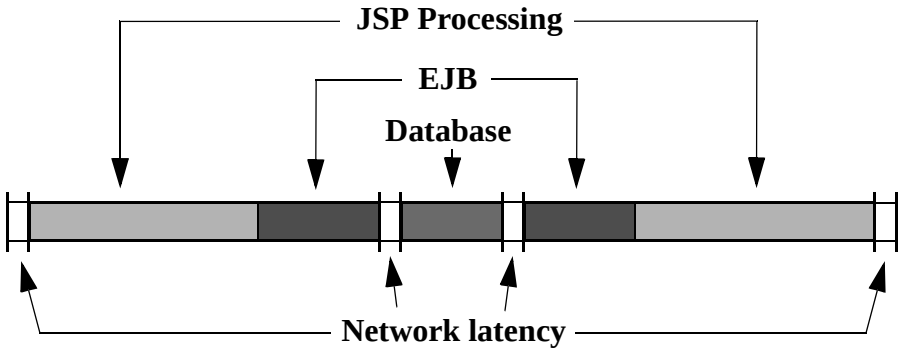
Several factors determine how well your application and server configuration perform.

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 (4GB on most 32-bit address systems).

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 connection cache (see “Clean up connections before releasing them to the cache” on page 105). Since EAServer pools and reuses component instances and connection caches, 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 26.
- **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 33. For database connections, use a connection cache. You can cache objects of other types within your component, servlet, or JSP class instance.

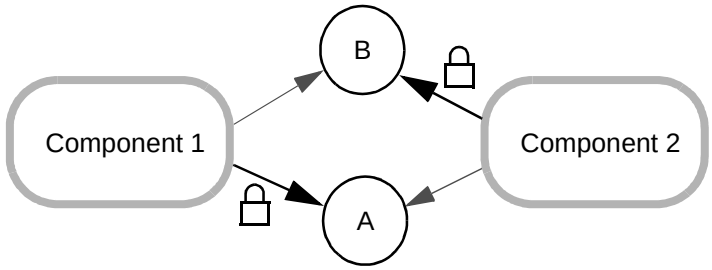
Threading

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 total number of EAServer threads, and partition the total to different tasks such as IIOP and HTTP request handling. 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, 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 30.

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

In 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. Both of these packages allow simple configuration of logging options and can be integrated with EAServer. For more information, see "Configuring log profiles" in the *EAServer System Administration Guide*.

Profiling software

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

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.

You can also turn on tracing for the Java garbage collector as described in "Trace-logging options" on page 9.

EAServer monitoring and tracing tools

EAServer includes these monitoring and tracing tools.

Runtime monitoring with EAServer Manager

EAServer Manager includes a runtime monitor that shows component, Web application, and connection cache statistics. For more information, see Chapter 11, "Runtime Monitoring," in the *EAServer System Administration Guide*.

In a clustered deployment, you can monitor the per-server load as described in "To view the current per-server load" in Chapter 7, "Load Balancing, Failover, and Component Availability," in the *EAServer 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.

- CtsComponents::CacheMonitor allows you to retrieve cache statistics for HTTP response caching, EJB CMP entity bean instance caching, and EJB CMP finder-results caching.

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

Trace-logging 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 32 for more information.
- For EJB CMP entity beans, you can configure the EAServer JDBC wrapper driver to record query execution statistics. See "CMP runtime monitoring" on page 75 for details.
- For in-server Java code, you can turn on tracing for the custom class loader. You can use the trace to identify classes that are loaded redundantly for different components and Web applications. To enable tracing, set the server property com.sybase.jaguar.server.classloader.debug to true. For information on configuring custom class lists, see Chapter 30, "Configuring Custom Java Class Lists," in the *EAServer Programmer's Guide*.
- For in-server Java code, you can turn on tracing for the Java garbage collector by setting the com.sybase.jaguar.server.jvm.verboseGC server property to true. The trace output describes how often the garbage collector runs, how long it takes, and what objects are deallocated. For information on the encoding of the output, see the documentation for the JDK version that you are running the server with.

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.

v **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 .1 second 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.

CHAPTER 2

Server Tuning

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

The performance tuning wizard

The performance tuning wizard walks you through the server settings that affect application performance. The wizard provides a convenient way to set all the settings described in this chapter. To start the wizard, highlight the icon for your server, then choose File | Performance Tuning Wizard. See the online help for more information on each setting.

General server tuning

These settings affect all applications.

Thread settings

The server threading properties affect the number of clients that can be served simultaneously and the memory used by each executing thread.

Number of threads

EAServer uses thread pooling rather than a thread-per-client model. Thread pooling allows more clients to connect than threads. Threads are created at server start-up, up to a specified maximum number. EAServer maintains separate thread pools for processing HTTP requests, IIOP requests, and internal processes that are not driven by client requests (services, garbage collection, and so forth). If the thread pools are too small, the server refuses client requests. If the size is too large, memory is wasted.

You can configure these properties to size thread pools:

Property	EAServer Manager location	Full name
HTTP threads	Server Properties/HTTP Config, Maximum Threads	com.sybase.jaguar.server.http.maxthreads
IIOP threads	Server Properties/Resources, Max Number Client Sessions	com.sybase.jaguar.server.maxconnections
Maximum threads	Server Properties/Resources, Max Number Threads	com.sybase.jaguar.server.maxthreads

Determining HTTP thread requirements

To determine how many HTTP threads are required, check the request pattern in the *httpstat.dat* file 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.

Determining IIOP thread requirements

The number of IIOP threads must be greater than or equal to the maximum number of IIOP clients that you expect. These clients include:

- Standalone Java, C++, PowerBuilder, or ActiveX clients
- EAServer Manager and other administrative or developer connections
- Web clients that run Java applets

Determining the required maximum threads setting

The rule for setting the maximum threads property is:

```
max threads = HTTP threads + IIOP threads + extra threads
```

The extra threads include those required to run internal processes that are not driven by client requests, including:

- The message service. The number of threads required can be configured in thread pool properties.
- Any other service components installed in the server. Allow one thread per service, or more if you configure a service to run in multiple threads.
- Components for which the Instances/Bind Thread option is enabled (that is, the com.sybase.jaguar.component.bind.thread property is set to true). Add one extra thread per component instance.
- The thread manager, if you use it in your application.
- The garbage collector.

Typically, 50 is a sufficient number of extra threads. You may need more if you increase the number or size of the thread pools used by the message service, you run additional service components, or if you use the thread manager. You may get by with less if you do not use these features.

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)

For information on setting the stack size, see "Configuring server stack size" in the *EAServer System Administration Guide*.

You can also configure the stack size for Java threads, as described in "JVM memory allocation parameters" on page 17.

Flow control

When the server is very busy with many client connections, client request threads may repeatedly conflict with each other for access to low-level system resources. Flow control provides a coarser level of granularity for synchronizing access to system resources by request threads. When enabled, flow control can improve performance by replacing multiple, serial choke points in the request processing sequence with a single choke point.

Flow control is enabled separately for HTTP and IIOP clients, by setting these properties on the Advanced tab in the Server Properties dialog box:

- com.sybase.jaguar.server.flowcontrol.http enables flow control for HTTP client threads. The default is false, which disables flow control. Set the value to true to enable flow control for HTTP client requests.
- com.sybase.jaguar.server.flowcontrol.iiop enables flow control for IIOP client threads. The default is false, which disables flow control. Set the value to true to enable flow control for IIOP client requests.

Warning! In some scenarios, IIOP client threads may deadlock when IIOP flow control is enabled. Deadlock can happen when using client demarcated transactions, in stateful component instances that lock shared resources, and other cases where locks may be held across subsequent client requests. Do not enable IIOP flow control in production servers without first load testing your applications with scenarios that reflect peak production loads.

- com.sybase.jaguar.server.flowcontrol.maxexethreads specifies the maximum number of threads that can concurrently execute code that is governed by flow control. The default is the value of the com.sybase.jaguar.server.maxthreads property (Resources/Maximum Threads in the Server Properties dialog box), which means all threads can proceed. Tune this number to get the best performance under peak stress conditions. Values between 15 and 30 are a good starting point. To tune the setting, monitor response time at peak load conditions, and raise or lower the value to find the setting that results in the best response time.

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. You can easily disable all settings with the tuning wizard described in “The performance tuning wizard” on page 11.

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 on the General tab in the EAServer Manager Server Properties dialog box.

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\user_setenv.bat* file (for Windows platforms) or *bin/user_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.

You can also configure the server class path by modifying the following server properties, using the advanced tab in the EAServer Manager server properties dialog box:

- com.sybase.jaguar.server.jvm.classpath to configure the CLASSPATH setting.
- com.sybase.jaguar.server.jvm.classpath.jars to specify JAR files in the *java/lib* directory to add to the CLASSPATH setting.
- com.sybase.jaguar.server.jvm.bootclasspath to configure the BOOTCLASSPATH setting.
- com.sybase.jaguar.server.jvm.bootclasspath.jars to specify JAR files in the *java/lib* directory to add to the BOOTCLASSPATH setting.

For syntax information, see the reference pages in Appendix B, “Repository Properties Reference,” in the *EAServer System Administration Guide*.

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, duplicate in the custom class list for different components can waste memory by loading duplicate class instances. Chapter 30, “Configuring Custom Java Class Lists,” in the *EAServer Programmer’s Guide* describes how to configure common class lists for components and Web applications.

Minimize Refresh in production servers
Refreshing components loads additional copies of all implementation classes. EAServer leaves the previous implementation in memory for use by existing client sessions. In effect, refresh introduces a controlled memory leak. For this reason, it is best to restart your production server after deploying a large number of new implementation classes.

Java VM type and version

EAServer supports several 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 from the command line, 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.

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. You should enable the JIT compiler for the server, unless advised not to in the *EAServer Release Bulletin* for your platform. You can set the property on the Java VM tab in the EAServer Manager server properties dialog box, or by using jagtool to set server property com.sybase.jaguar.server.jvm.nojit to `false`.

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. You can tune the memory used by setting these properties of the EAServer Manager Server Properties dialog box, or by using jagtool. To set these properties, use the syntax documented in their reference pages in Appendix B, "Repository Properties Reference," in the *EAServer System Administration Guide*:

- com.sybase.jaguar.server.jvm.maxHeapSize specifies the maximum heap size. 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.

- com.sybase.jaguar.server.jvm.minHeapSize specifies the minimum, or initial heap size. While the maximum size is reserved at start-up, only the minimum size is monitored and allocated from by the JVM. 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.
- com.sybase.jaguar.server.jvm.options configures additional start-up options for the Java VM. The -XssStackSize parameter can be tuned to configure the stack size for Java threads. StackSize is the amount of virtual memory reserved for the stack of each Java thread.
- Optionally set com.sybase.jaguar.server.jvm.verboseGC to true to enable trace output from the Java garbage collector. The trace output describes how often the garbage collector runs, how long it takes, and what objects are deallocated. For information on the encoding of the output, see the documentation for the JDK version that you are running the server with.

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, using the syntax described in the reference pages for these properties in the *EAServer System Administration Guide*, Appendix B, "Repository Properties Reference."

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 options set in the server property com.sybase.jaguar.server.jvm.options, set on the Advanced tab in the EAServer Manager Server Properties dialog box or using jagtool. Use the syntax described in the com.sybase.jaguar.server.jvm.options reference page in the *EAServer System Administration Guide*, Appendix B, "Repository Properties Reference."

To verify the Java VM options, set the server property com.sybase.jaguar.server.jvm.displayOptions to true, then restart the server. The server logs all the options at start-up, including those that are configured by dedicated properties such as 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 MASP or Open Server clients, remove these listeners. Unused listeners waste memory and network resources.

The following listener properties affect performance:

Property	EAServer Manager location	Full name
Keep alive HTTP only	Listener Properties/Advanced/ Keep alive	com.sybase.jaguar.listener.http.conn.keepalive
Maximum requests *HTTP only*	Listener Properties/Advanced/ Maximum Requests	com.sybase.jaguar.listener.http.conn.maxrequests
Request pool size *Solaris platform only*	Listener Properties/Advanced/ Request Pool Size	com.sybase.jaguar.listener.solaris.tli.maxoutcon

For listeners that use the SSL protocol, you can tune the SSL session caching parameters described in "SSL session caching" on page 21.

HTTP keep alive

This setting determines how long the server keeps idle HTTP connections open. The same browser client may send several requests at periodic intervals, for example, if the user is browsing linked documents or paging through consecutive forms. In these cases, there is a performance benefit if the connection remains open, since it takes time to reestablish a new connection. The default value is 100 seconds. To change the setting, enter a different time in seconds.

HTTP maximum requests

This setting specifies the maximum number of HTTP requests to service before closing each connection. The default is 100. Tuning the value provides a way to prevent clients from monopolizing HTTP threads and connections with a series of short-duration requests.

Connection request pool size

On Solaris, you can configure the size of the pool used to handle outstanding connection requests. 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.

Values must be a positive integer less than or equal to 4096. If this property is not set, the default is 128. Values greater than 4096 are truncated to 4096 to avoid excessive memory allocation at start-up.

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.

SSL session caching

For improved performance, EAServer caches SSL session identifiers and allows clients to reuse them. Since creating an SSL session requires CPU-intensive computations, SSL session reuse results in a relatively large performance gain over setting up completely new security sessions for each connection. The SSL settings for a listener are configured in the security profile set in the listener Security Profile property (if using jagtool, the property com.sybase.jaguar.listener.security). For details on creating security profiles, see "Configuring security profiles" in Chapter 12, "Security Configuration Tasks," in the *EAServer Security Administration and Programming Guide.*

The settings below, on the Advanced tab in the EAServer Manager Security Profile Properties dialog box, control how SSL clients can reuse sessions for subsequent and simultaneous connections.

Setting	Description
SSL Cache Size	The number of entries in SSL session cache. If using jagtool, set as security profile property com.sybase.jaguar.security.sesscachesize.
SSL Session Share	The number of concurrent connections that can simultaneously use the same session entry (ID) in the session cache. If using jagtool, set as security profile property com.sybase.jaguar.security.sessshare.
SSL Session Linger	The duration for which a session entry is kept in the SSL session cache after the last SSL session using this session ID was closed. If using jagtool, set as security profile property com.sybase.jaguar.security.cachetime.

Cached sessions allow the client to reuse a session in a subsequent connection. The SSL Cache Size setting controls how many entries can be cached. Set this to a number less than or equal to the maximum connections setting for the server. The default cache size for security profiles created in EAServer Manager is 30. The cache requires approximately 64 bytes per entry. The SSL Session Linger value specifies how long cached session IDs remain valid. The default is 8 hours.

The SSL Session Share setting specifies how many simultaneous connections can share one session ID. Session sharing can improve performance when the client opens multiple connections simultaneously. For example, a browser client may open several connections at once to download images linked to an HTML page. Session sharing allows the client to reuse the session for the second and subsequent connections, up to the number of concurrent connections specified by the SSL Session Share value. The default value is 10.

Note These are advanced SSL parameters. They should be set only by someone who is knowledgeable about SSL.

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. For example, to set the limit to 1024:

1 Use a text editor to open the *bin/serverstart.sh* file.

2 Below the first comment block, add this line:

```
ulimit -n 1024
```

See your UNIX documentation for more details on the ulimit command.

