Sun Java System Application Server Enterprise Edition 8.2 Deployment Planning Guide

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 819–4741–11
August 2007

070813@18135

Contents

Figures

Tables

Examples

Preface

Deployment Planning Guide explains how to build a production deployment.

This preface contains information about and conventions for the entire Sun Java™ System Application Server documentation set.

Application Server Documentation Set

The Application Server documentation set describes deployment planning and system installation. The Uniform Resource Locator (URL) for stand-alone Application Server documentation is `http://docs.sun.com/app/docs/coll/1310.4`. The URL for Sun Java Enterprise System (Java ES) Application Server documentation is `http://docs.sun.com/app/docs/coll/1310.3`. For an introduction to Application Server, refer to the books in the order in which they are listed in the following table.

TABLE P–1 Books in the Application Server Documentation Set

Book Title	Description
Release Notes	Late-breaking information about the software and the documentation. Includes a comprehensive, table-based summary of the supported hardware, operating system, Java Development Kit (JDK™), and database drivers.
Quick Start Guide	How to get started with the Application Server product.
Installation Guide	Installing the software and its components.
Deployment Planning Guide	Evaluating your system needs and enterprise to ensure that you deploy the Application Server in a manner that best suits your site. General issues and concerns that you must be aware of when deploying the server are also discussed.
Developer's Guide	Creating and implementing Java 2 Platform, Enterprise Edition (J2EE™ platform) applications intended to run on the Application Server that follow the open Java standards model for J2EE components and APIs. Includes information about developer tools, security, debugging, deployment, and creating lifecycle modules.
J2EE 1.4 Tutorial	Using J2EE 1.4 platform technologies and APIs to develop J2EE applications.

TABLE P–1 Books in the Application Server Documentation Set *(Continued)*

Book Title	Description
Administration Guide	Configuring, managing, and deploying Application Server subsystems and components from the Administration Console.
High Availability Administration Guide	Post-installation configuration and administration instructions for the high-availability database.
Administration Reference	Editing the Application Server configuration file, `domain.xml`.
Upgrade and Migration Guide	Migrating your applications to the new Application Server programming model, specifically from Application Server 6.x and 7. This guide also describes differences between adjacent product releases and configuration options that can result in incompatibility with the product specifications.
Performance Tuning Guide	Tuning the Application Server to improve performance.
Troubleshooting Guide	Solving Application Server problems.
Error Message Reference	Solving Application Server error messages.
Reference Manual	Utility commands available with the Application Server; written in man page style. Includes the `asadmin` command line interface.

Related Documentation

Application Server can be purchased by itself or as a component of Java ES, a software infrastructure that supports enterprise applications distributed across a network or Internet environment. If you purchased Application Server as a component of Java ES, you should be familiar with the system documentation at `http://docs.sun.com/coll/1286.2`. The URL for all documentation about Java ES and its components is `http://docs.sun.com/prod/entsys.5`.

For other Sun Java System server documentation, go to the following:

- Message Queue documentation
- Directory Server documentation
- Web Server documentation

Additionally, the following resources might be useful:

- The J2EE 1.4 Specifications (`http://java.sun.com/j2ee/1.4/docs/index.html`)
- The J2EE 1.4 Tutorial (`http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html`)
- The J2EE Blueprints (`http://java.sun.com/reference/blueprints/index.html`)

Default Paths and File Names

The following table describes the default paths and file names that are used in this book.

TABLE P–2 Default Paths and File Names

Placeholder	Description	Default Value

install-dir	Represents the base installation directory for Application Server.	Sun Java Enterprise System (Java ES) installations on the Solaris™ platform:
		`/opt/SUNWappserver/appserver`
		Java ES installations on the Linux platform:
		`/opt/sun/appserver/`
		Other Solaris and Linux installations, non-root user:
		user's home directory`/SUNWappserver`
		Other Solaris and Linux installations, root user:
		`/opt/SUNWappserver`
		Windows, all installations:
		SystemDrive`:\Sun\AppServer`

domain-root-dir	Represents the directory containing all domains.	Java ES installations on the Solaris platform:
		`/var/opt/SUNWappserver/domains/`
		Java ES installations on the Linux platform:
		`/var/opt/sun/appserver/domains/`
		All other installations:
		install-dir`/domains/`

domain-dir	Represents the directory for a domain. In configuration files, you might see *domain-dir* represented as follows: `${com.sun.aas.instanceRoot}`	*domain-root-dir*`/`*domain-dir*
instance-dir	Represents the directory for a server instance.	*domain-dir*`/`*instance-dir*

Typographic Conventions

The following table describes the typographic changes that are used in this book.

TABLE P–3 Typographic Conventions

Typeface	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories, and onscreen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% you have mail.`
`AaBbCc123`	What you type, contrasted with onscreen computer output	`machine_name%` **`su`** `Password:`
AaBbCc123	A placeholder to be replaced with a real name or value	The command to remove a file is `rm` *filename*.
AaBbCc123	Book titles, new terms, and terms to be emphasized (note that some emphasized items appear bold online)	Read Chapter 6 in the *User's Guide*. A *cache* is a copy that is stored locally. Do *not* save the file.

Symbol Conventions

The following table explains symbols that might be used in this book.

TABLE P–4 Symbol Conventions

Symbol	Description	Example	Meaning
`[ ]`	Contains optional arguments and command options.	`ls [-l]`	The `-l` option is not required.
`{ | }`	Contains a set of choices for a required command option.	`-d {y|n}`	The `-d` option requires that you use either the `y` argument or the `n` argument.
`${ }`	Indicates a variable reference.	`${com.sun.javaRoot}`	References the value of the `com.sun.javaRoot` variable.
-	Joins simultaneous multiple keystrokes.	Control-A	Press the Control key while you press the A key.
+	Joins consecutive multiple keystrokes.	Ctrl+A+N	Press the Control key, release it, and then press the subsequent keys.