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 26
- 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:

- **Random number seeding** EAServer uses encryption algorithms that generate pseudorandom number generation. EAServer generates a random seed at start-up by collecting a variety of data from system sources. On some machines, the default seeding mechanism can cause start-up lag times. You can change the seeding mechanism as described in "Setting the JAGUAR_RANDOMSEED variable" in the *EAServer System Administration Guide*.
- **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. To control the number of messages read into memory, set the default.maximum property for the message service, as described in "Other global settings" on page 127.
- **Service components** All service components must return from their start methods before EAServer accepts client connections. Lengthy processing in the start method can delay server start-up. For more information, see Chapter 33, "Creating Service Components," in the *EAServer Programmer's Guide*.

- **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. You can reduce JSP compilation time by tuning the Web application settings discussed in "Runtime compilation settings" on page 82.
- **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). You can set a time limit for servlet initialization to complete by setting these properties:
  - For the server, the Timeout setting on the Servlets tab in the EAServer Manager Server Properties dialog box specifies an initialization timeout for servlets that are installed directly in the server. You can also set this property as com.sybase.jaguar.server.servlet.init-timeout using jagtool.
  - For the Web application, the Timeout setting on the General tab in the EAServer Manager Web Application Properties dialog box specifies an initialization timeout for servlets in the Web application. You can also set this property as com.sybase.jaguar.webapplication.init-timeout using jagtool.
  - For individual servlets, the com.sybase.jaguar.servlet.init.timeout property, set using jagtool or on the Advanced tab in the EAServer Manager Web Component or Servlet Properties dialog box.

  To specify a timeout, set the property to a positive integer, which specifies the number of seconds to wait. The default is 0, which indicates that there is no time limit.

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. You can set the following server and component properties to change this behavior:

- com.sybase.jaguar.server.destroyPooledInstancesOnShutdown for the server specifies whether to destroy pooled component instances when shutting down. A value of `true` indicates that pooled instances must be destroyed explicitly before shutting down. For example, an EJB component's ejbRemove method is called. This allows the pooled instance to clean up resources, such as closing database connections. A value of `false` specifies that instances are not destroyed. The default is `true`.

  If many component instances are pooled, explicit destruction of instances may lengthen the time required to shut down or restart the server.

  You can override the setting for individual components by setting the com.sybase.jaguar.component.destroyPooledInstancesOnShutdown component property.

- com.sybase.jaguar.server.destroyPooledInstancesOnShutdownTimeout for the server specifies how long to wait for each instance destruction method to return when destroying pooled instances during server shutdown. The value specifies the time to wait, in seconds. If no value is specified, the default is 5. You can override the setting for individual components by setting the component property com.sybase.jaguar.component.destroyPooledInstancesOnShutdownTimeout.

Set these properties on the Advanced tab in the Server Properties or Component Properties dialog boxes.

Servlet destruction

EAServer calls each servlet's destroy method before shutting down or after you have refreshed or stopped the servlet using EAServer Manager. If service calls are still active, 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. You can specify a finite timeout by setting these properties:

- For the server, the Destroy Timeout setting on the Servlets tab in the EAServer Manager Server Properties dialog box specifies a timeout for servlets that are installed directly in the server. You can also set this property as com.sybase.jaguar.server.servlet.destroy-wait-time using jagtool.

- For the Web application, the Destroy Timeout setting on the General tab in the EAServer Web Application Properties dialog box specifies a timeout for servlets in the Web application. You can also set this property as com.sybase.jaguar.webapplication.destroy-wait-time using jagtool.
- For individual servlets, the com.sybase.jaguar.servlet.destroy.wait-time property, set using jagtool or on the Advanced tab in the EAServer Manager Web Component Properties dialog box.

To specify a timeout, set the property to a positive integer, which specifies the number of seconds to wait. The default is 0, which specifies that EAServer calls destroy immediately.

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 17.
- **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. "Number of threads" on page 12 describes how to configure the number of server threads. "Thread stack size" on page 13 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 84.
- **Servlet buffer pools** EAServer pools the internal buffers that are required to build servlet responses. You can tune the number of buffers as described in "Servlet buffer pools" on page 79.
- **Entity bean instance and query caching** EAServer can cache instance data and finder-method results for EJB-CMP entity beans. See "Entity instance and query caching" on page 63 for more information.

- **Connection cache sizes** You can configure the number of connections stored in each cache as described in “Tuning the cache size” on page 108.
- **Component instance pool sizes** You can configure the pool size for component instances as described in “Instance pooling” on page 33. The memory required for each instance depends on your implementation.
- **Mirror cache sizes** In a clustered deployment, you can configure EAServer to replicate HTTP session data and stateful component data in memory. For more information, see:
  - “HTTP session replication mechanism” on page 121
  - “Component state replication” on page 123
- **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 30, “Configuring Custom Java Class Lists,” in the *EAServer Programmer’s Guide* describes how to configure common class lists for components and Web applications.
- **Use of the hot refresh feature** Refreshing components and Web applications loads additional copies of all implementation classes. EAServer leaves the previous implementation in memory for use by existing client sessions. In effect, refresh introduces a controlled memory leak. For this reason, it is best to restart your production server after deploying a large number of new implementation classes. 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 components, refresh the package that it is installed in rather than the whole server.

CHAPTER 3

Component Tuning

Running the performance tuning wizard

EAServer Manager includes wizards to tune the settings discussed in this chapter. To run the wizard, highlight the component icon and choose File | Performance Tuning Wizard.

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. An easy way to do this is to run the performance and tuning wizard on your components.

Thread-related issues

EAServer is scalable because it is multithreaded and multiprocessor-safe, using a thread pooling model to run components. 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

The Instances/Bind Thread setting in the EAServer Manager Component Properties dialog box specifies whether component instances must be bound to the thread that creates the instance. You can also set this property as com.sybase.jaguar.component.bind.thread using jagtool.

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 ActiveX components, PowerBuilder components if required (see “PowerBuilder component performance” on page 44), and components of other types that use thread-local storage. Otherwise, disable this feature so EAServer can run the component on any available thread.

Concurrency

The Instances/Concurrency setting in the EAServer Manager Component Properties dialog box specifies whether component instances can execute concurrently on multiple threads. You can also set this property as com.sybase.jaguar.component.thread.safe using jagtool.

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

If the Instances/Sharing option is enabled in the EAServer Manager Component Properties dialog box, a single instance serves all client requests. You can also set this property as com.sybase.jaguar.component.sharing using jagtool.

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. Such bottlenecks can reduce performance and may lead to larger problems such as deadlock or starvation. 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. For these reasons, Sybase recommends that you avoid 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.

Thread monitors are not active for component calls that use the lightweight container (see “Lightweight container” on page 40).

To create or configure a thread monitor, follow the instructions in Thread monitor properties in the *EAServer System Administration Guide*.

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

1 Display the properties for the component or method, then display the Advanced tab.

2 For a component, if the com.sybase.jaguar.component.monitor property is not present, add it. Otherwise modify this property. Set the value to the name of the thread monitor.

   For a method, if the com.sybase.jaguar.method.monitor property is not present, add it. Otherwise modify this property. Set the value to the name of the thread monitor.

3 Regenerate and recompile the component skeleton.

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. 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.

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 "Component lifecycles" in the *EAServer Programmer's Guide* for more information.

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.

If you use stateful components, configure the Instance Timeout setting on the Component properties: Resources tab in EAServer Manager or by setting the com.sybase.jaguar.component.timeout property with jagtool. This setting limits the time that an instance can be bound to a client session.

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 components by default. "Component lifecycles" in the *EAServer Programmer's Guide* describes how you can implement components of other types that support pooling.

Configuring component instance pool properties

To enable pooling, set the Pooling option on the Instances tab in the EAServer Manager Component Properties dialog box. You can set this property as com.sybase.jaguar.component.pooling if using jagtool.

You can configure the number of instances pooled for a component by configuring the properties in Table 3-2. These properties are found on the Resources tab in the EAServer Manager Component Properties dialog box.

Table 3-2: Component instance pooling properties

Property	Description
Maximum pooled instances	Specifies the maximum pool size. If the maximum pool size is reached, EAServer destroys excess instances after deactivation. The default is 0, which means no limit.
Minimum pooled instances	The minimum pool size. The default is 0.
Named instance pool	Specifies that the component shares space in a named instance pool, as described in “Named instance pools” on page 35.
Maximum active instances	Specifies the maximum number of instances that can exist at the same time. For a C++ component that runs as an external process, specifies the maximum number of simultaneously running external processes. If a request arrives when the maximum number of instances exist and are all busy, the request blocks, with blocking time constrained by the Maximum Wait setting.
Maximum wait	Specifies the maximum client wait time when the Maximum Active Instances property is set to specify a limit on the number of simultaneous active instances and the limit has been reached. The request blocks, with blocking time constrained by the Maximum Wait property. If the blocking time expires, the caller receives a CORBA::NO_RESOURCES exception.

While instance pooling can decrease client response time, it can also increase memory usage in the server. If pooled instances often sit idle, the memory used for pooling is wasted.

You can configure the maximum and minimum pool size to constrain the memory used to maintain an instance pool. For example, a heavily used component should have higher minimum and maximum pool sizes than a less commonly used component. If a component has periods of high use followed by low use, you can set a minimum size to allow the instances to be released during times of low use. Doing so frees memory for use by other components.

EAServer does not preallocate instances for the pool. The pool size grows as additional instances are required to satisfy client requests, up to the maximum specified size (if a maximum size is specified). Once the minimum pool size is reached, the size will not shrink below this size. To release idle pooled instances, EAServer has a garbage collector thread that runs periodically. Each time it runs, the garbage collector removes one idle instance from the pool, unless the minimum pool size has been reached.

If you configure a minimum pool size, configure a maximum size that is slightly larger. The difference between the maximum and minimum size provides a damping factor that prevents repeated instance allocation and deallocation if the actual pool size hovers near the minimum size.

If the maximum pool size is reached, EAServer still creates new instances, but destroys them immediately rather than placing them in the pool. You can set the Maximum Active Instances property to put an absolute limit on the number of instances that can run. Doing so allows you to partition the server resources among different components.

You can monitor the effectiveness of these settings by checking the Maximum Pooled Instances and Peak Maximum Instances Pooled setting for the package that contains your components. Ideally, the number of instances should rarely rise above the number of pooled instances. See "Using the Runtime Monitor" in the EAServer System Administration Guide for more information.

Named instance pools

Named instance pools provide more administrative control over how components are pooled. This feature allows you to control the number of component instances that EAServer creates for a set of components. Set the com.sybase.jaguar.component.instancePool property to assign your component to a pool other than the default. If this property is not set, the component runs in one of the default pools described in Table 3-3.

Table 3-3: Preconfigured instance pools

Name	Description	Maximum number of instances
SystemPool	The default for components whose Instances/Bind Thread property is disabled.	infinite, denoted by -1
BindThread	The default for components whose Instances/Bind Thread property is enabled.	512

By default, EAServer assigns components that do not have the Bind Thread setting enabled to SystemPool. This pool imposes no limit on the number of instances. For components placed in SystemPool, the component's properties control how many instances can be pooled.

The BindThread pool has a limited size that you can modify. As described in "Bind thread" on page 30, components that run with Bind Thread are resource intensive. Setting a limit on these instances prevents them from monopolizing the available threads.

You can create new pools with different instance pool settings to partition the memory used to run different components. For example, you can create a larger instance pool for frequently used components and create smaller pools for seldom used components.

In EAServer Manager, you can manage the instance pool settings in the top-level Instance Pools folder. Sybase recommends that you do not modify the SystemPool settings, or create new pools that have no size limit.

- **Creating an instance pool**

  1 Highlight the Instance Pools folder.

  2 Select File | New Instance Pool.

  3 In the New Instance Pool dialog box, enter a name for the instance pool, and click OK.

  4 In the Instance Pool Properties dialog box, on the General tab, enter a description and the maximum number of instances. By default, the maximum number of instances is set to 512. Sybase recommends that you do not create an instance pool with the maximum number of instances set to `-1` (infinite). If you need an instance pool with an unlimited number of instances, use the preconfigured SystemPool.

  5 To enable debugging, which writes instance pool information to the server log file:

    a Select the Advanced tab.

    b Click Add.

    c Enter these values:

    - Property – com.sybase.jaguar.instancepool.debug
    - Value – `true`

- **Viewing or modifying an instance pool**

  1 Expand the Instance Pools folder.

  2 Highlight the instance pool you want to modify and choose File | Properties.

Note You must refresh an instance pool or refresh the server before any changes to instance pool properties take effect.

---

v **Refreshing an instance pool**

1 Expand the Instance Pools folder, then highlight the instance pool you want to refresh.

2 Choose File | Refresh.

v **Assigning a component to an instance pool**

If you do not specify an instance pool for a component, EAServer assigns one of the default instance pools automatically using the following criteria. If a component's Instances/Bind Thread property is enabled, EAServer assigns the component to the BindThread instance pool; otherwise, it assigns the component to the SystemPool instance pool. If you enable the Instances/Bind Thread option for a component in the SystemPool, EAServer changes the instance pool selection so the component uses the BindThread instance pool.

Assign the component to an instance pool as follows:

1 In EAServer Manager, highlight the component to assign to an instance pool.

2 Select File | Properties.

3 On the Resources tab, select a Named Instance Pool from the drop-down list.

v **Deleting an instance pool**

1 Expand the Instance Pools folder.

2 Highlight the instance pool you want to modify and choose Edit | Delete.

Warning! Do not delete the preconfigured SystemPool or BindThread pools.

v **Managing instance pools using jagtool**

- You can also use these jagtool commands to create and configure instance pools: copy, create, delete, list, props, refresh, and set_props. For example, to create a new instance pool called "MyPool," use this syntax:

```
jagtool create InstancePool:MyPool
```

Optimizing intercomponent calls

If your components make many intercomponent calls, you can use one of the following techniques to improve performance:

- For EJB components, use local interfaces, the EAServer lightweight container, or call-by-reference. See “Optimizing in-server EJB calls” on page 39 for more information.
- For components of other types, use pseudocomponents. Since pseudocomponents are executed locally, in the same process, they do not incur the network overhead of client/server communication. When used in EAServer, pseudocomponents avoid the small thread- and context-management overhead incurred when the EAServer component dispatcher executes intercomponent calls. However, pseudocomponents are not suitable for applications that require the transaction control, security constraints, and other services provided by the EAServer component dispatcher. For more information, see Chapter 34, “Creating and Using EAServer Pseudocomponents,” in the EAServer Programmer’s Guide.

Java component performance

Java/CORBA and EJB components benefit from most of the settings described in “Common component performance issues” on page 29.

You can improve the performance of Java 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 30, “Configuring Custom Java Class Lists,” in the *EAServer Programmer’s Guide*.

Minimize Refresh in production servers
Refreshing components loads additional copies of all implementation classes. EAServer leaves the previous implementation in memory for use by existing client sessions. In effect, refresh introduces a controlled memory leak. For this reason, it is best to restart your production server after deploying a large number of new implementation classes.

EJB component performance

EJB components benefit from most of the settings described in "Common component performance issues" on page 29. You can also configure common class loaders as described in "Java component performance" on page 38. You can configure the settings below to further tune EJB components.

The techniques described below are specific to EJB components. 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.
- The EAServer Lightweight container offers further performance gain over local interfaces alone, and can be used with EJB 1.1 components and EJB 2.0 components that lack a local interface.
- You can also enable Pass-by-reference semantics in EAServer to achieve the same benefits as local interfaces when using EJB 1.1 components.