TABLE P–4 Symbol Conventions *(Continued)*

Symbol	Description	Example	Meaning
→	Indicates menu item selection in a graphical user interface.	File → New → Templates	From the File menu, choose New. From the New submenu, choose Templates.

Documentation, Support, and Training

The Sun web site provides information about the following additional resources:

- Documentation (`http://www.sun.com/documentation/`)
- Support (`http://www.sun.com/support/`)
- Training (`http://www.sun.com/training/`)

Third-Party Web Site References

Third-party URLs are referenced in this document and provide additional, related information.

Note – Sun is not responsible for the availability of third-party web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused or alleged to be caused by or in connection with use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. To share your comments, go to `http://docs.sun.com` and click Send Comments. In the online form, provide the full document title and part number. The part number is a 7-digit or 9-digit number that can be found on the book's title page or in the document's URL. For example, the part number of this book is 819-4741.

◆ ◆ ◆ CHAPTER 1

Product Concepts

The Sun Java System Application Server provides a robust platform for the development, deployment, and management of J2EE applications. Key features include scalable transaction management, container-managed persistence runtime, web services performance, clustering, high availability session state, security, and integration capabilities.

Application Server is available in three editions, each of which is designed to provide specific functionality for specific usage scenarios and service levels.

Platform Edition is free and is intended for software development and department-level production environments. It provides a production platform for deploying Java EE applications and is integrated with the Solaris operating system. It contains a developer-friendly, lightweight J2EE container. It is simple to configure because of the small number of processes and is supported by a fast and efficient deployment interface. Application Server Platform Edition is also integrated with some developer tools, including the NetBeans 5.5 integrated development environment (IDE).

Standard Edition is different from Platform Edition in the following ways:

- Application Server Standard Edition provides a high-performance web container that scales better than Application Server Platform Edition
- Application Server Standard Edition provides enhanced multi-machine administration capabilities that allow you to control more than one server from a single console of a remote server.

Application Server Standard Edition provides the following service availability capabilities:

- The load balancing feature allows you to forward client requests to more than one application server to achieve minimum response time and better throughput.
- More than one application server can be running in clustering for better scalability and failover.

Application Server Enterprise Edition adds data availability to Application Server Standard Edition. Application Server Enterprise Edition uses the high availability database (HADB) as a highly available (HA) session store. The conversational states are stored in HADB and are recoverable for the HTTP sessions and SFSBs. Availability provides failover protection of application server instances in a cluster. Application Server Enterprise Editionis most suitable for enterprise-scale applications and service deployment where session state is important.

This chapter covers the following topics:

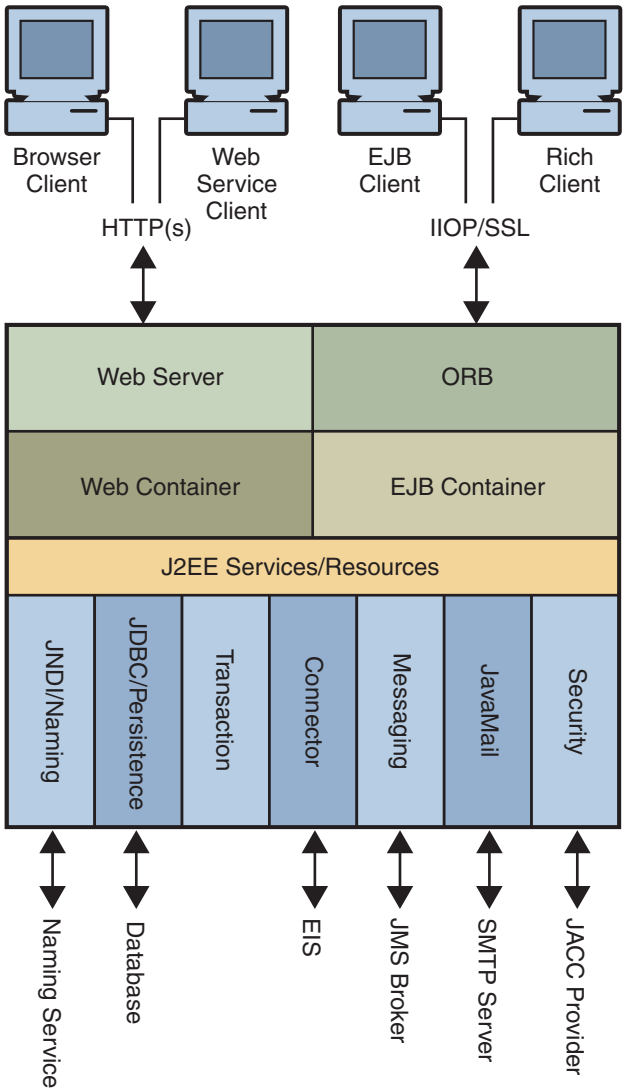
J2EE Platform Overview

The Application Server implements Java 2 Enterprise Edition (J2EE) 1.4 technology which defines the standard for developing multitier enterprise applications. The J2EE platform simplifies enterprise applications by basing them on standardized, modular components, by providing a complete set of services to those components, and by handling many details of application behavior automatically, without complex programming.

J2EE Applications

J2EE applications are made up of components such as JavaServer Pages (JSP), Java servlets, and Enterprise JavaBeans (EJB) modules. These components enable software developers to build large-scale, distributed applications. Developers package J2EE applications in Java Archive (JAR) files (similar to zip files), which can be distributed to production sites. Administrators install J2EE applications onto the Application Server by deploying J2EE JAR files onto one or more server instances (or clusters of instances).

The following figure illustrates the components of the J2EE platform discussed in the following sections.



Containers

The J2EE application model divides enterprise applications into three fundamental parts: components, containers, and connectors. Components are the key focus of application developers, while system vendors implement containers and connectors to conceal complexity and promote portability. Containers intercede between clients and components, providing services transparently to both, including transaction support and resource pooling. Container mediation allows many component behaviors to be specified at deployment time, rather than in program code.