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 "Calling local interface methods" in the *EAServer Programmer's Guide*.

Lightweight container

EAServer provides a lightweight version of the standard EJB component container. The performance of some intercomponent calls in EAServer can be improved by enabling the lightweight container (LWC). The LWC coexists with the standard container. The LWC provides optimized performance for in-server EJB calls, while preserving all expected EJB semantics (such as pass by value when using remote interfaces). You can use the LWC for calls that use local or remote interfaces. The LWC can reduce CPU utilization for calls from session beans to entity beans, such that EJB-standard entity beans become as fast as lightweight persistence frameworks.

Comparing LWC to local interfaces

Both LWC and local interfaces improve performance by eliminating parameter marshalling, but the LWC provides even better performance by eliminating internal calls to enforce transactional and security requirements. You can also use LWC on EJB components that lack local interfaces, including EJB 1.1 components.

Determining if your component can use LWC

Because the LWC eliminates standard dispatcher code to enforce transaction semantics and security restrictions, you must verify that your components and the application's use of them satisfies these restrictions before enabling LWC.

The LWC is suitable for EJB components with transaction attribute set to Required, when such components are called by other EJB components with transaction attributes "Required" or "Requires New." EAServer verifies the following before using the lightweight container for EJB-to-EJB calls to ensure that EJB 1.1/2.0 semantics are fully preserved:

- The calling (source) component, servlet, or JSP uses an EJB reference or EJB local reference to make the call. No remote invocations are allowed.
- The LWC is compatible with the transaction and security properties of the calling (source) and called (target) components, meaning that the target inherits the same transaction context and calling identity from the caller. The calling component must be executing a transaction, and the target cannot have the run-as identity property set.

If the LWC is enabled for the called component, but the constraints are not satisfied, the call fails and EAServer logs an error.

When using the LWC, object references cannot be marshalled. For example, if component A calls B using the remote interface, and B returns a reference to component C, B will fail with exception java.io.NotSerializableException or ClassCastException if LWC is enabled for component C. To avoid this issue, disable LWC for the target components whose object references are passed as parameters or return values, or use local interfaces to call these components.

Enabling LWC

You must enable the lightweight container in server and component properties. To enable LWC for a server, select the LWC option on the Components tab in EAServer Manager's Server Properties dialog box. If using jagtool, set the server property com.sybase.jaguar.server.lwc to true.

To enable LWC for components, select the LWC option on the Instances tab in EAServer Manager's Component Properties dialog box. If using jagtool, set the component property com.sybase.jaguar.component.lwc to true.

By default, the LWC supports only EJB-to-EJB intercomponent calls. You can additionally enable calls from the Web tier (servlets and JSPs) by selecting the Enable Skeletons option in the Component Properties Instances tab and on the Server Properties Components tab. This option causes EAServer to generate a skeleton that allows non-LWC invocations from the Web tier. If using jagtool, enable this option by setting the server property com.sybase.jaguar.server.lwc.enableSkeletons to true and the component property com.sybase.jaguar.component.lwc.enableSkeletons to true.

You must regenerate stubs and skeletons for your components after changing any of these settings.

You can set the server property com.sybase.jaguar.server.lwc.debug to enable logging of additional information about LWC invocations.

Pass-by-reference semantics

EAServer supports the proprietary EJB pass-by-reference mechanism supported by some other J2EE vendors. To enable pass-by-reference for a component, set the property com.sybase.jaguar.component.passByReference to true. The default is false. When set to true, 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 regenerate stubs and skeletons.

This feature is not intended for new development, which should use standard EJB 2.0 local interfaces. When used, remote clients cannot call the component. The feature cannot be used with components that already have a local interface. If two or more components share the same home and remote interfaces, then all or none of those components must be configured for pass-by-reference.

Entity bean read-only methods

For entity beans that use bean-managed persistence, you can mark business methods that do not modify data. Doing so allows EAServer to skip calls to the ejbStore method after the business method returns. Performance improves through elimination of redundant database updates. Set this property using the Read Only option in the EAServer Method Properties dialog box. If using jagtool, set the method property com.sybase.jaguar.method.flags.

Read-only methods in CMP entity beans
When using EJB CMP entity beans, the persistence engine detects read-only method invocations. You do not need to set the Read Only method property. The ejbStore method is always called, but never performs data storage.

Entity bean database update frequency

For entity beans that use bean-managed persistence, you can configure when EAServer calls the ejbStore method by setting the com.sybase.jaguar.component.store property for the component, using jagtool or the Advanced tab in the EAServer Manager Component Properties dialog box.

This property specifies when ejbStore must be called. The default value of `afterCreate,afterInvoke` is required for EJB 2.0 compliance and is safe for all compliant entity bean implementations.

If you insert values in the ejbCreate method, and do not modify any field values in the ejbPostCreate method, you can safely remove `afterCreate` from the setting. Doing so improves performance by eliminating redundant updates to the database.

You can use the `beforeCompletion` setting rather than `afterInvoke` if all updates to one table come from one entity bean, and you do not mind if finder methods return stale values because updates are deferred during a transaction. While this setting yields the best performance, you may get the wrong result in architectures where more than one component can update a table, for example, if two entity beans in one transaction update the same table, or if updates from session and entity beans are mixed in one transaction.

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.
- **Set an instance timeout** Set the component Instance Timeout property as described in he Instance Timeout setting on the Component properties: Resources tab in EAServer Manager or by setting the com.sybase.jaguar.component.timeout property with jagtool. The timeout ensures removal or passivation of component instances when the client crashes or the end user walks away from their desk.
- **Optionally configure passivation** By default, EAServer destroys instances that time out. In single server deployments, you can optionally configure *passivation* so that EAServer saves the instance state data to a remote database before removing the instance from memory. Passivation allows EAServer to free memory used by idle instances, while still allowing clients to invoke the instance. If the client calls the instance again, EAServer restores the state data from the database. For more information, see “Using EJB activation and passivation” in Chapter 28, “Configuring Persistence for Stateful Session Components,” in the *EAServer Programmer’s Guide*.
- **Configure replication for clustered deployments** In a clustered deployment, you must configure the mechanism that EAServer uses to replicate state data between servers in the cluster. See “Component state replication” on page 123 for more information.

- **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. Set the Maximum Active Instances field on the Resources tab in the EAServer Manager Component Properties dialog box or by using jagtool to set the com.sybase.jaguar.component.objects property.

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 the EAServer Programmer's Guide.

If you cannot remove memory leaks or instabilities from legacy or third-party code called from your C++ components, you can run the component in its own process. While doing so does not improve the performance of the component, it can improve the overall server performance by isolating the effect of memory leaks or instabilities. For more information, see "Running C++ components externally" in the *EAServer Programmer's Guide*.

The suggestions in "Common component performance issues" on page 29 apply to C++ components.

PowerBuilder component performance

PowerBuilder version 8.0.3, in tandem with EAServer changes introduced in version 4.1.3, introduces many performance improvements such as improved memory management, reduced virtual address space, and less system resource use by DataStore objects. 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 3-4 to minimize resource use.

Table 3-4: 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.
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 you are using a Windows deployment with a PBVM version lower than 8.0.3 and you declare a DataStore as an instance or global variable. Bind Thread is not required in UNIX deployments, Windows deployments with PBVM 8.0.3 or later, or in earlier-version Windows deployments where you only use local variables for DataStore references. If the Bind Thread property is set to true when you deploy, be sure that the component is in the BindThread instance pool. This is the default pool for components with Bind Thread enabled unless you override the setting. For more information on threading issues that affect PowerBuilder components, see the *Application Techniques* manual in the PowerBuilder documentation. **Note** The Bind Thread property must be set to TRUE in the PowerBuilder IDE if you are using live editing to build your component. Remember to change it to FALSE when deploying your components for production use.

Setting	Description
Garbage collection	The PBVM uses a garbage collection model to free memory used by unreferenced and orphaned objects. A garbage collector thread runs periodically to perform this task. You can call the GarbageCollect() PowerScript® function to force garbage collection to occur immediately. Doing so may increase the performance of applications that use huge amounts of memory. For details on how to code components to force garbage collection, see the Sybase white paper Forcing the Garbage Collection Process in PowerBuilder 6.0/7.0/8.0 at http://www.sybase.com/detail/1,6904,1013157,00.html. 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.

Before reading this chapter, you should be familiar with the deployment settings described in Chapter 27, "Creating Entity Components," in the *EAServer Programmer's Guide*.

Generated entity bean subclasses

Beginning in EAServer 4.1.3, you can use the "Generated Class" option for EJB 1.1 and 2.0 CMP entity beans. This option offers better performance than the "Automatic Persistence" option, since the interaction between the storage component and the CMP implementation is more direct. If you import CMP entity beans from an EJB-JAR file, the "Generated Class" option is enabled by default. For existing CMP entity beans, you can configure it manually as described below.

To configure the "Generated Class" option in EAServer Manager:

1 Display the Persistence/General subtab in EAServer Manager.

2 Choose Generated Class for the Persistence option.

3 Optionally enter a class name for the generated subclass in the Generated Class field. If you do not specify a class name, the default is:

```
java-package._ps_package_component
```

Where *java-package* is the Java package of the implementation class, *package* is the EAServer package name, and *component* is the component name.

4 Configure the other Persistence tab options as described for Automatic Persistence in the *EAServer Programmer's Guide*. Click OK to save the properties.

5 Generate (or regenerate) skeletons for the component. EAServer generates the specified subclass.

To configure this option using jagtool or an EAServer XML configuration file, set these component properties:

- com.sybase.jaguar.component.ps to "generated"
- com.sybase.jaguar.component.ps.class to the class name

Creating and tuning database tables

While EAServer automatically creates entity bean tables for most 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 "Configuring table-mapping properties" in Chapter 27, "Creating Entity Components," in the *EAServer Programmer's Guide*.

Automatic key generation settings

EAServer supports several mechanisms for automatic key generation, described in "Enabling automatic key generation" in Chapter 27, "Creating Entity Components," in the *EAServer Programmer's Guide*.

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-use-rate setting described in "Configuring key generation to use a key lookup table" in the *EAServer Programmer's Guide*. To prevent different threads from creating duplicate keys, EAServer uses a semaphore to synchronize the key increment operation. Each thread reserves *key_use_rate* key values per increment. The key use rate can be tuned to reduce inter-thread contention for locks on the key table. The default of 100 results in good performance for most applications. Very large 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.

Concurrency control options

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.

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 as described below.

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 as described below, 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:

- Enable the Select With Lock option on the Persistence/General subtab. When using jagtool or XML configuration files, set com.sybase.jaguar.component.selectWithLock to true.
- Enable the Select for Update option by setting the com.sybase.jaguar.component.selectForUpdate property to true on the Advanced tab in the Component Properties dialog box or by using jagtool or an XML configuration file. This setting requests an exclusive database lock be obtained at select time to avoid deadlocks during lock promotion. 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 component property com.sybase.jaguar.component.touchColumn 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.
- Configure the table-mapping select queries and add "holdlock" or the appropriate lock syntax for your database. For more information, see "Configuring table-mapping properties" in Chapter 27, "Creating Entity Components," in the *EAServer Programmer's Guide*.

Also make sure that OCC is disabled by setting the Timestamp field to "none" on the Persistence/General subtab in the Component Properties dialog box.

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 CMP wrapper driver to allow verification queries to be batched with other deferred operations. See “Using CMP JDBC wrapper drivers” on page 60 for more information.

Configuring OCC options

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

To check that PCC is disabled, verify that the Select With Lock option on the Persistence/General subtab in the EAServer Manager Component Properties dialog box is disabled. On the Advanced tab, verify that com.sybase.jaguar.component.selectForUpdate is not set or set to false. When using jagtool or XML configuration files, verify the properties com.sybase.jaguar.component.selectWithLock and com.sybase.jaguar.component.selectForUpdate are both false.

Specify the timestamp mechanism in the Timestamp field on the Persistence/General subtab in the EAServer Manager Component Properties dialog box. Table 4-1 describes the allowable values. When using jagtool or XML configuration files, set the com.sybase.jaguar.component.timestamp property. If multiple tables are used and you specify a timestamp column, all tables must contain a column with the same name and datatype.

Table 4-1: Timestamp field values

To configure	Set the timestamp value to
A timestamp column	The name of a single column in each table that serves as the timestamp to detect update collisions. If the component uses multiple tables, each must contain a timestamp column with this name. The column type can be: • A 4-byte integer – this is the default timestamp column type. All processes that update the table(s) must increment the timestamp with each update, or your DBA can create an update trigger to increment the timestamp automatically. • 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. You must also define a field mapping property to specify the timestamp datatype as described in "Setting field-mapping properties" in the *EAServer Programmer's Guide*. For example, if the column name is ts, specify the mapping as: ts[dbts not null] dbts is a logical type name mapped to the timestamp type in the Sybase_ASE and Sybase_ASA database types. If the database does not support timestamps, a 4-byte integer counter is used instead.
A table level timestamp	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. When using a timestamp table, database triggers are required to increment the timestamp for each update, delete, or insert to tables that are mapped to the component or components that require the timestamp. You can set the component property com.sybase.jaguar.component.ts.triggers property so EAServer creates triggers, create triggers yourself, or add code to existing triggers.
All values comparison	Leave blank.
PCC	Set the value to "none" to disable optimistic concurrency control. In this case, you are strongly advised to configure locking as described in "Enabling PCC" on page 52.

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 For EJB CMP entity beans and entity components that use automatic persistence and optimistic concurrency control.

Auto-retry must be enabled for the component that initiates the transaction, which is typically a session bean in EJB applications. Auto-retry works only for intercomponent calls, not for direct invocations of entity beans from the Web tier or base clients.

Auto-retry can be configured in component and server properties as follows:

- In component properties, use the Advanced tab to set the property com.sybase.jaguar.component.tx_retry. A value of true enables auto-retry. A value of false disables auto-retry. If this property is not set, the value of the server property com.sybase.jaguar.server.tx_retry is used. If neither the component property or server property is set, the default is false.
- In server properties, use the Advanced tab to set the property com.sybase.jaguar.server.tx_retry. The default of false disables auto-retry for all components for which auto-retry is not explicitly enabled. Specify true to enable auto-retry for components for which auto-retry is not explicitly set to false.

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).

Configuring CMP isolation level

When using OCC, you can set the component property com.sybase.jaguar.component.cmp_iso_level to specify the effective transaction isolation level for CMP entity beans. This setting allows the performance benefits of OCC, while also enforcing an effective transaction isolation level as you would use with pessimistic concurrency control. Table 4-2 lists the allowable isolation levels.

Table 4-2: CMP isolation level values

Setting	Effect
`read_cache`	ejbLoad is satisfied by reading from the object cache if possible. Otherwise, data is loaded from the remote database. Not recommended, as use of this isolation level can result in "lost" updates. Instead, use `read_cache_verify_updates`.
`read_cache_verify_updates`	ejbLoad is satisfied by reading from the object cache if possible. Otherwise, data is loaded from the remote database. If the entity is changed or removed, the corresponding SQL update or delete verifies that the data was not changed after it was loaded from the DBMS. This setting is suitable when it is acceptable for a read-only transaction to use stale cache data. To limit the use of stale data, specify a cache timeout for the object cache or configure database change notification as described in "Enabling database change notification" on page 70.
`read_committed`	ejbLoad is satisfied by reading from the remote database. If the entity is changed or removed, the corresponding SQL update or delete does not verify that the data was not changed after it was loaded from the DBMS. Not recommended, as use of this isolation level can result in "lost" updates. Instead, use `read_committed_verify_updates`.
`read_committed_verify_updates`	ejbLoad is satisfied by reading from the remote database. If the entity is changed or removed, the corresponding SQL update or delete verifies that the data was not changed after it was loaded from the DBMS. This setting provides a good balance of data integrity and performance. However, for some application data models, the maintenance of full data integrity requires a higher isolation level such as `repeatable_read`. `read_committed_verify_updates_with_cache` may provide better performance.
`read_committed_with_cache`	ejbLoad is satisfied by reading from the object cache if possible. Otherwise, data is loaded from the remote database. If the entity is changed or removed, the corresponding SQL update or delete does *not* verify that the data was not changed after it was loaded from the DBMS. Otherwise, for read-only access, and only if ejbLoad was satisfied from cache, a commit-time verification ensures that the data has not changed since it was originally loaded from the DBMS. This ensures that any cached data that was used is still current at commit time, but does not prevent concurrent or conflicting updates. This setting is not recommended, as it can result in lost updates. Instead, use `read_committed_verify_updates_with_cache`.

Setting	Effect
`read_committed_verify_updates_with_cache`	ejbLoad is satisfied by reading from the object cache if possible. Otherwise, data is loaded from the remote database. If the entity is changed or removed, the corresponding SQL update or delete verifies that the data was not changed after it was loaded from the DBMS. Otherwise, for read-only access, and only if ejbLoad was satisfied from cache, a commit-time verification ensures that the data has not changed since it was originally loaded from the DBMS. This ensures that any cached data that was used is still current at commit time. This setting does not prevent concurrent updates but does prevent conflicting updates. This setting is suitable when it is not acceptable for a read-only transaction to use stale data, and where commit-time verification is cheaper than satisfying ejbLoad from the DBMS; in particular, where a table timestamp is used, or where a CMP wrapper driver is used (the CMP wrapper drivers can batch verification statements together at commit time).
`repeatable_read`	ejbLoad is satisfied by reading from the remote database. If the entity is changed or removed, the corresponding SQL update or delete will verify that the data was not changed after it was loaded from the DBMS. Otherwise, for read-only access, a commit-time verification ensures that the data has not changed since it was loaded from the DBMS. If pessimistic locking is enabled with the Select With Lock or Select For Update option, and is supported by the DBMS, verification is skipped as the shared/exclusive locks that are obtained at load time will prevent conflicting updates. **Warning!** Pessimistic locking may increase the occurrence of deadlock. This setting is suitable for cases where uncontrolled concurrent updates may result in data integrity problems (even for read-only access). `repeatable_read_with_cache` may provide better performance, although if many transactions are updating the same rows, pessimistic locking with no cache is probably preferable.

Setting	Effect
`repeatable_read_with_cache`	Uses the object cache: ejbLoad is satisfied by reading from the object cache if possible. Otherwise, data is loaded from the remote database. If the entity is changed or removed, the corresponding SQL update or delete verifies that the data was not changed after it was loaded from the DBMS. Otherwise, for read-only access, a commit-time verification ensures that the data has not changed since it was originally loaded from the DBMS. This setting is suitable for cases where uncontrolled concurrent updates may result in data integrity problems (even for read-only access), where it is not acceptable for a read-only transaction to use stale cache data, and where commit-time verification is cheaper than satisfying ejbLoad from the DBMS; in particular, where a table timestamp is used, or where a CMP wrapper driver is used (the CMP wrapper drivers can batch verification statements together at commit time). If there are many concurrent updates from EAServer transactions in the same server, you can configure soft-locking for the component to alleviate update contention—see “Using soft locking” on page 59. If many transactions from other sources are updating the same rows, you may get better performance using `repeatable_read` with pessimistic locking.

If the isolation level is not explicitly set, the default value depends on other property settings, as follows:

- If the component Timestamp property is set to “none”, the default is `read_committed`.
- If pessimistic concurrency control is enabled, the default is `repeatable_read`.
- Otherwise, the default is the value of the server property com.sybase.jaguar.server.cmp_iso_level, if set.
- Otherwise, the default is `read_committed_verify_updates`.

If object caching is enabled for the component (com.sybase.jaguar.component.objectCache is set), and the selected isolation level does not end with “_with_cache”, then EAServer uses the next higher isolation level that has the “_with_cache” suffix.

'serializable' isolation level
EAServer does not directly support the serializable isolation level. You can achieve this level using a table timestamp and the `repeatable_read` or `repeatable_read_with_cache` setting. However, the table timestamp mechanism is not suitable for tables that are frequently changed. You can also achieve this isolation level by using bean-managed transaction demarcation, and setting the isolation level for the JDBC connection before each transaction begins (you cannot change the isolation level during a transaction).

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.

You can enable soft locking by setting the following component properties:

- com.sybase.jaguar.component.softLock – a value of true enables soft locking. The default of false disables soft locking.
- com.sybase.jaguar.component.softLock.timeout – 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.

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 52.

Connection cache settings

The cache used by EJB CMP entity beans is specified on the Persistence/General subtab in the EAServer Manager component properties dialog box. Tune the cache settings described below.

Tuning the cache size and database type

Tune the cache size parameters as described in “Connection cache settings” on page 108. 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 Database type properties in Appendix B, “Repository Properties Reference,” in the *EAServer System Administration Guide*.

Using CMP JDBC wrapper drivers

EAServer includes customized JDBC drivers for use by CMP entity beans. 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. The Sybase wrapper driver also supports automatic creation of semi-temporary stored procedures to further improve performance.

Two wrapper drivers are supported:

- **Sybase** This driver is a wrapper around the Sybase jConnect driver. To use the driver, specify the class name com.sybase.ejb.cmp.SybaseDriver as the driver in your connection cache. Connection cache properties for this wrapper driver are the same as are used by com.sybase.jdbc2.jdbc.SybDriver, plus those listed in Table 4-3.
- **Oracle** This driver is a wrapper around the Oracle JDBC driver. To use the driver, specify the class name com.sybase.ejb.cmp.OracleDriver as the driver in your connection cache. Connection cache properties for this wrapper driver are the same as are used by oracle.jdbc.driver.OracleDriver, plus those listed in Table 4-3.

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 connection caches can be pinged sucessfully.

Table 4-3 lists the additional properties supported by the wrapper drivers. Tune these the settings in Table 4-3 in addition to the connection cache properties described in “Connection cache settings” on page 108.

Table 4-3: Sybase CMP JDBC wrapper driver properties

Property	Legal Values	Default Value	Description
CMP_DRIVER. DEBUG	true/false	false	Enables debug trace output which is written to the server log file.
CMP_DRIVER. DATABASE_TYPE	(entity name)	Sybase_ASE	Type of database (should be the same as the com.sybase.jaguar.conncache.db_type property for the cache). Use Sybase_ASA when connecting to Adaptive Server Anywhere, as there are some subtle differences between Adaptive Server Enterprise and Adaptive Server Anywhere in the handling of SQL batches.
CMP_DRIVER. MAXIMUM_ BATCH_PARAMETERS	0 or positive	99 (subject to change)	Maximum number of parameters in a batch.
CMP_DRIVER. MAXIMUM_BATCH_ STATEMENTS	0 or positive	3 (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.
CMP_DRIVER. PREPARE_CALL (Sybase only)	true/false	true	Set to true to enables the use of TDS-protocol RPC calls instead of language statements for more efficient communication with the database server and avoids repeated SQL statement parsing. When set to false, commands are sent as TDS-protocol language commands to execute stored procedures.

Property	Legal Values	Default Value	Description
CMP_DRIVER. PRINT_WARNINGS	true/false	true	Enables all database warning messages received by wrapper driver to be printed in server log.
CMP_DRIVER. STATEMENT_CACHE_SIZE (Oracle only)	0 or positive	9 (subject to change)	When using the Oracle wrapper driver, specifies the maximum number of cached prepared statements per connection. Setting this too high may result in too many JDBC prepared statements being cached, and the DBMS may run out of procedure cache or the server may run out of memory. Larger values will give better performance as long as memory is available.
CMP_DRIVER. TRACE_COMMIT	true/false	false	Trace transaction commit, rollback and autoCommit changes.
CMP_DRIVER. TRACE_CONNECT	true/false	false	Trace connect and reconnect operations.
CMP_DRIVER. TRACE_CREATE (Sybase only)	true/false	false	When using the Sybase driver, trace the creation of semi-permanent stored procedures.
CMP_DRIVER. TRACE_EXECUTE	true/false	false	Trace the execution of stored procedures and SQL command batches.
CMP_DRIVER. TRACE_EXECUTE_MS	0 or positive	0	Trace the execution of stored procedures and SQL command batches that take longer than the specified number of milliseconds. Specify 0 to disable. This setting takes precedence over CMP_DRIVER.TRACE_EXECUTE.

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.

Entity instance and query caching

EAServer supports object and query caching for EJB entity beans and entity components that use automatic persistence. 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.

Cache synchronization can be enabled to minimize the occurrences of transaction rollback due to overlapping updates. EAServer supports cache synchronization between servers in a cluster and from the database to EAServer. In some cases, the overhead of synchronization may outweigh the benefits incurred.

Cache architecture

Object and query caching place an in-memory cache and a cache manager component in between component instances and the associated database. You can configure the object cache and cache manager used by each entity component. You can configure the query cache used by each finder and ejbSelect method. You can configure caches that are dedicated to a single component or query method or shared by multiple components and query methods.

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 “Configuring object caching” on page 65. Query caching must be configured for each finder and ejbSelect method, on the Persistence/Query Mapping subtab in the Component Properties dialog box. See “Enabling query caching” on page 68. 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 `read_committed_verify_updates` or `read_committed_with_cache_verify_updates`. See “Configuring CMP isolation level” on page 55. 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)** EAServer uses OCC verification queries when you specify an isolation level that includes `verify` in the name. At commit time, the verification query checks whether the data has changed since it was originally selected. You should not disable OCC when using object caching, and you should use a timestamp column rather than using the default value-comparison technique of concurrency control. For details on configuring the timestamp column, see "Configuring OCC options" on page 53.

Read consistency using timeouts and synchronization For applications that have a more lax requirement for read consistency, you can configure cache timeouts and synchronization 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. You can also configure your database to notify the cache manager of updates, inserts, and deletes. Doing so allows EAServer to refresh the cache contents after data is modified by another application. See "Enabling database change notification" on page 70 for more information. The same notification technique is used for both object caching and query caching.

In addition, if the component is deployed in a cluster, you can configure inter-server synchronization, which uses the EAServer message service to replicate data changes between servers in the cluster. This ensures that all caches have the same data. To use this option, configure the Cache Synchronization property described in "Configuring object caching" on page 65.

Configuring object caching

For each entity component that uses automatic persistence, enable object caching on the Persistence/Object Cache subtab in the Component Properties dialog box. The settings are:

- **Enable Object Cache** Enables object caching for the component. To enable caching, you must also specify the name of the cache manager component. To use the built-in implementation, enter:

  ```
  CtsComponents/ObjectCache
  ```

  The object cache is enabled implicitly if you configure an isolation level that requires the use of the cache. See "Configuring CMP isolation level" on page 55.

- **Cache Name** Specifies the name of a cache shared by multiple components. Named caches must be configured as described in “Creating a named cache” on page 68.
- **Cache Size** If a named cache is not used, specifies the size of a cache that is dedicated to this component. If you specify both a Cache Name and a Cache Size, the component uses the shared cache specified by the Cache Name property, with the size specified in the named cache property file.

  Specify the size in megabytes, kilobytes, or bytes with the syntax shown in the following table:

Syntax	To indicate
*n*M or *n*m	*n* megabytes, for example: 512M
*n*K or *n*k	*n* kilobytes, for example: 1024K
n	*n* bytes, for example: 536870912

- **Cache Size Check Interval** When writing to the object cache, EAServer checks the size of the entry to see if the cache size would be exceeded. If so, least recently used entries are flushed from the cache until there is room for the new entry. The time spent calculating size can adversely affect performance. If you specify a size check interval *N*, EAServer performs the size calculation on only every *N*th entry, and uses a running average size for the interim entries. Setting a size check interval can improve performance. However, if the size of the data varies a lot, setting a size check interval may lead to inaccurate cache size estimations, resulting in memory use beyond the configured cache size. To configure this setting, set the com.sybase.jaguar.component.objectCache.sizeCheckInterval on the Advanced tab in the Component Properties dialog box. The default is 1, which means the size of each entry is checked.
- **Cache Timeout** Specifies how long cache entries remain valid. Components that use a named cache inherit the default timeout from the named cache configuration, but you can override this default. Components that use a dedicated cache have a default timeout of infinity. Specify the timeout in seconds. A value of 0 indicates infinity.

- **Cache Synchronization** For components deployed in a cluster, specifies whether caches on different servers in the cluster are synchronized. Cache synchronization is not necessary if you have configured an isolation level; in that case, the OCC verification queries prevent the use of stale data in transactions. The isolation level setting is easier to configure and provides a higher guarantee of data consistency.

  Table 4-4 describes the synchronization options. Components that use a named cache inherit the named cache's `sync` property; the inherited setting can be overridden by setting the component property.

Table 4-4: Cache synchronization options

Option	Explanation
None	(The default.) Indicates no synchronization is performed. This value is appropriate if the component is not deployed in a cluster, you have configured a component isolation level, or the cache timeout provides adequate read consistency for transactions.
Mirror	Replication without transactional consistency. This value is appropriate for entity components that maintain transient data (that is, the data is not saved to persistent database). If you use this option, you must configure mirror pairs for your cluster as described in "Cluster configuration for in-memory failover" in the *EAServer Programmer's Guide*.
Replicate	Replication with transactional consistency. For updates, the complete instance state is replicated between servers.
Invalidate	Replication with transactional consistency. For updates, the instance state is not replicated. Rather, updates are propagated by refreshing the cache entry from the remote database. This value may yield better performance than the `replicate` option if: • Your component's state is large, and • The cluster has many members. In this case, the overhead of replicating instance state may exceed that of refreshing each cache from the database.

Cache synchronization requires a working message service
If using object caching in a cluster, make sure the EAServer message service is configured and running on each server. The cache manager uses the message service for cache synchronization between servers. Chapter 8, "Setting up the Message Service," in the *EAServer System Administration Guide* describes how to run the message service.