In the Application Server, each server instance includes two containers: web and EJB. A container is a runtime environment that provides services such as security and transaction management to J2EE components. Web components, such as Java Server Pages and servlets, run within the web container. Enterprise JavaBeans run within the EJB container.

J2EE Services

The J2EE platform services simplify application programming and allow components and applications to be customized at deployment time to use resources available in the deployment environment. This section gives a brief overview of the J2EE platform naming, deployment, transaction, and security services.The J2EE platform provides services for applications, including:

- *Naming* -A naming and directory service binds objects to names. A J2EE application can locate an object by looking up its Java Naming and Directory Interface (JNDI) name.
- *Security* - The Java Authorization Contract for Containers (JACC) is a set of security contracts defined for the J2EE containers. Based on the client's identity, containers can restrict access to the container's resources and services.
- **Transaction management** - A transaction is an indivisible unit of work. For example, transferring funds between bank accounts is a transaction. A transaction management service ensures that a transaction is either completed, or is rolled back.
- Message Service - Applications hosted on separate systems can communicate with each other by exchanging messages using the Java™ Message Service (JMS). JMS is an integral part of the J2EE platform and simplifies the task of integrating heterogeneous enterprise applications.

Web Services

Clients can access a J2EE 1.4 application as a remote web service in addition to accessing it through HTTP, RMI/IIOP, and JMS. Web services are implemented using the Java API for XML-based RPC (JAX-RPC). A J2EE application can also act as a client to web services, which would be typical in network applications.

Web Services Description Language (WSDL) is an XML format that describes web service interfaces. Web service consumers can dynamically parse a WSDL document to determine the operations a web service provides and how to execute them. The Application Server distributes web services interface descriptions using a registry that other applications can access through the Java API for XML Registries (JAXR).

Client Access

Clients can access J2EE applications in several ways. Browser clients access web applications using hypertext transfer protocol (HTTP). For secure communication, browsers use the HTTP secure (HTTPS) protocol that uses secure sockets layer (SSL).

Rich client applications running in the Application Client Container can directly lookup and access Enterprise JavaBeans using an Object Request Broker (ORB), Remote Method Invocation (RMI) and the internet inter-ORB protocol (IIOP), or IIOP/SSL (secure IIOP). They can access applications and web services using HTTP/HTTPS, JMS, and JAX-RPC. They can use JMS to send messages to and receive messages from applications and message-driven beans.

Clients that conform to the Web Services-Interoperability (WS-I) Basic Profile can access J2EE web services. WS-I is an integral part of the J2EE standard and defines interoperable web services. It enables clients written in any supporting language to access web services deployed to the Application Server.

The best access mechanism depends on the specific application and the anticipated volume of traffic. The Application Server supports separately configurable listeners for HTTP, HTTPS, JMS, IIOP, and IIOP/SSL. You can set up multiple listeners for each protocol for increased scalability and reliability.

J2EE applications can also act as clients of J2EE components such as Enterprise JavaBeans modules deployed on other servers, and can use any of these access mechanisms.

External Systems and Resources

On the J2EE platform, an external system is called a *resource*. For example, a database management system is a JDBC resource. Each resource is uniquely identified and by its Java Naming and Directory Interface (JNDI) name. Applications access external systems through the following APIs and components:

- *Java Database Connectivity (JDBC)* - A database management system (DBMS) provides facilities for storing, organizing, and retrieving data. Most business applications store data in relational databases, which applications access via JDBC. The Application Server includes the PointBase DBMS for use sample applications and application development and prototyping, though it is not suitable for deployment. The Application Server provides certified JDBC drivers for connecting to major relational databases. These drivers are suitable for deployment.
- *Java Message* Service - Messaging is a method of communication between software components or applications. A messaging client sends messages to, and receives messages from, any other client via a messaging provider that implements the Java Messaging Service (JMS) API. The Application Server includes a high-performance JMS broker, the Sun Java System Message Queue. The Platform Edition of Application Server includes the free

Platform Edition of Message Queue. Application Server Enterprise Edition includes Message Queue Enterprise Edition, that supports clustering and failover.

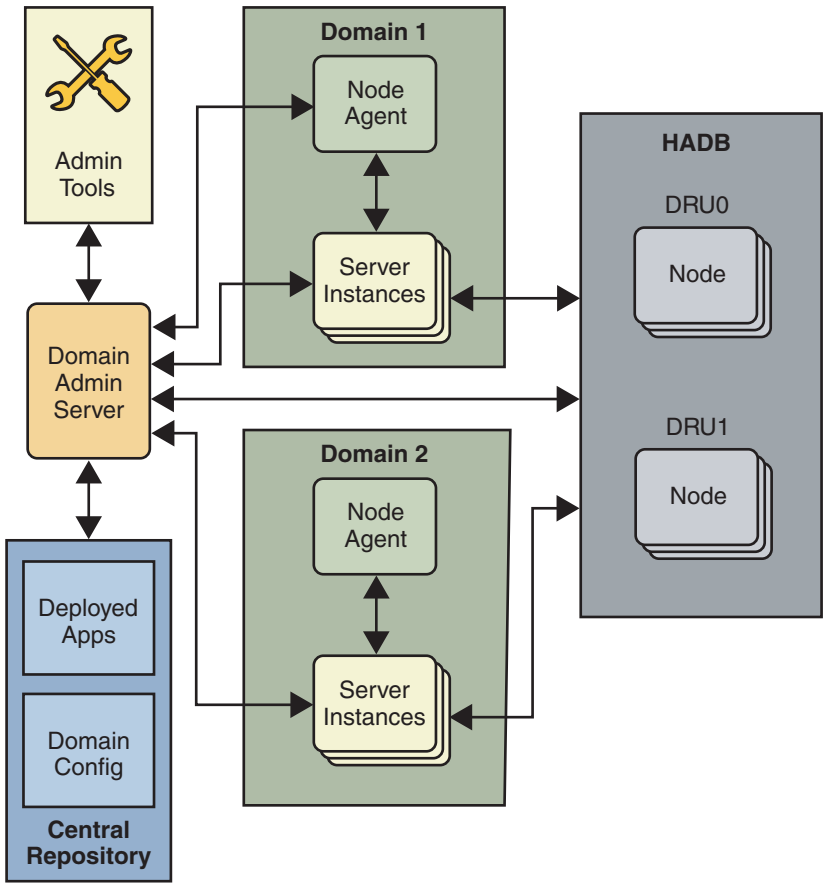
- **J2EE Connectors** - The J2EE Connector architecture enables integrating J2EE applications and existing Enterprise Information Systems (EIS). An application accesses an EIS through a portable J2EE component called a *connector* or *resource adapter*, analogous to using JDBC driver to access an RDBMS. Resource adapters are distributed as standalone Resource Adapter Archive (RAR) modules or included in J2EE application archives. As RARs, they are deployed like other J2EE components. The Application Server includes evaluation resource adapters that integrate with popular EIS.
- **JavaMail** - Through the JavaMail API, applications can connect to a Simple Mail Transport Protocol (SMTP) server to send and receive email.