- **Creating a named cache**

  If you want a cache to be shared by multiple components, finder methods or ejbSelect methods, you must create a named cache as follows:

  1. If the *Repository/ObjectCache* directory does not exist under your EAServer installation, create it.
  2. Create a text file in the *Repository/ObjectCache* directory named *Cache.props*, where *Cache* is the cache name used in component properties.
  3. Add lines as shown below to configure the cache properties. All properties are optional. If not set, the default values apply:

```
com.sybase.jaguar.objectcache.size=size-value
com.sybase.jaguar.objectcache.timeout=timeout-value
com.sybase.jaguar.objectcache.sync=sync-method
```

  These values correspond to the component object caching properties, as described in the table below. Each cache property uses the same value syntax as the corresponding component property:

Named cache property	Component property
`com.sybase.jaguar.objectcache.size`	Cache Size. If not specified, the default is unlimited.
`com.sybase.jaguar.objectcache.timeout`	Cache Timeout. If not specified, the default is infinity.
`com.sybase.jaguar.objectcache.sync`	Cache Synchronization

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, append `[cache]` to the end of the Query Mapping property value that corresponds to the method. For example:

```
[default][cache]
```

Or, for a query mapped to an EJB-QL query:

ejbQuery:[cache]

You can specify optional parameters with this syntax:

[cache *cache-params*]

Where *cache-params* is a list of parameters listed in Table 4-5, with each parameter separated from the next by white space, for example:

[default][cache size=1M timeout=10]

Table 4-5: Query cache configuration parameters

Parameter	To indicate
`name=`*name*	The cache name. Specifying a named cache allows multiple queries to use one cache. The named cache must be created and configured as described for named object caches in "Configuring object caching" on page 65. Only one of name or size may be specified.
`size=`*size*	The cache size. Only one of name or size may be specified. The value syntax is: • ***n*`M` or ***n*`m` to specify a size in Megabytes, for example: `1M` • ***n*`K` or ***n*`k` to specify a size in kilobytes, for example: `512k` • ***n*** to specify a size in bytes, for example: `1048576`
`timeout=`*seconds*	The cache timeout in seconds. A value of 0 indicates infinity.
`verify`	Specifies that finder results must be verified at the end of the transaction. Restrictions apply—see "Verifying cached finder method results" on page 70 for more information.
`ignore insert`	If database change notification is enabled, inserts do not invalidate the cache.
`ignore delete`	If database change notification is enabled, deletes do not invalidate the cache.
`ignore update`	If database change notification is enabled, updates do not invalidate the cache.

Verifying cached finder method results To obtain verified finder method semantics, include the keyword `verify` in the cache settings for the finder query. For example, use `[cache verify]` in place of `[cache]`. For components using object caching, this setting specifies that any finder method data used from cache should be verified at commit time with an appropriate database query. The verification query runs with an isolation level that is equivalent to higher than the component isolation level. Query cache verification requires the use of a table-level timestamp, and all tables referenced in the SQL query must use the same table timestamp as the entity bean for which the finder method is defined.

Configuring transaction local cache settings

EAServer uses the transaction local cache to minimize the number of database reads required when finder methods are called in a transaction. For example, if a finder method returns 100 rows, the worst case requires 101 queries to retrieve the data for each row. The transaction local cache helps achieve the ideal of selecting all required data at the beginning of the transaction.

The transaction local cache is enabled automatically if finder queries return enough data to populate the cache. In other words, the finder query returns all rows in the table. The default query properties do this. If you have modified them, verify that they return all rows. For more information, see "Specifying finder- and ejbSelect-method queries" in Chapter 27, "Creating Entity Components," in the *EAServer Programmer's Guide*.

You can set the component property com.sybase.jaguar.component.tlc.sort to specify whether EAServer sorts entries before calling ejbStore. Setting this property to true helps to avoid deadlock when separate transactions concurrently update multiple instances of the same component. The default of false may provide better performance by eliminating the sorting step. You cannot enable sorting unless the primary key class implements the java.lang.Comparable interface. Most java.lang utility classes implement this interface, such as String, Integer and so forth.

Enabling database change notification

This feature allows the use of database triggers to notify EAServer's entity object cache of changes to the underlying table rows. The notification mechanism works as follows:

1. Database triggers call a stored procedure sp_publish to publish a message for each SQL insert, update or delete.
2. sp_publish "publishes" the messages by placing them in a table cms_notify.
3. A cluster-wide singleton service, *CtsComponents/DatabaseNotify*, pulls notification messages from the cms_notify table using stored procedure sp_notify. These messages are then published to the EAServer message service. The expected latency for message delivery (from trigger to cache entry removal) is approximately one second at most.
4. The storage component (when using *CtsComponents/JdbcStorage*) listens for messages on selected topics, parses the messages for key fields, and notifies the Object Cache to remove the relevant entries.

- **Enabling database change notification**

1. Install the required stored procedures in the target database(s). See "Sample script for database stored procedures" on page 72.
2. In the EAServer Manager properties for your server, use the Advanced tab to configure the property `com.sybase.jaguar.server.services` to include the Message Service and Database Notify components, for example:

```
CtsComponents/MessageService,CtsComponents/DatabaseNotify
```

   If you have never run the message service in your installation, configure the message service as described in Chapter 8, "Setting up the Message Service," in the *EAServer System Administration Guide*. Database change notification requires a working message service.

3. Optionally add an entry to *MessageServiceConfig.props* to specify the name(s) of connection caches for databases which need to be monitored for notification messages. These connection caches must have type JDBC, for example:

   ```
   dn.caches=SybaseCache,OracleCache
   ```

   By default, the cache referenced by the `cms.cache` property will be used.

4. Optionally add an entry to *MessageServiceConfig.props* to specify the JDBC callable statement (or prepared statement) to be used to pull change notification messages from the database, for example:

   ```
   sp_notify={call my_own_notify_proc ?,?}
   ```

   By default, the callable statement is:

   ```
   {call sp_notify ?,?}
   ```

5 For each entity component that is to be configured for database notification, enable the Create Database Triggers option on the Persistence/General subtab in the Component Properties dialog box. This option requests automatic creation of triggers.

6 Optionally change the message service topic names associated with database tables. The default topic name is the unqualified table name. You must change the topic name if multiple databases contain tables with the same name. To change the topic name associated with a table, set the table mapping property for the table's notify operation, as described on "Configuring table-mapping properties" in the *EAServer Programmer's Guide*.

Sample script for database stored procedures

For Oracle databases, a sample script is provided in the the file *DatabaseNotify_Oracle.sql* in the *Repository/Component/CtsComponents* subdirectory of your EAServer installation. The sample script below is for Sybase Adaptive Server Enterprise. Modifications are required for use on other databases:

```
use master
go

if not exists (select name from sysdatabases where name = "notifydb")
  begin
    create database notifydb
    exec sp_dboption notifydb, "trunc log on chkpt", "true"
  end
go

use notifydb
go

checkpoint
go

if not exists (select 1 from sysobjects where name="cms_notify" and type="U")
  begin
    create table cms_notify
    (
      id numeric(16,0) identity primary key,
      type char(1) not null,
      name varchar(100) not null,
      message varchar(255) not null,
```

```
      options varchar(255) not null
    )
  end
go

if not exists (select 1 from sysusers where name="guest")
  exec sp_adduser guest
go

use sybsystemprocs
go

if exists (select 1 from sysobjects where name="sp_notify" and type="P")
  drop proc sp_notify
go

create proc sp_notify
  (@from numeric(16,0),
  @last numeric(16,0))
as
  if @from <= @last
    delete from notifydb..cms_notify where id >= @from and id <= @last
  declare @loop int
  select @loop = 1
  while @loop <= 60
    begin
      declare @rows int
      select @rows = count(*) from notifydb..cms_notify
      if @rows > 0
        begin
          set rowcount 100
          select id, type, name, message, options
            from notifydb..cms_notify
            order by id
          return
        end
      waitfor delay "00:00:01"
      select @loop = @loop + 1
    end
go

sp_procxmode sp_notify, anymode
go

grant execute on sp_notify to public
go
```

```
if exists (select 1 from sysobjects where name="sp_publish" and type="P")
  drop proc sp_publish
go

create proc sp_publish
  (@topic varchar(255),
  @message varchar(255),
  @options varchar(255))
as
  insert into notifydb..cms_notify (type, name, message, options)
    values ("T", @topic, @message, @options)
go

sp_procxmode sp_publish, anymode
go

grant execute on sp_publish to public
go

if exists (select 1 from sysobjects where name="sp_send" and type="P")
  drop proc sp_send
go

create proc sp_send
  (@topic varchar(255),
  @message varchar(255),
  @options varchar(255))
as
  insert into notifydb..cms_notify (type, name, message, options)
    values ("Q", @topic, @message, @options)
go

sp_procxmode sp_send, anymode
go

grant execute on sp_send to public
go
```

Customizing the implementation

The storage component responds to any suitably formatted messages that are published to the configured topic names for each mapped table. You can provide you own implementation of the stored procedures or the notification component.

To publish a change message, the Message Service 'text' property should be “insert”, “delete” or “update”, each key column should have a corresponding property (unless multiple rows were affected in which case key columns should be omitted). If using the Java Message Service (JMS) to publish the messages, use a TextMessage and use header properties for the key column values.

CMP runtime monitoring

You can enable logging of statistics from the CMP engine by setting the com.sybase.jaguar.conncache.cmp_stats property for the connection cache that your component uses. Set the value to a time interval in seconds; the CMP engine logs statistics at the specified interval. Statistics output includes table statistics, cache usage statistics, and query statistics. This data can be useful for tuning other connection cache and component properties for best performance.

You can also configure the CMP wrapper driver trace properties described in “Using CMP JDBC wrapper drivers” on page 60.

CHAPTER 5

Web Application Tuning

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

Using the performance tuning wizard

EAServer Manager includes wizards to tune the performance-related settings of Web applications and Web components. These wizards walk you through the configuration of many of the settings discussed in this chapter. To run the wizards:

- For a Web component, highlight the Web component icon and choose File | Performance Tuning Wizard.
- For a Web application, highlight the Web application icon and choose File | Performance Tuning Wizard.

See the online help for each wizard page if you need more information.

Tuning server and Web application settings

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

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:

- **Servlet engine tracing** The server property com.sybase.jaguar.server.servlet.trace enables tracing in the servlet engine. You can set this property in the Log/Trace tab of the Server Properties dialog box, or on the Advanced tab.
- **Web application security tracing** The Web application property com.sybase.jaguar.webapplication.sectrace enables tracing in the security subsystem of the EAServer Web application container. Set this property on the Advanced tab in the Web application properties dialog box.
- **Response cache debugging** The server property com.sybase.jaguar.server.http.cache.debug enables tracing in the static page caching engine. You can set this property on the Static Page Caching tab in the Server Properties dialog box, using the Enable Server Log Debug Messages check box. You can also use the Advanced tab.

An easy way to verify that all tracing properties are disabled is to run the server and Web application performance tuning wizards.

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 in the EAServer Manager Web Application Properties dialog box, or by setting the com.sybase.jaguar.webapplication.session-config property.

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 30, "Configuring Custom Java Class Lists," in the *EAServer Programmer's Guide*.

Minimize Refresh in production servers
Refreshing components loads additional copies of all implementation classes. EAServer leaves the previous implementation in memory for use by existing client sessions. In effect, refresh introduces a controlled memory leak. For this reason, it is best to restart your production server after deploying a large number of new implementation classes.

Servlet buffer pools

Internally, EAServer uses 4K and 8K temporary buffers when assembling servlet responses. These buffers are pooled and reused to avoid the overhead of repeated buffer allocation and garbage collection. To tune the number of buffers pooled, you can set the server properties below, using the Advanced tab in the Servlet properties window:

- com.sybase.jaguar.server.servlet.max8kbuffers specifies the number of internal 4K servlet response buffers.
- com.sybase.jaguar.server.servlet.max4kbuffers specifies the number of internal 4K servlet response buffers.

The default for both is 128. You can override the default by setting the value to a positive integer. A value of 0 means buffers are never pooled

The required buffers are allocated on an as-needed basis, rather than being preallocated as server start-up. Once allocated, buffers are pooled and reused until the specified size is reached. If a peak in client activity requires more buffers than the pool size, additional buffers are allocated, then released for garbage collection after use.

The default configuration suffices for most applications. If the buffer pool size is too small, performance.may decline due to allocation of new buffers. Allocation is costly because the Java VM initializes the allocated byte arrays to 0, which is not required by EAServer. Garbage collection is also costly. On the other hand, if the buffer size is too large, buffers allocated during periods of peak activity may be rarely used while consuming memory that would otherwise be available for other tasks.

For request processing, EAServer uses 8K buffers by default, and uses 4K buffers only when a servlet calls ServletResponse.setBufferSize() to request a buffer size other than 8K. If your application never or seldom changes the buffer size, you can set com.sybase.jaguar.server.servlet.max4kbuffers to 0 so that 4K buffers are not pooled.

To determine whether the settings are correct, examine the servlet request patterns to see if the number of concurrent requests often exceeds the buffer pool sizes. If so, consider increasing the value.

Clustered deployments

If you deploy your Web application in a cluster, tune the settings described in “Web application settings” on page 120.

HTTP and HTTPS listener configuration

The HTTP listener parameters can affect the performance of your application. “Listener tuning” on page 19 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 Programmer's Guide*, from Chapter 8, "Creating Enterprise JavaBeans 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 Chapter 22, "Creating Java Servlets," 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.

Set the Load During start-up option on the General tab in the Web Component Properties dialog box, or set the com.sybase.jaguar.servlet.load-on-startup property for the servlet or JSP in the Web application deployment properties.

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 81. 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 12, "Using jagtool and jagant," in the *EAServer System Administration Guide*.

JSP file timestamp checking

If you have precompiled all JSPs, you can turn off JSP file timestamp checking for the Web application. Doing so can increase performance by eliminating the required system calls. The com.sybase.jaguar.webapplicaton.jspc-interval Web application property controls how often EAServer checks to see if the JSP should be recompiled. The default is 0, which means the server must check the file timestamp every time the JSP is invoked. To disable checking, set the property to -1. To check after every *n* seconds, set the value to a positive integer *n*. For more information, see the description of this property in Appendix B, "Repository Properties Reference," in the *EAServer System Administration Guide*.

Runtime compilation settings

If your application design requires runtime JSP compilation, you can configure the settings described here to decrease the time required to compile. These settings can also reduce server start-up time if you have configured JSPs to compile at start-up.

To reduce the compilation time for JavaServer Pages (JSPs), you can configure the class path for JSPs separately from the EAServer class path. By default, the JSP compiler class path includes the EAServer process class path plus these class entries and JAR files that are specific to the Web application:

- The *work/servername/servlet/WebApp-webapp* directory, where EAServer generated servlet classes for JSPs. Here, *servername* is the name of the server, and *webapp* is the name of the Web application.
- The *Repository/WebApplication/webapp/WEB-INF/classes* directory, where you deploy classes used by servlets and JSPs in the Web application. Here, *webapp* is the Web application name.
- The JAR files in the *Repository/WebApplication/webapp/WEB-INF/lib* directory. Here, *webapp* is the Web application name.
- The JAR files in the *extensions* directory of your EAServer installation.

For the fastest JSP compile times, you must configure the compiler class path to eliminate unnecessary entries. To do so, set these properties on the Advanced tab in the Web Application Properties and Web Component Properties dialog boxes:

- com.sybase.jaguar.webapplication.jsp.compile-use-eas-cp

  For the Web application, set this property to false to exclude entries from the EAServer process CLASSPATH setting in the JSP compiler class path. The default of true indicates that the server class path should be included.

- com.sybase.jaguar.servlet.jsp.compile-use-eas-cp

  For individual JSPs (Web components in EAServer Manager), set this property to false to exclude entries from the EAServer process CLASSPATH setting in the JSP compiler class path. The default of true indicates that the server class path should be used.

- com.sybase.jaguar.webapplication.jsp.compile-use-third-party

  For the Web application, set this property to specify whether JAR files in the EAServer *java/lib* subdirectory are included in the JSP compiler class path. If this property is true, all JAR files in this directory are included. The default is false. This property is ignored if the com.sybase.jaguar.webapplication.jsp.compile-use-eas-cp property is set to true.