Application Server Components

This section describes the components in the Sun Java System Application Server:

The following figure illustrates how these Application Server components interact using a simple example topology that provides high availability. In this example topology, one administrator manages two machines organized as a cluster. HADB and Application server processes are located on the same machine. The Domain Administration Server may be hosted on a separate machine by itself or on any one of the machines hosting application server instances. The lines in the diagram indicate communication or control.

The administration tools, such as the browser-based Admin Console, communicate with the domain administration server (DAS), which in turn communicates with the node agents and server instances.



Server Instances

A server instance is a Application Server running in a single Java Virtual Machine (JVM) process. The Application Server is certified with Java 2 Standard Edition (J2SE) 5.0 and 1.4. The recommended J2SE distribution is included with the Application Server installation.

It is usually sufficient to create a single server instance on a machine, since the Application Server and accompanying JVM are both designed to scale to multiple processors. However, it can be beneficial to create multiple instances on one machine for application isolation and rolling upgrades. In some cases, a large server with multiple instances can be used in more than admiistrative domain. The administration tools make it easy to create, delete and manage server instances across multiple machines.

Administrative Domains

An *administrative domain* (or simply *domain*) is a group of server instances that are administered together. A server instance belongs to a single administrative domain. The instances in a domain can run on different physical hosts.

You can create multiple domains from one installation of the Application Server. By grouping server instances into domains, different organizations and administrators can share a single Application Server installation. Each domain has its own configuration, log files, and application deployment areas that are independent of other domains. Changing the configuration of one domain does not affect the configurations of other domains. Likewise, deploying an application on a one domain does not deploy it or make it visible to any other domain. At any given time, an administrator can be authenticated to only one domain, and thus can only perform administration on that domain.

Domain Administration Server (DAS)

A domain has one Domain Administration Server (DAS), a specially-designated application server instance that hosts the administrative applications. The DAS authenticates the administrator, accepts requests from administration tools, and communicates with server instances in the domain to carry out the requests.

The administration tools are the `asadmin` command-line utility and the browser-based Admin Console. The Application Server also provides a JMX-based API for server administration. The administrator can view and manage a single domain at a time, thus enforcing secure separation.

The DAS is also sometimes referred to as the *admin server* or *default server*. It is referred to as the default server because it is the default target for some administrative operations.

Since the DAS is an application server instance, it can also host J2EE applications for testing purposes. However, do not use it to host production applications. You might want to deploy applications to the DAS, for example, if the clusters and instances that will host the production application have not yet been created.

The DAS keeps a repository containing the configuration of each domain and all the deployed applications. If the DAS is inactive or down, there is no impact on the performance or availability of active server instances, however administrative changes cannot be made. In certain cases, for security purposes, it may be useful to intentionally stop the DAS process; for example to freeze a production configuration.

Administrative commands are provided to backup and restore domain configuration and applications. With the standard backup and restore procedures, you can quickly restore working configurations. If the DAS host fails, you must create a new DAS installation to restore the previous domain configuration. For instructions, see "Recreating the Domain Administration Server" in *Sun Java System Application Server Enterprise Edition 8.2 Administration Guide*.

Sun Cluster Data Services provides high availability of the DAS through failover of the DAS host IP address and use of the Global File System. This solution provides nearly continuous availability for DAS and the repository against many types of failures. Sun Cluster Data Services are available with the Sun Java Enterprise System or purchased separately with Sun Cluster. For more information, see the documentation for Sun Cluster Data Services.

Clusters

A cluster is a named collection of server instances that share the same applications, resources, and configuration information. You can group server instances on different machines into one logical cluster and administer them as one unit. You can easily control the lifecycle of a multi-machine cluster with the DAS.

Clusters enable horizontal scalability, load balancing, and failover protection. By definition, all the instances in a cluster have the same resource and application configuration. When a server instance or a machine in a cluster fails, the load balancer detects the failure, redirects traffic from the failed instance to other instances in the cluster, and recovers the user session state. Since the same applications and resources are on all instances in the cluster, an instance can failover to any other instance in the cluster.

Clusters, domains, and instances are related as follows:

- An administrative domain can have zero or more clusters.
- A cluster has one or more server instances.
- A cluster belongs to a single domain

Node Agents

A node agent is a lightweight process that runs on every machine that hosts server instances, including the machine that hosts the DAS. The node agent:

- Starts and stops server instances as instructed by the DAS.
- Restarts failed server instances.
- Provides a view of the log files of failed servers and assists in remote diagnosis
- Synchronizes the local configuration repository of each server instance with the DAS's central repository. For details, see Chapter 8, "Configuring Node Agents," in *Sun Java System Application Server Enterprise Edition 8.2 High Availability Administration Guide*.
- When an instance is initially created, creates directories the instance needs and synchronizes the instance's configuration with the central repository.
- Performs appropriate cleanup when a server instance is deleted.