- com.sybase.jaguar.servlet.jsp.compile-use-third-party

For individual JSPs (Web components), set this property to specify whether JAR files in the EAServer *java/lib* subdirectory are included in the JSP compiler class path. If this property is true, all JAR files in this directory are included. The default is false. This property is ignored if the com.sybase.jaguar.webapplication.jsp.compile-use-eas-cp property is set to true for the Web application or the com.sybase.jaguar.servlet.jsp.compile-use-eas-cp is set to true for the Web component.

- com.sybase.jaguar.webapplication.jsp.compile-extra-cp

  For the Web application, specifies additional JAR files and directories to include in the JSP compiler class path. Specify the paths in a comma-separated list, with paths relative to the EAServer installation directory. For example, to include *$JAGUAR/java/lib/iaws.jar* and $*JAGUAR/java/classes/extra.jar*, set the property to:

  ```
  java/lib/iaws.jar,java/classes/extra.jar
  ```

- com.sybase.jaguar.servlet.jsp.compile-extra-cp

  For individual JSPs, specifies additional JAR files and directories to include in the JSP compiler class path, using the same syntax as the com.sybase.jaguar.webapplication.jsp.compile-extra-cp property.

Understanding HTTP response caching options

EAServer supports caching of static content and servlet responses.

Static page caching

Caching of static content improves performance by eliminating file I/O. You configure the static page cache at the server level, as described in "Static Page Caching" in the *EAServer System Administration Guide*. Follow these guidelines to get the best performance from your cache:

- **Exclude seldom-used files** Review your server's request logs and identify files that are rarely requested. These files should be excluded from cache, so that when requested, they do not cause more frequently requested content to be removed from the cache.

- **Tune the timeout value** The cache timeout value determines how long an entry can remain in cache before EAServer checks for updated content on disk. If you post a new version of a cached file, clients receive the old version from cache until the timeout expires or you flush the cache. Configure the timeout to allow content updates while minimizing the occurrence of cache misses. You can also configure the server to never timeout (using a very large value), and use the following strategy to post updated content:

  a Post updated content at regular intervals.

  b After posting, flush the static page cache using EAServer Manager or by running a client that calls the flushStaticPageCache method in the Jaguar::Management component.

- **Tune the cache size** The cache size must be sufficient to cache the most frequently requested content (assuming you have excluded seldom-requested files). To estimate the required size, calculate the total size of your static content files, minus the size of excluded files, then multiply by your required hit/miss ratio.

When the cache is tuned, you can further improve the speed at which static page contents are served by disabling the HTTP request log (using the HTTP Config properties in the Server Properties dialog box). However, consider this option carefully as the request data can be helpful to diagnose performance problems that arise from changes in your configuration or user interests. For example, you may find newly posted content creates a large spike in the request pattern. The request log helps you identify popular content that you should add to the cache.

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 page 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 page caching” on page 87 describes how to configure this mechanism.

- **Servlet Java cache** This mechanism caches servlet and JSP responses in core memory. It offers the best performance, but has stricter requirements on what can be cached. For example, you cannot cache servlets that return different content based on request or session parameters. “Using the servlet Java cache” on page 92 describes how to configure this mechanism.
- **Partial page caching** Partial page 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 page caching allows you to cache only the static parts of a response. Partial page caching is supported by a tag library for use in JSPs, and a public API for use in servlets. “Using partial page caching” on page 93 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).

How do I cache JSPs not defined in EAServer Manager?

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. For example, to create and map a Web component for a JSP defined in *myJSP.jsp*:

1 Create a JSP Web component called “myJSP”.

2 Set the file property to “myJSP.jsp”.

3 Create a Web application servlet mapping where *servlet* = “myJSP” and *URL Pattern* = “/myJSP.jsp”.

For more information, see “Creating and configuring JSPs in EAServer” in Chapter 24, “Creating JavaServer Pages,” in the *EAServer Programmer’s Guide*.

Dynamic page caching

Dynamic page 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 page caching instead—see “Using partial page caching” on page 93.

When page 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 six 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. If you do not require this level of refinement in the cache key, consider using the servlet Java cache instead—see "Using the servlet Java cache" on page 92.

Configuring page caching for servlets and JSPs

Use EAServer Manager to enable caching for your servlets and JSPs. To enable caching for JSPs, you must have a Web component defined for the JSP in EAServer Manager. To configure default values for the Web component caching properties, see "Configuring Web application page caching properties" on page 90.

v **To configure page caching for a servlet or JSP:**

1 Expand the Web application folder and highlight the servlet or JSP.

2 Choose File | Web Component Properties.

3 Select the Caching tab.

4 Check Enable Cache. By default, page caching is disabled.

Note The first time you configure page caching for a Web component, the page caching property values default to the same as those in the Web application.

5 To replace the current property values with those in the Web application, select Get WebApp Settings.

6 To include the output from all the pages that the Web component includes or forwards requests to, select Cache Entire Tree. For more information, see "Caching an entire tree" on page 90.

7 Optionally, edit the timeout value, and enter the parameters that you want to use in the key as listed in Table 5-1.

Table 5-1: Page caching properties

Parameter	Description
Cache Timeout	Enter the number of seconds to keep the Web component's content in the cache; the default is 600; a value of 0 indicates no timeout. The timeout value is stored in the com.sybase.jaguar.servlet.cache.timeout property.
Session Local	To include the session ID in the key, select this option; by default, it is not included. The session ID can identify session specific items, such as shopping carts. The value is saved in the com.sybase.jaguar.servlet.cache.use-sessionid property.
Locale Sensitive	Select to include the accept-languages header in the key; by default, it is not included. The value is saved in the com.sybase.jaguar.servlet.cache.locale-sensitive property.
Request Parameters	To include request parameters in the key, enter the parameter names as a comma-separated list. To include all the request parameters, select the Enable All box; it is selected by default. A single servlet can produce different responses based on which parameters it receives. A key that includes the request parameters allows different responses from the same servlet to be cached separately. The request parameters are saved in the com.sybase.jaguar.servlet.cache.request-parameters property.
Session Attributes	To include session attributes in the key, enter the attributes as a comma-separated list. To include all the session attributes, check the Enable All box. The session attributes list is saved in the com.sybase.jaguar.servlet.cache.session-attributes property.
Request Headers	To include request headers in the key, enter the header names as a comma-separated list. For example, if you include the date header, EAServer looks for cache entries whose date headers match the request's date header. The request headers list is saved in the com.sybase.jaguar.servlet.cache.request-headers property.
Message Topics	For a key that includes the number of times a message has been published to a particular topic, enter the message topic. To use this as a key parameter, the message service must be configured; for details, see Chapter 31, "Using the Message Service." This option is useful when a servlet constructs its response based on the values in a database table. A database trigger can be used to call the message service and publish a message to the topic each time the database table is updated. When a servlet or JSP is requested, EAServer can call the message service's getStatistics method to get the total number of messages published to the topic and compare the value to those in the cache entries. The message topics are stored in the `com.sybase.jaguar.servlet.cache.message-topics` property.

Configuring Web application page caching properties

You can configure page caching for a Web application, as well as for a Web component. EAServer first checks the component-level page caching properties, and if none exists, checks the application-level properties. In other words, the Web application settings serve as defaults for the Web component properties described in “Configuring page caching for servlets and JSPs” on page 88.

v **Configuring page caching for a Web application**

1 Display the Page Caching tab in the Web Application Properties dialog box.

2 Optionally, edit the timeout value, and enter the parameters that you want to use in the key. The parameters you enter here are the default settings for all the servlets and JSPs in the Web application. See Table 5-1 on page 89 for a description of the page caching properties.

Note By default, page caching is disabled, and you cannot enable page caching at the Web application-level.

Caching an entire tree

To use page caching for a JSP that forwards request to, or dynamically includes, other JSPs or static files, consider these factors. By default, when you enable page 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 page 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 page 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.

Using page caching with filters that modify a response

When using page caching with filters that wrap a servlet response, EAServer must have access to the response. To provide access to the response, the wrapper must implement a getResponse method that returns a ServletResponse object. To implement the wrapper, you can either:

- Extend the Servlet 2.3 API javax.servlet.http.HttpServletResponseWrapper class, which implements the getResponse method, or
- Extend the javax.servlet.http.HttpServletResponse class and implement the getResponse method yourself.

Also, EAServer caches a servlet response immediately after the servlet executes. Therefore, if a filter modifies a response after the servlet executes, the modifications are not saved to the cache.

For more information about creating and using filters, see Chapter 23, "Using Filters and Event Listeners," in the *EAServer Programmer's Guide*.

Using the servlet Java cache

The servlet Java cache stores servlet output in Java core memory, which offers a faster response than page caching for servlets that can run under these limitations:

- The output does not change during the cache timeout period, and does not depend on request method, parameters, or headers.
- The servlet runs with exact path mappings. Responses are not cached if the servlet uses prefix mappings, default mappings, or extension mappings.
- Cached content is returned without modification.
- Cached response headers are returned without modification, except for:
  - The Set-Cookie header, which depends on the com.sybase.jaguar.servlet.javacache.session property described below.
  - The Connection header, which depends on the request Connection header.
- Return code 200 is used for all cached replies. If the original response uses any other return code, it is not cached.
- Chunked replies are not cached.
- Only responses to GET requests are cached.

Servlets that cannot run under these limitations may still use the page cache.

To enable and configure the servlet Java cache, configure the following servlet properties for each servlet to be cached, using the Advanced tab in the Servlet Properties window:

- com.sybase.jaguar.servlet.javacache.enabled enables the Java cache for the servlet. A value of true enables the Java cache. The default value of false disables the cache. If the Java cache and the page cache are both enabled, the Java cache takes precedence.

- com.sybase.jaguar.servlet.javacache.session specifies how session cookie settings are treated by checking Set-Cookie headers in the request and response headers. Allowable values for this property are:

Value	To indicate
`no` (the default)	No session support. EAServer does not check request Set-Cookie headers or return response Set-Cookie headers.
`keep`	Attempt to preserve valid sessions. If the request includes a session identifier, EAServer checks if the session is valid. If it is not, the response Set-Cookie header is set to indicate an invalid session. Otherwise, no Set-Cookie header is returned.
`create`	Preserve valid session and create a new session if the previous session is invalid. If the request includes a session identifier, EAServer checks if the session is valid. If it is not, the response Set-Cookie header is set to indicate a new session. Otherwise, no Set-Cookie header is returned.

- com.sybase.jaguar.servlet.javacache.maxsize specifies the size, in kilobytes, of the largest reply that can be cached. Responses larger than this are not cached. The default is 8. The hard upper limit is 100. If you set a value greater than 100, the effective cache size is 100K.
- com.sybase.jaguar.servlet.javacache.timeout specifies the time, in seconds, that cached responses remain valid. The default is 60. A negative value specifies an infinite timeout, that is, cached responses do not expire.

Using partial page caching

Partial page 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. Partial page caching is supported by a tag library for use in JSPs, and a public API for use in servlets.

Using the caching tag library

The tag library implementation is provided in *CacheTags.jar*, installed in the *extensions* 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.
messageTopics	A comma-delimited list of message topics to include in the key. If not specified, no message topics are included in the key. This option is useful when a servlet constructs its response based on the values in a database table. A database trigger can be used to call the message service and publish a message to the topic each time the database table is updated. When a servlet or JSP is requested, EAServer can call the message service's getStatistics method to get the total number of messages published to the topic and compare the value to those in the cache entries. To use this as a key parameter, the EAServer message service must be configured and running.
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.

Attribute	Comments
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.
namespace	Specifies what name space the cache is part of. EAServer tracks usage to determine which entry is the least recently used when entries must be removed. Caches in the same name space share the same use statistics, and EAServer evicts the least recently used entry from all the caches in the name space. The default value is "jspfragmentcache" which means unless otherwise specified, every cache is part of the same name space.
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.
size	Specifies the size for this cache. Once the cache is full, entries are evicted based on a least recently used policy. Size is specified as a string using either Mb for megabytes or Kb for kilobytes, for example "10Mb" means 10 megabytes. If not specified, the default is 1Mb.

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.

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	Comments
name	The cache name.
scope	The cache scope. If not specified, the default is application.

Attribute	Comments
parameters	Same as for the cache tag.
attributes	Same as for the cache tag.
messageTopics	Same as for the cache tag.
localeSensitive	Same as for the cache tag.
headers	Same as for the cache tag.

The flushCacheByScope tag

You can use this tag to flush all entries from all caches in the specified scope. Specify the scope as the scope attribute. The default is application, which flushes 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.

Portability considerations

The J2EE specifications do not address HTTP response caching. Therefore, any caching implementation is proprietary. To allow portability of JSPs that use the caching tag library, EAServer includes a nonoperational implementation that you can include when exporting WAR files for deployment to other servers. This JAR file is *EmptyCacheTags.jar* in the *java/taglibs* directory of your EAServer installation. To include it in a WAR file, select the Export Empty Cache Tags option when exporting the Web application as a WAR file or within an EAR file. If you are using jagtool, specify -emptycachetags true in the jagtool options.

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.jaguar.servlet;
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.jaguar.servlet
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, String, String)

Description Creates a new cache.

Syntax

Package	com.sybase.jaguar.servlet
Class	CacheManager

```
public void createCache(String cacheName, String nameSpace, String size)
throws CacheNameException
```

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

nameSpace
Specifies what name space the cache is part of. Caches in the same name space share the same MRU/LRU chains so that when an entry is evicted, it evicts the least recently used in all the caches in the same namespace. If you pass as null, the cache is in the default name space.

size
The cache size, specified as a string using either Mb for megabytes or Kb for kilobytes, for example “10Mb” means 10 megabytes.

CacheManager.getData(String, PageCacheKey)

Description

Retrieves data from the cache.

Syntax

Package	com.sybase.jaguar.servlet
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.jaguar.servlet
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.jaguar.servlet
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)

Description

Flushes caches associated with the specified scope.

Syntax

Package	com.sybase.jaguar.servlet
Class	CacheManager

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

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.

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

Description

Creates a cache key for the specified inputs.

Syntax

Package	com.sybase.jaguar.servlet
Class	CacheManager

public PageCacheKey getCacheKey(HttpServletRequest request, String parameters, String attributes, String headers, String messageTopics, 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.

messageTopics

A comma-separated list of message topics 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.jaguar.servlet.PageCacheKey instance containing the input data.

CHAPTER 6

Database Access Tuning

This chapter describes how to tune connection caches and the settings that affect the performance of the EAServer transaction manager. See Chapter 2, “Understanding Transactions and Component Lifecycles,” in the *EAServer Programmer’s Guide* for more information on EAServer transactions.

In addition to the suggestions in this chapter, consult your database performance and tuning documentation. If you are using Sybase Adaptive Server Enterprise, the *Performance and Tuning Guide* is available on the Sybase Product Manuals Web site at http://sybooks.sybase.com/as.html.

Component design and implementation

The design and implementation of your code to access databases can have a significant effect on performance.

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.

In 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. In stateful components of other types, avoid 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. If you cannot avoid these design patterns, configure a transaction timeout as described in "Transaction timeout" on page 106.

Many design patterns that depend on long-running transactions can be easily modified to use optimistic concurrency control. 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.

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.

When you must 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.

Use database server optimizations

Tune 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 Sybase Product Manuals Web site at http://sybooks.sybase.com/as.html.

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 more information, see "Component properties: Transactions" in the *EAServer Programmer's Guide*. 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 connection caches.

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.

If a component requires different transaction attributes for different contexts, you can create a copy of the component definition in EAServer Manager and change only the transaction attribute.

Clean up connections before releasing them to the cache

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 cached connections; you must explicitly clean up Statement objects before releasing a connection back into the cache. 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 cache. 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 the *EAServer Programmer's Guide*.

Server and component transaction settings

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

Transaction timeout

Make sure you have configured a transaction timeout if your application uses EJB bean-managed transactions, or if you use components of other types that keep transactions open across method calls. The transaction timeout setting specifies the maximum duration of an EAServer transaction. The default configuration allows transactions to remain open indefinitely. A finite timeout allows transactions to be closed when the client crashes or hangs during an open transaction.

You can set the timeout for components on the Resources tab in the EAServer Manager Component Properties dialog box. If you are using jagtool, the property name is com.sybase.jaguar.component.tx_timeout.

You can set a default timeout at the server level by setting server property com.sybase.jaguar.server.tx_timeout (set on the Advanced tab in the Server Properties dialog box). EAServer determines the transaction timeout period as follows:

- If the component transaction timeout property is set to a nonzero value, this is the timeout period.
- Otherwise, the server transaction timeout property is checked (the server transaction timeout is specified by the com.sybase.jaguar.server.tx_timeout property). If the server transaction timeout is non-zero, this specifies the timeout period.

- Otherwise, the component Instance Timeout value is checked. A nonzero value specifies the transaction timeout period as well as the instance timeout period.
- Otherwise, the transaction timeout is infinite.

For both the component and server setting, the timeout period is configured in seconds, with 0 indicating no timeout. The default for a new server is 0. When specifying timeouts, Sybase recommends a resolution of 5 seconds. Network transport time is included in the measured timeout period. You may need to configure a larger timeout period if clients connect over slow networks.

EAServer checks for timeouts after each method returns. Your component will not be deactivated in the middle of an invocation because of a timeout. When a transaction times out, the next method invocation in the client-side ORB throws the CORBA::TRANSACTION_ROLLEDBACK system exception.

Transaction memory table size

The transaction manager uses an internal table to track the status of pending transactions. You can tune the size of the table by setting the server property com.sybase.jaguar.server.jta.tranTableSize. The default is 1024.

This table provides cached storage of the transaction information. If the number of pending transactions exceeds the table size, new memory is allocated and deallocated as needed. For best performance, set the value to at least one-half the maximum number of simultaneous transactions expected in your application.

Unexpected deadlock errors

If you see unexpected deadlock or slow database throughput when executing transactional components, check for the following setting in the properties of the component that initiates the transaction:

```
com.sybase.jaguar.component.iso_level=serializable
```

This setting causes the transaction to run with serializable isolation level, which can cause deadlocks or degraded performance for any component that uses JDBC connections, other than EJB 1.0 session beans. For components that are not EJB 1.0 session beans, delete this setting using the Advanced tab in the component properties dialog box. This setting was erroneously used as a default for new components in earlier versions of EAServer.

If the component is an EJB 1.0 session bean, verify that the Transaction Isolation Level setting on the Transactions tab matches your application requirements.

Connection cache settings

Connection caches increase performance by allowing reuse of database connections, eliminating the overhead of repeatedly creating and destroying connections to the same database. For general information on connection caches, see:

- Chapter 4, “Database Access,” in the *EAServer System Administration Guide* describes how to create and configure connection caches.
- Chapter 26, “Using Connection Management,” in the *EAServer Programmer’s Guide* describes how to code your applications to use connection caches.

The following sections describe how to tune connection cache settings for the best performance.

Tuning the cache size

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

Table 6-1: Properties to configure connection cache size

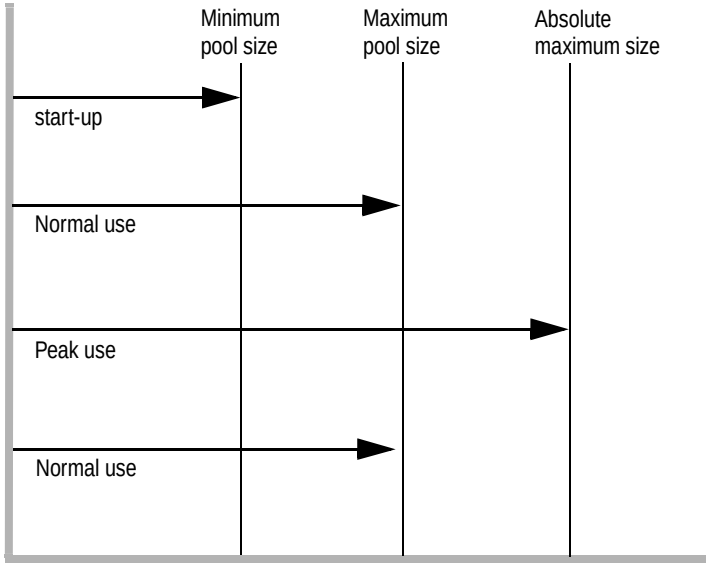
Property	Description
Minimum Connection Pool Size	The minimum number of pooled connections, allocated at server start-up. If not set, the default is 0. With jagtool, you can set as com.sybase.jaguar.conncache.poolsize.min.

Property	Description
Maximum Connections	The absolute maximum number of connections that can be created from the cache. Requests for excess connections either block or fail. A value of 0 indicates that there is no limit. With jagtool, you can set as com.sybase.jaguar.conncache.poolmanager.maxconnection.
Maximum Connection Pool Sizes	The maximum number of connections that can be cached. If connections are allocated beyond this number, the cache manager drops the excess connections when they are released. With jagtool, you can set as com.sybase.jaguar.conncache.poolsize.max.
Wait for Connections	When the maximum connections limit is reached, specifies whether requests for excess connections fail immediately or wait until a connection is released. If this setting is enabled, the request waits. With jagtool, you can set as com.sybase.jaguar.conncache.wait.
Pooled Connection Idle Timeout	Specifies the number of seconds an idle connection remains in the pool before it is dropped. The default is 300 seconds (5 minutes). Idle connections are dropped until the minimum pool size is reached. To disable the monitoring of idle connections, set to a negative value. For more information, see “Disabling the cache size monitor thread” on page 111. With jagtool, you can set as com.sybase.jaguar.conncache.idletimeout.
Pooled Connection Refresh Rate	The refresh rate for the cache, that is, how often the cache manager checks for excess connections that have been idle longer than the idle timeout period. The default is 600 seconds (10 minutes). To disable the monitoring of idle connections, set to a negative value. For more information, see “Disabling the cache size monitor thread” on page 111. With jagtool, you can set as com.sybase.jaguar.conncache.refreshrate.

Set the pool size so the majority of database connections are taken from the cache. You can tune the minimum pool size and refresh rate 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 connection cache.

Figure 6-1: Connection cache growth patterns



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

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

During peak use, additional connections may be required beyond the maximum pool size, up to the absolute maximum. When these excess connections are released, they are closed rather than placed in the cache. Setting an absolute maximum prevents your application from overwhelming the database with too many connections or exceeding database license limits. You can set the Wait for Connections setting to determine what happens when your application asks for connections in excess of the absolute maximum size. You can also set the absolute maximum to 0 to indicate no limit.

Note When many connections are created in excess of the maximum pool size, you may see a drop in performance. You may also see a drop in performance if excess, unpooled connections are rapidly created and destroyed to service short transactions. To avoid these scenarios, raise the maximum pool size.

When the activity level drops, the cache manager removes idle connections if you have configured a refresh rate and idle connection limit. If there is no activity, the cache size drops back down to the minimum.

Monitoring cache activity

You can monitor the cache activity in EAServer Manager to determine how effective the cache settings are. See “Monitoring connection caches and managed connection factories” in Chapter 11, “Runtime Monitoring,” in the *EAServer System Administration Guide* for more information.

Disabling the cache size monitor thread

The cache manager runs a thread to monitor the number of connections and close those that are in excess of the minimum size when they have been idle longer than the idle timeout period. If you do not want the cache size reduced below the maximum pool size, you can set the Idle Timeout and Refresh Rate to negative values. When you use these settings, the cache monitor thread terminates when the maximum pool size is reached.

Tuning caches 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 “Using CMP JDBC wrapper drivers” on page 60.

Remove unused connection caches

Remove unused connection caches or set the Minimum Pool Size setting to 0. EAServer allocates the minimum pool size for each cache, and unused connections waste memory and network resources.

Sanity checking

If sanity checking is enabled, the cache manager runs a stock query to verify that connections are ready for use before placing them back in the cache. 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 sanity checking on the Caching tab in the EAServer Manager Cache Properties dialog box or by using jagtool to set the com.sybase.jaguar.conncache.checkallowed property to false.

SQL tracing

You can enable tracing of the commands issued through each connection in a cache on the SQL Tracing tab in the EAServer Manager Connection Properties dialog box. For details, see "SQL tracing properties" in the *EAServer System Administration Guide*.

SQL tracing can help you debug performance issues that are caused by poor database response times. However, the file I/O does increase the response time of EAServer, so disable SQL tracing unless you are debugging database performance issues.

Using the caching APIs

In Java/CORBA, C, C++, and ActiveX components, you must use the EAServer APIs to obtain a cache reference and retrieve connections from the cache. For best performance, use by-name lookup to obtain cache handles and pass the cache handle when obtaining and releasing connections. Doing so avoids internal table searches in the cache manager. For information on using these APIs, see Chapter 26, "Using Connection Management," in the *EAServer Programmer's Guide*.

Dynamic prepare on jConnect caches

Ensure that connection caches 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. In EAServer 4.1.1 and later, a warning is printed in the log file if a connection cache has this property set to TRUE.

Database and 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. If using Sybase Adaptive Server Enterprise, see the *Performance and Tuning Guide*, available in the Adaptive Server Enterprise Documentation on the Sybase Web site at http://sybooks.sybase.com/as.html.

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

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 7, “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 replicate client session data between servers. You can minimize the performance impact by minimizing your use of stateful components and HTTP session storage, and by tuning the state replication mechanisms.

Cluster settings that affect performance

You can tune these cluster properties for best performance.

Heartbeat detection

Heartbeat detection determines how often the cluster's name servers test whether each server is online. When the name server detects a server has gone offline, it stops directing IIOP clients to that server.

Heartbeat detection affects only IIOP clients and interserver calls. If you partition components, that is, you do not install all components into every server, interserver calls are required when a component calls another components that is not available on the same server. In these cases, enable heartbeat detection and tune the test interval. A shorter interval minimizes the chance that clients attempt to connect to servers that have gone offline, but if the interval is too short, you can waste resources with excessive broadcasting from the name servers to the member servers. The default of two minutes works well for most applications.

If your application does not have any IIOP clients or use interserver calls, you can disable heartbeat detection in EAServer. (Note HTTP client load balancing and failover are performed outside of EAServer.)

To change these setting in EAServer Manager, follow the instructions in "Heartbeat detection" in Chapter 6, "Clusters and Synchronization," in the *EAServer System Administration Guide*. To change this setting with jagtool, use the set_props command to set these properties for the primary server:

- com.sybase.jaguar.server.CosNaming.heartbeat
- com.sybase.jaguar.server.CosNaming.heartbeatfrequency

Synchronize the cluster after modifying these settings.

Load balancing policy

EAServer supports several algorithms to balance the IIOP client load between servers in the cluster. The EAServer name service uses the specified algorithm to determine which server each client connects to when the client resolves the component name. For more details, see "Load balancing overview" in Chapter 7, "Load Balancing, Failover, and Component Availability," in the *EAServer System Administration Guide*.

These settings affect only applications that use IIOP clients or that require inter-server calls between cluster members. The settings do not affect Web applications, since HTTP client load balancing is done outside of EAServer.

You can configure the load balancing policy to ensure the IIOP client load is evenly distributed. You can also change connection settings in your client programs to help ensure an even load distribution, as described in "IIOP client settings that affect load balancing" on page 118. EAServer supports these distribution policies:

- **Random weighted** Static, even distribution of naming requests using a random selection algorithm to map name requests to destination servers. The load is likely to balance evenly over time, but can vary due to the random nature of the distribution algorithm and the fact that some components load the server more heavily than others.
- **Round-robin** Static, even distribution of naming requests using a round-robin selection algorithm to map name requests to destination servers. The load is likely balance evenly over time, but can vary due to the fact that some components load the server more than others.
- **Weighted** Same as random, but the selection is weighted using the weights you assign to each server. Over time, each server carries a portion of the load in proportion to the weight that you assign to each server. Use this algorithm if some machines in the cluster can support more clients than others.
- **Adaptive** Same as random, but the selection is weighted using weights that are calculated based on a sampling of each server's existing load. This policy provides the highest assurance that the load will balance evenly across servers at any time. However, the broadcasting and collection of sampled load data does add slight overhead.

To configure these settings in EAServer Manager, follow the instructions in "Configuring load balancing" in Chapter 7, "Load Balancing, Failover, and Component Availability," in the *EAServer System Administration Guide*. To configure these settings with jagtool, use the set_props command to set these properties for the cluster:

- com.sybase.jaguar.cluster.DLBbroadcastInterval
- com.sybase.jaguar.cluster.DLBcalculateInterval
- com.sybase.jaguar.cluster.DLBenable
- com.sybase.jaguar.cluster.DLBmaxWeight
- com.sybase.jaguar.cluster.DLBpolicy
- com.sybase.jaguar.cluster.DLBrefreshInterval
- com.sybase.jaguar.cluster.DLBsampleInterval
- com.sybase.jaguar.cluster.DLBweights

Synchronize the cluster after modifying these settings.

Using partitioning You can also further balance the load by partitioning components and Web applications between different logical servers. For example, you might install your Web application in the logical server Jaguar1, using this server name to start with this configuration on two machines in the cluster, and install the packages containing your components in the logical server Jaguar2, using this server name to start the Jaguar2 configuration on four machines in the cluster. A drawback of this configuration is that component invocations from the Web tier and intercomponent calls can require interserver communication over the network, which is slower than in-server invocations and prevents the use of some optimizations such as EJB local interfaces.

IIOP client settings that affect load balancing

In a cluster, the load balancing policy decides the algorithm used to map client naming requests to destination servers. While the load balancing policy can evenly distribute the initial client connections, long running IIOP 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 these settings in the client runtime:

- **Socket reuse limit** Limits how many times the client runtime reuses a network connection to call methods from one server. The default is 0, which indicates no limit. The default is inappropriate for a long-running client program that calls many methods from servers in a cluster. If sockets are reused indefinitely, the client can build an affinity for servers that it has already connected to rather than randomly distributing its server-side processing load among all the servers in the cluster. In these cases, tune the property to best balance client performance against cluster load distribution. In Sybase testing, settings between 10 and 30 proved to be a good starting point. If the reuse limit is too low, client performance degrades. The following table describes how to set this property for each client type:

Client type	Property name
Java	com.sybase.CORBA.socketReuseLimit
EJB	com.sybase.ejb.socketReuseLimit
C++, ActiveX, and PowerBuilder	ORBsocketReuseLimit You can also set the environment variable JAG_SOCKETREUSELIMIT.