Each physical host must have at least one node agent for each domain to which the host belongs. If a physical host has instances from more than one domain, then it needs a node agent for each domain. There is no a

Because a node agent starts and stops server instances, it must always be running. Therefore, it is started when the operating system boots up. On Solaris and other Unix platforms, the node agent can be started by the `inetd` process. On Windows, the node agent can be made a Windows service.

For more information on node agents, see Chapter 8, "Configuring Node Agents," in *Sun Java System Application Server Enterprise Edition 8.2 High Availability Administration Guide*.

Named Configurations

A *named configuration*is an abstraction that encapsulates Application Server property settings. Clusters and standalone server instances reference a named configuration to get their property settings. With named configurations, J2EE containers' configurations are independent of the physical machine on which they reside, except for particulars such as IP address, port number, and amount of heap memory. Using named configurations provides power and flexibility to Application Server administration.

To apply configuration changes, you change the property settings of the named configuration, and all the clusters and standalone instances that reference it pick up the changes. You can only delete a named configuration when all references to it have been removed. A domain can contain multiple named configurations.

The Application Server comes with a default configuration, called `default-config`. The default configuration is optimized for developer productivity in Application Server Platform Edition and for security and high availability in the Application Server Standard and Enterprise Editions.

You can create your own named configuration based on the default configuration that you can customize for your own purposes. Use the Admin Console and `asadmin` command line utility to create and manage named configurations.

HTTP Load Balancer Plug-in

The load balancer distributes the workload among multiple physical machines, thereby increasing the overall throughput of the system. The Application Server Enterprise Edition includes the load balancer plug-in for the Sun Java System Web Server, the Apache Web Server, and Microsoft Internet Information Server.

The load balancer plug-in accepts HTTP and HTTPS requests and forwards them to one of the application server instances in the cluster. Should an instance fail, become unavailable (due to network faults), or become unresponsive, requests are redirected to existing, available machines. The load balancer can also recognize when a failed instance has recovered and redistribute the load accordingly.

For simple stateless applications, a load-balanced cluster may be sufficient. However, for mission-critical applications with session state, use load balanced clusters with HADB.

To setup a system with load balancing, in addition to the Application Server, you must install a web server and the load-balancer plug-in. Then you must:

- Create Application Server clusters that you want to participate in load balancing.
- Deploy applications to these load-balanced clusters.

Server instances and clusters participating in load balancing have a homogenous environment. Usually that means that the server instances reference the same server configuration, can access the same physical resources, and have the same applications deployed to them. Homogeneity assures that before and after failures, the load balancer always distributes load evenly across the active instances in the cluster.

Use the `asadmin` command-line tool to create a load balancer configuration, add references to clusters and server instances to it, enable the clusters for reference by the load balancer, enable applications for load balancing, optionally create a health checker, generate the load balancer configuration file, and finally copy the load balancer configuration file to your web server `config` directory. An administrator can create a script to automate this entire process.

For more details and complete configuration instructions, see Chapter 5, "Configuring HTTP Load Balancing," in *Sun Java System Application Server Enterprise Edition 8.2 High Availability Administration Guide*.

Session Persistence

J2EE applications typically have significant amounts of session state data. A web shopping cart is the classic example of a session state. Also, an application can cache frequently-needed data in the session object. In fact, almost all applications with significant user interactions need to maintain a session state. Both HTTP sessions and stateful session beans (SFSBs) have session state data.

While the session state is not as important as the transactional state stored in a database, preserving the session state across server failures can be important to end users.The Application Server provides the capability to save, or persist, this session state in a repository. If the application server instance that is hosting the user session experiences a failure, the session state can be recovered. The session can continue without loss of information.

The Application Server supports the following types of session persistence stores:

- Memory
- High availability (HA)
- File

With memory persistence, the state is always kept in memory and does not survive failure. With HA persistence, the Application Server uses HADB as the persistence store for both HTTP and SFSB sessions. With file persistence, the Application Server serializes session objects and stores them to the file system location specified by session manager properties. For SFSBs, if HA is not specified, the Application Server stores state information in the session-store sub-directory of this location.

Checking an SFSB's state for changes that need to be saved is called *checkpointing*. When enabled, checkpointing generally occurs after any transaction involving the SFSB is completed, even if the transaction rolls back. For more information on developing stateful session beans, see "Using Session Beans" in *Sun Java System Application Server Enterprise Edition 8.2 Developer's Guide*. For more information on enabling SFSB failover, see "Stateful Session Bean Failover" in *Sun Java System Application Server Enterprise Edition 8.2 High Availability Administration Guide*.

Apart from the number of requests being served by the Application Server, the session persistence configuration settings also affect the number of requests received per minute by the HADB, as well as the session information in each request.

For more information on configuring session persistence, see Chapter 9, "Configuring High Availability Session Persistence and Failover," in *Sun Java System Application Server Enterprise Edition 8.2 High Availability Administration Guide*.

IIOP Load Balancing in a Cluster

With IIOP load balancing, IIOP client requests are distributed to different server instances or name servers. The goal is to spread the load evenly across the cluster, thus providing scalability. IIOP load balancing combined with EJB clustering and availability features in the Sun Java System Application provides not only load balancing but also EJB failover.

When a client performs a JNDI lookup for an object, the Naming Service creates a `InitialContext` (IC) object associated with a particular server instance. From then on, all lookup requests made using that IC object are sent to the same server instance. All `EJBHome` objects looked up with that `InitialContext` are hosted on the same target server. Any bean references obtained henceforth are also created on the same target host. This effectively provides load balancing, since all clients randomize the list of live target servers when creating `InitialContext` objects. If the target server instance goes down, the lookup or EJB method invocation will failover to another server instance.

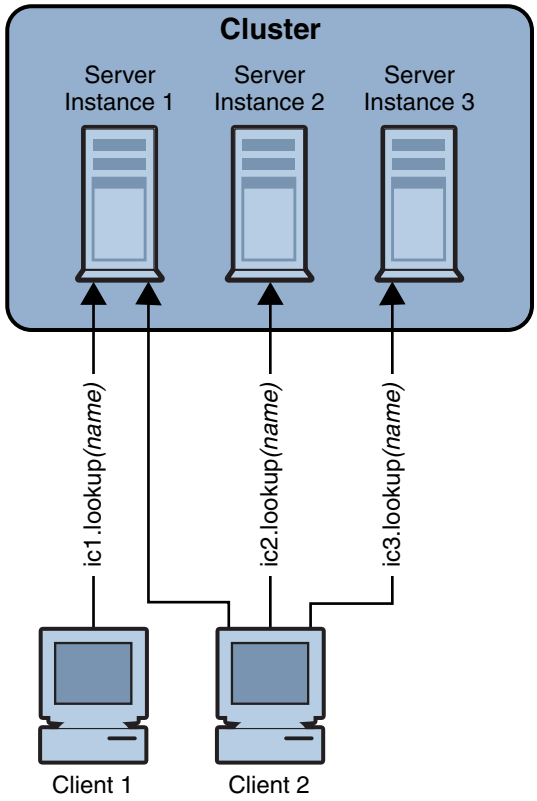


FIGURE 1–1 IIOP Load Balancing in a Cluster

For example, as illustrated in this figure, ic1, ic2, and ic3 are three different `InitialContext` instances created in Client2's code. They are distributed to the three server instances in the cluster. Enterprise JavaBeans created by this client are thus spread over the three instances. Client1 created only one InitialContext object and the bean references from this client are only on Server Instance 1. If Server Instance 2 goes down, the lookup request on ic2 will failover to another server instance (not necessarily Server Instance 3). Any bean method invocations to beans previously hosted on Server Instance 2 will also be automatically redirected, if it is safe to do so, to another instance. While lookup failover is automatic, Enterprise JavaBeans modules will retry method calls only when it is safe to do so.

IIOP Load balancing and failover happens transparently. No special steps are needed during application deployment. Adding or deleting new instances to the cluster will not update the existing client's view of the cluster. You must manually update the endpoints list on the client side.

Message Queue and JMS Resources

The Sun Java System Message Queue (MQ) provides reliable, asynchronous messaging for distributed applications. MQ is an enterprise messaging system that implements the Java Message Service (JMS) standard. MQ provides messaging for J2EE application components such as message-driven beans (MDBs).

The Application Server implements the Java Message Service (JMS) API by integrating the Sun Java System Message Queue into the Application Server. The Enterprise Edition of the Application Server includes the Enterprise version of MQ which has failover, clustering and load balancing features.

For basic JMS administration tasks, use the Application Server Admin Console and `asadmin` command-line utility.

For advanced tasks, including administering a Message Queue cluster, use the tools provided in the *install_dir*`/imq/bin` directory. For details about administering Message Queue, see the *Sun Java System Message Queue Administration Guide*.

For information on deploying JMS applications and MQ clustering for message failover, see "Planning Message Queue Broker Deployment" on page 62.

High-Availability Database

This section discusses the following topics:

J2EE applications' need for session persistence was previously described in "Session Persistence" on page 29. The Application Server uses the high-availability database (HADB) as a highly available session store. HADB is included with the Application Server Enterprise Edition, but in deployment can be run on seprate hosts. HADB provides a highly available data store for HTTP session and stateful session bean data.

The advantages of this decoupled architecture include:

- Server instances in a highly available cluster are loosely coupled and act as high performance J2EE containers.
- Stopping and starting server instances does not affect other servers or their availability.

- HADB can run on a different set of less expensive machines (for example, with single or dual processors). Several clusters can share these machines. Depending upon the deployment needs, you can run HADB on the same machines as Application Server (co-located) or different machines (separate tier). For more information on the two options, see "Co-located Topology" on page 71
- As state management requirements change, you can add resources to the HADB system without affecting existing clusters or their applications.

Note – HADB is optimized for use by Application Server and is not meant to be used by applications as a general purpose database.

HADB System Requirements

The system requirements for HADB hosts are:

- At least one CPU per HADB node.
- At least 512 MB memory per node

For network configuration requirements, see Chapter 2, "Installing and Setting Up High Availability Database," in *Sun Java System Application Server Enterprise Edition 8.2 High Availability Administration Guide*. For additional requirements for very high availability, see "Mitigating Double Failures" on page 37.

HADB Architecture

HADB is a distributed system consisting of pairs of *nodes*. Nodes are divided into two data redundancy units (DRUs), with a node from each pair in each DRU, as illustrated in "Data Redundancy Units" on page 34.

Each node consists of:

- A set of processes for transactional state replication
- A dedicated area of shared memory used for communication among the processes.
- One or more secondary storage devices (disks).

A set of HADB nodes can host one or more *session databases*. Each session database is associated with a distinct application server cluster. Deleting a cluster also deletes the associated session database.

Nodes and Node Processes

There are two types of HADB nodes:

- *Active nodes* that store data.

- *Spare nodes* that do not contain any data initially, but perform as active nodes if an active node becomes unavailable. Spare nodes are optional but useful for achieving higher availability.

Each node has a parent process and several child processes. The parent process, called the node supervisor (NSUP), is started by the management agent. It is responsible for creating the child processes and keeping them running.