- **Idle connection timeout** 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. The following table describes how to set this property for each client type:

Client type	Property name
Java	com.sybase.CORBA.idleConnectionTimeout You must also specify a garbage collection period by setting the com.sybase.CORBA.GCInterval property to an equal or lesser value. Connections are only closed when garbage collected.
EJB	com.sybase.ejb.idleConnectionTimeout You must also specify a garbage collection period by setting the com.sybase.ejb.GCInterval property to an equal or lesser value. Connections are only closed when garbage collected.
C++, ActiveX, and PowerBuilder	ORBidleConnectionTimeout You can also set the environment variable JAG_IDLECONNECTIONTIMEOUT.

- **Connection timeout** Sets 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.connectionTimeout
EJB	com.sybase.ejb.connectionTimeout
C++, ActiveX, and PowerBuilder	Not supported.

For information on setting client runtime properties for EJB, Java, C++, and ActiveX clients, see these chapters in the *EAServer Programmer's Guide*:

- Chapter 8, "Creating Enterprise JavaBeans Clients"
- Chapter 12, "Creating CORBA Java Clients"
- Chapter 15, "Creating CORBA C++ Clients"
- Chapter 20, "Creating ActiveX Clients"

For information on PowerBuilder clients, see the PowerBuilder *Application Techniques* manual.

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 "Clustered Web applications" in Chapter 21, "Creating Web Applications," in the *EAServer Programmer's Guide*.

Chapter 5, "Web Application Tuning," describes the Web application settings that you can tune for single-server deployments. In a clustered deployment, these additional settings affect performance:

- Your choice of session replication options

- The cache size, if replicating sessions in memory

HTTP session replication mechanism

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 can use a remote database server or in-memory storage.

If using in-memory replication, EAServer replicates data between *mirror pairs*, that is, a pair of servers configured to share the same user session. In-memory replication can perform better than using a database, but requires more memory on each server. Specify a cache size to constrain the memory required by the cache, and specify a timeout to free up memory used by idle sessions. To set these parameters, follow the instructions in "Clustered Web applications" in Chapter 21, "Creating Web Applications," in the *EAServer Programmer's Guide*.

If using database replication, all servers in the cluster store session data in a remote database. This mechanism can perform slower than in-memory replication, but requires less memory. Also, EAServer can share a client's session data on more than two servers. If using this technique:

- Tune the connection cache settings as described in "Connection cache settings" on page 108.
- Make sure the table used is indexed. For information on creating the database table, see "Table schema for binary storage" in Chapter 29, "Configuring Persistence Mechanisms," in the *EAServer Programmer's Guide*.

Lazy session validation

In the default configuration, EAServer validates a client's HTTP session during each request. In a clustered deployment, session validation is more resource intensive since EAServer stores the session in a database or using a replicated cache. If you enable lazy validation, EAServer validates the session only when a servlet or JSP calls ServletRequest.getSession() or ServletRequest.getSession(boolean). To enable this setting, set the com.sybase.jaguar.webapplication.lazydistributedhttpsessionvalidation Web application property to true.

Lazy validation can improve performance. However, enabling lazy authentication has the following side effects:

- The last-accessed-time session attribute is set only when the servlet or JSP accesses the session. Consequently, the session may expire sooner than expected if the client accesses only static pages or servlets and JSPs that do not access the session data.
- When the session is invalidated, the client is not assigned a new session until they request a page that requires a session.
- The client's security credentials (if any) are available only to JSPs and servlets that are marked protected via the security constraints property. Other pages cannot retrieve the client's credentials—the ServletRequest.getUserPrincipal() method returns null even though the client is logged in.

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 have the Automatic Failover option enabled on the Transactions tab in the EAServer Manager Component Properties dialog box. To configure this setting with jagtool or in the EJB deployment descriptor, set the property com.sybase.jaguar.component.auto.failover to true.

Component state replication

If your application uses stateful components such as EJB stateful session beans, you must configure a mechanism for EAServer to replicate state data between servers in the cluster. You can use a remote database server or in-memory replication. For more information, see Chapter 28, "Configuring Persistence for Stateful Session Components," in the *EAServer Programmer's Guide*.

If using in-memory replication, EAServer replicates state data between *mirror pairs*, that is, a pair of servers configured to share the same user session. In-memory replication can perform better than using a database, but requires more memory on each server. Configure the cache size and session timeout to control how much memory the cache uses. For instructions, see "Requirements for in-memory stateful failover" in Chapter 29, "Configuring Persistence Mechanisms," in the *EAServer Programmer's Guide*:

If using database replication, all servers in the cluster store session data in a remote database. This mechanism can perform slower than in-memory replication, but requires less memory. Also, EAServer can share a client's session data on more than two servers. If using this technique:

- Tune the connection cache settings as described in "Connection cache settings" on page 108.
- Make sure the table used is indexed. For information on creating the database table, see "Table schema for binary storage" in Chapter 29, "Configuring Persistence Mechanisms," in the *EAServer Programmer's Guide*.

EJB CMP entity bean instance and query caching

EJB CMP entity bean caching is described in "Entity instance and query caching" on page 63. When using this feature in a clustered deployment, you can configure the cache settings to determine whether cached data is synchronized between servers in the cluster. "Cache coherency and transaction consistency" on page 64 discusses the pros and cons of cache synchronization. To configure this setting, follow the instructions in "Configuring object caching" on page 65.

CHAPTER 8

Message Service Tuning

This chapter describes how to tune the EAServer message service for best performance. Before reading this chapter, you should be familiar with message service configuration and programming, as described in these chapters:

- Chapter 8, "Setting up the Message Service," in the *EAServer System Administration Guide*
- Chapter 31, "Using the Message Service," in the *EAServer Programmer's Guide*

Best practices for coding

For best performance, follow these recommendations when coding to the JMS or message service APIs:

- **Consider storage types carefully** EAServer supports two options for message storage and delivery:
  - Persistent messages are stored in a remote database and also cached in memory. Since the messages are stored in a database, they are not lost when the server goes offline or restarts.
  - Transient messages are stored in memory only, and can be lost if the server goes offline or restarts.

EAServer processes transient messages faster because they do not require database interaction. However persistent messages offer more reliable delivery (depending on the settings described "Queue and topic settings" on page 128).

Use transient messages if your application requirements allow the possibility of lost messages. 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.

- **Use transactions only when necessary** Transacted sessions create additional overhead. If some messages require transacted sessions but others do not, create a separate session for the nontransactional messages.
- **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 values, consider using compression.
- **Clean up after yourself** Close resources such as connections, sessions, queues, and topics as soon as you no longer need them.

Global message service settings

These settings can be configured by running the Message Service Configuration wizard in EAServer Manager or by editing the *MessageServiceConfig.props* file in the EAServer *Repository/Components/CtsComponents* subdirectory. For more information, see Chapter 8, "Setting up the Message Service," in the *EAServer System Administration Guide*.

Database and connection cache

You can specify the connection cache used by the message service in the configuration wizard. Tune the cache settings as described in "Connection cache settings" on page 108. 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.

Tracing

For best performance, make sure tracing is disabled. Set the properties listed in Table 8-1 to false; these properties are set in the *MessageServiceConfig.props* file.

Table 8-1: Message service tracing properties

Property	Specifies
cms.debug	General tracing
cms.debug.session	Session level tracing
cms.debug.network	Network level tracing

Other global settings

You can modify these settings by editing the *MessageServiceConfig.props* file.

default.maximum

This global property configures a maximum limit for persistent messages stored in the <system> queue and user defined queues with the queue maximum property set to 0 or a negative number (see “Queue size” on page 130). To set the global property, specify the message limit, for example:

```
default.maximum=120
```

The default is 100. This setting restricts in-memory caching of persistent messages; you can change it to tune the memory used to hold persistent messages in the <system> queue and user defined queues where the maximum property is 0 or a negative value.

This setting also determines how many persistent messages EAServer reads into memory during message service initialization. A large default size can delay server startup when there is a large backlog of unprocessed messages, since the message service reads this many messages into memory when initializing.

The setting does not restrict the number of transient messages in the system queue or in user defined queues where the maximum property is 0 or a negative value.

session.timeout

This property specifies the default timeout for temporary message queues for which you have not set an explicit timeout. Specify the timeout in seconds. The default is 60. The minimum value is 5; values less than this have the same effect as setting the timeout to 5 seconds.

Queue and topic settings

These settings affect the performance of the applications message consumers and message producers. You can configure them in the message queue and connection factories that you create in EAServer Manager. To associate message queue properties with a topic, specify the message queue name in the connection factory properties, then use the connection factory to create topics in your application.

REQUIRES_ACKNOWLEDGE

In queue properties, this setting specifies whether EAServer must redeliver messages that the consumer has not acknowledged. If set to false, messages can be lost if the client fails while the message is being delivered. A value of true guarantees delivery of persistent messages, and guarantees delivery of transient messages as long as the server does not fail or shut down (contingent on the queue settings and message time-to-live property). Setting this property to false for the connection factory yields significantly improved throughput for bulk publishing of transient messages.

REQUIRES_TRANSACTION or SUPPORTS_TRANSACTION

In queue properties, if you set REQUIRES_TRANSACTION to true, EAServer guarantees that persistent messages are delivered only once. In the connection factory properties, the REQUIRES_TRANSACTION and SUPPORTS_TRANSACTION properties determine whether EAServer processes persistent messages in the context of component- or client-initiated transactions. To improve throughput for bulk publishing, sending, or receiving transient messages, set these properties to false.

Quality of protection

In queue properties, the qop setting specifies the required level of SSL security. The default of “none” allows the use of plain IIOP. For best performance, use the default unless the application requires the additional security provided by SSL.

Tables for persistent messages

By default, all persistent messages are stored in one database table named message_queue. To specify a different table, set the table property for the message queue in EAServer Manager. You may get better performance using dedicated tables for each queue and topic. In any case, make sure the table used is indexed in the database.

Queue size

Set the queue maximum property to constrain the size of the in-memory queue. A positive number specifies how many messages can be stored in memory at once. When the number of messages exceeds the specified limit, EAServer discards messages in the order that they would have been retrieved.

A negative number or the default of 0 specifies that there is no size limit (other than that imposed by available memory) for the number of transient messages in the queue. In this case, the number of persistent messages in the queue is limited by the default.maximum setting in the *MessageServiceConfig.props* file—see "Other global settings" on page 127.

Tune the queue size to balance 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.

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

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

Timeout settings

In queue or topic properties, the 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. The default of 0 specifies no timeout. Any transient messages that are in memory when a timeout occurs are discarded.

If your application uses temporary queues or topics, the global session.timeout property specifies the default timeout–see "Other global settings" on page 127. If this value is not set, the default is 60 seconds. You can override the default by using a connection factory to create the temporary queue or topic, with a timeout specified in the EAServer Manager queue indicated by the connection factory's CONFIG_QUEUE property.

Thread pools

Using a thread pool can significantly improve performance. For applications that use the JMS API, create thread pools in EAServer Manager as described in "Thread pools" in the *EAServer System Administration Guide*. If using jagtool, create and configure thread pools with the jmscreate and jmsset_props commands, respectively. In applications that call the EAServer CtsComponents::MessageService API, you can create thread pools programmatically. For details, see "Creating thread pools programmatically" in the *EAServer Programmer's Guide*.

For EJB MDB components and other components that you install as message listeners, EAServer by default delivers messages using a single worker thread (the default thread pool is <system> and cannot be modified). 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, use a customized thread pool to increase throughput. When you use a thread pool with multiple worker threads, EAServer creates multiple component instances that run in different worker threads to process messages concurrently.

When a thread pool is used for client notification, the message queue object is implemented with a specialized server IIOP handler that uses only a few waiting threads to handle blocking receive calls, so it avoids waking large numbers of threads for client notification.

- **Assigning a thread pool to an MDB**

  Assigning a thread pool to an MDB component allows EAServer to create multiple instances for concurrent message processing on different threads. Create and assign the thread pool as follows:

  1. Create a thread pool for component notification, and set the workers property to a value greater than 1.
  2. Assign this thread pool to the MDB:
     1. In EAServer Manager, highlight the MDB, and select File | Properties.
     2. On the MDB Type tab, append the name of the thread pool you just created to the Listener name. For example, if you created a thread pool called "threads1" and the Listener Name is *MyPkg/MyComp*, change the Listener Name to `MyPkg/MyComp[threads1]`.

     See Chapter 31, "Using the Message Service," in the *EAServer Programmer's Guide* for more information about configuring MDBs.

- **Assigning a thread pool to a message listener using non-EJB components**

  EAServer allows you to create non-EJB components that act as message listeners, as described in "Listeners" in the *EAServer System Administration Guide*. Use a thread pool to allow concurrent message processing by multiple instances of the component. To create and assign the thread pool:

  1 Verify that the component supports concurrent execution. For more information, see "Concurrency" on page 30.

  2 Create a thread pool for component notification, and set the workers property to a value greater than 1.

  3 In the message queue where you installed the component as a listener, specify the thread pool name in brackets after the component name, using the format:

  ```
  package_name/component_name[threadpool_name]
  ```

- **Using a thread pool for client notifications**

  Using thread pools to improve performance is generally suitable only for high-volume client notification with transient messages. When message delivery is transactional or IIOP/SSL via the QOP property, the thread pool's reader and writer threads are not used. To create client applications that use the thread pool:

  1 Create a thread pool. For initial testing, set the value of readers to "3", writers to "2", and workers to "0". Later, based on your own performance measurements, you can increase the number of reader and writer threads if the change improves throughput.

  2 Create a connection factory or modify the one you already use, and set the THREAD_POOL property to the name of the thread pool.

  3 In your client code, use this connection factory to create the topics or queues that you use to create and receive messages.

Shared listeners

In client applications, a shared listener can greatly improve performance for nondurable topic subscribers by creating a single message queue for all the topic subscriptions. To use this feature:

1 In EAServer Manager create a topic connection factory (or modify the one that you use already). Set the SHARED_LISTENER property to true.

2 In your code:

   a Use this connection factory to create your topic subscriptions.

   b Install a message listener on the first topic subscription, then each nondurable subscription that uses the connection, receives messages from this listener.

EAServer imposes two restrictions for shared listeners:

- Do not call setMessageListener with a null parameter. This shuts down the current listener, which may be in use by other subscribers.
- Do not call setMessageListener with the name of another listener, which shuts down the current listener and registers the new listener.

The key log table

The message service uses a database table named key_log to control the processing of messages with duplicate key values. If you set the IGNORE_DUPLICATE_KEY option for the queue properties, or pass this option to the send or publish methods, EAServer must check for duplicate keys. To do so, the message service inserts the key to the key_log table, which has a unique index on the message key value and queue name. A failed insert indicates that the key is a duplicate.

The key_log table can grow large enough to affect database and message service performance. In this case, you should periodically remove entries that are old enough that the key is unlikely to be reused.

v **To control the size of the key log:**

1 Add a column to the key_log table using the database date type, such as datetime for Sybase databases. Modify the kl.create property in *MessageServiceConfig.props* so that the column is included when EAServer creates the table.

2 Set the column default to the current date and time. For example, specify getdate() if using a Sybase database server. Specify the default in the database table schema, or modify the kl.create property in *MessageServiceConfig.props* to insert the value.

3 Periodically run a database command to delete rows that are more than *X* days old, where *X* is a value determined heuristically to specify entries that are old enough that the key is unlikely to be reused. If you obtain message keys from an external system and forward them to the EAServer message service, consider the likelihood that the external system will send a message with the same key more than once.

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

W