The child processes are:

- Transaction server process (TRANS), that coordinates transactions on distributed nodes, and manages data storage.
- Relational algebra server process (RELALG) that coordinates and executes complex relational algebra queries such as sorts and and joins.
- SQL shared memory server process (SQLSHM) that maintains the SQL dictionary cache.
- SQL server process (SQLC), that receives client queries, compiles them into local HADB instructions, sends the instructions to TRANS, receives the results and conveys them to the client. Each node has one main SQL server and one sub-server for each client connection.
- Node manager server process (NOMAN) that management agents use to execute management commands issued by the hadbm management client.

Data Redundancy Units

As previously described, an HADB instance contains a pair of DRUs. Each DRU has the same number of active and spare nodes as the other DRU in the pair. Each active node in a DRU has a *mirror node* in the other DRU. Due to mirroring, each DRU contains a complete copy of the database.

The following figure shows an example HADB architecture with six nodes: four active nodes and two spare nodes. Nodes 0 and 1 are a mirror pair, as are nodes 2 and 3. In this example, each host has one node. In general, a host can have more than one node if it has sufficient system resources (see "HADB System Requirements" on page 33).

Note – You must add machines that host HADB nodes in pairs, with one machine in each DRU.

HADB achieves high availability by replicating data and services. The data replicas on mirror

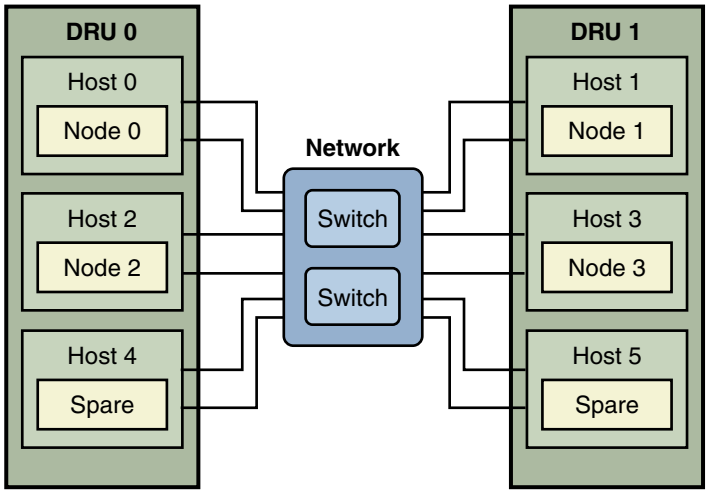


FIGURE 1–2 Sample HADB Configuration with Double Interconnects

nodes are designated as *primary replicas* and *hot standby replicas*. The primary replica performs operations such as inserts, deletes, updates, and reads. The hot standby replica receives log records of the primary replica's operations and redoes them within the transaction life time. Read operations are performed only by the primary node and thus not logged. Each node contains both primary and hot standby replicas and plays both roles. The database is fragmented and distributed over the active nodes in a DRU. Each node in a mirror pair contains the same set of data fragments. Duplicating data on a mirror node is known as *replication*. Replication enables HADB to provide high availability: when a node fails, its mirror node takes over almost immediately (within seconds). Replication ensures availability and masks node failures or DRU failures without loss of data or services.

When a mirror node takes over the functions of a failed node, it has to perform double work: its own and that of the failed node. If the mirror node does not have sufficient resources, the overload will reduce its performance and increase its failure probability. When a node fails, HADB attempts to restart it. If the failed node does not restart (for example, due to hardware failure), the system continues to operate but with reduced availability.

HADB tolerates failure of a node, an entire DRU, or multiple nodes, but not a "double failure" when both a node and its mirror fail. For information on how to reduce the likelihood of a double failure, see "Mitigating Double Failures" on page 37

Spare Nodes

When a node fails, its mirror node takes over for it. If the failed node does not have a spare node, then at this point, the failed node will not have a mirror. A spare node will automatically replace a failed node's mirror. Having a spare node reduces the time the system functions without a mirror node.

A spare node does not normally contain data, but constantly monitors for failure of active nodes in the DRU. When a node fails and does not recover within a specified timeout period, the spare node copies data from the mirror node and synchronizes with it. The time this takes depends on the amount of data copied and the system and network capacity. After synchronizing, the spare node automatically replaces the mirror node without manual intervention, thus relieving the mirror node from overload, thus balancing load on the mirrors. This is known as *failback* or *self-healing*.

When a failed host is repaired (by shifting the hardware or upgrading the software) and restarted, the node or nodes running on it join the system as a spare nodes, since the original spare nodes are now active.

Spare nodes are not required, but they enable a system to maintain its overall level of service even if a machine fails. Spare nodes also make it easy to perform planned maintenance on machines hosting active nodes. Allocate one machine for each DRU to act as a spare machine, so that if one of the machines fails, the HADB system continues without adversely affecting performance and availability.

Note – As a general rule, have a spare machine with enough Application Server instances and HADB nodes to replace any machine that becomes unavailable.

Example Spare Node Configurations

The following examples illustrate using spare nodes in HADB deployments. There are two possible deployment topologies: *co-located*, in which HADB and Application Servers reside on the same hosts, and *separate tier* , in which they reside on separate hosts. For more information on deployment topologies, see Chapter 3, “Selecting a Topology”

Example: co-located configuration

As an example of a spare node configuration, suppose you have a co-located topology with four Sun Fire™ V480 servers, where each server has one Application Server instance and two HADB data nodes.

For spare nodes, allocate two more servers (one machine per DRU). Each spare machine runs one application server instance and two spare HADB nodes.

Example: separate tier configuration

Suppose you have a separate-tier topology where the HADB tier has two Sun Fire™ 280R servers, each running two HADB data nodes. To maintain this system at full capacity, even if one machine becomes unavailable, configure one spare machine for the Application Server instances tier and one spare machine for the HADB tier.

The spare machine for the Application Server instances tier must have as many instances as the other machines in the Application Server instances tier. Similarly, the spare machine for the HADB tier must have as many HADB nodes as the other machines in the HADB tier.

Mitigating Double Failures

HADB's built-in data replication enables it to tolerate failure of a single node or an entire DRU. By default, HADB won't survive a *double failure* , when a mirror node pair or both DRUs fail. In such cases, HADB become unavailable.

In addition to using spare nodes as described in the previous section, you can minimize the likelihood of a double failure by taking the following steps:

- **Providing independent power supplies**: For optimum fault tolerance, the servers that support one DRU must have independent power (through uninterruptible power supplies), processing units, and storage. If a power failure occurs in one DRU, the nodes in the other DRU continue servicing requests until the power returns.
- **Providing double interconnections**: To tolerate single network failures, replicate the lines and switches between DRUs as shown in Figure 1–2.

These steps are optional, but will increase the overall availability of the HADB instance.

HADB Management System

The HADB management system provides built-in security and facilitates multi-platform management. As illustrated in the following figure, the HADB management architecture contains the following components:

- "Management Client" on page 38
- "Management Agent" on page 39
- "Management Domains" on page 39
- "Repository" on page 40

As shown in the figure, one HADB management agent runs on every machine that runs the HADB service. Each machine typically hosts one or more HADB nodes. An HADB management domain contains many machines, similar to an Application Server domain. At least two machines are required in a domain for the database to be fault tolerant, and in genera there must be an even number of machines to form the DRU pairs. Thus, a domain contains many management agents.

As shown in the figure, a domain can contain one or more database instances. One machine can contain one or more nodes belonging to one or more database instances.

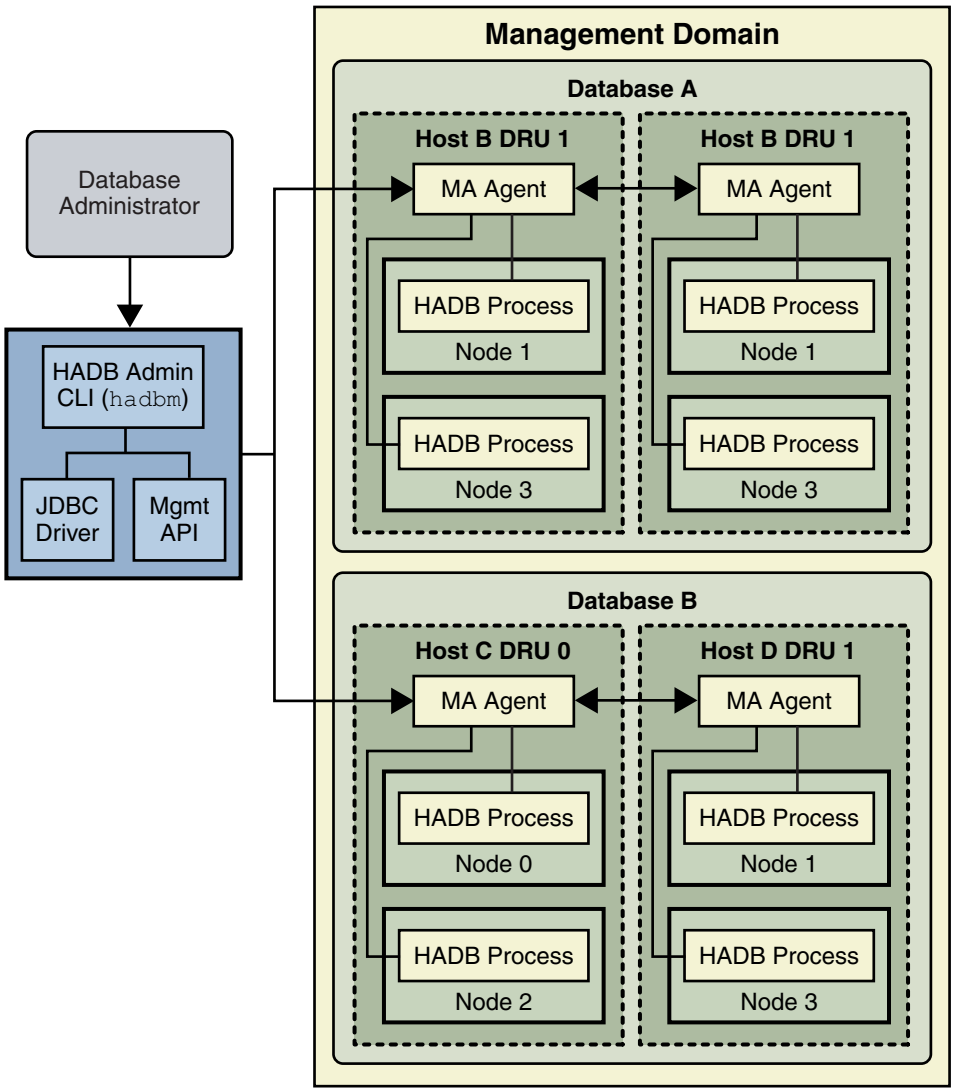


FIGURE 1–3 HADB Management Architecture

Management Client

The HADB *management client* is a command-line utility, `hadbm`, for managing the HADB domain and its database instances. HADB services can run continously— even when the associated Application Server cluster is stopped—but must be shut down carefully if they are to be deleted. For more information on using `hadbm`, see Chapter 3, "Administering High Availability Database," in *Sun Java System Application Server Enterprise Edition 8.2 High Availability Administration Guide*.

You can use the asadmin command line utility to create and delete the HADB instance associated with a highly available cluster. For more information, see Chapter 9, "Configuring High Availability Session Persistence and Failover," in *Sun Java System Application Server Enterprise Edition 8.2 High Availability Administration Guide*.

Management Agent

The management agent is a server process (named `ma`) that can access resources on a host; for example, it can create devices and start database processes. The management agent coordinates and performs management client commands such as starting or stopping a database instance.

A management client connects to a management agent by specifying the address and port number of the agent. Once connected, the management client sends commands to HADB through the management agent. The agent receives requests and executes them. Thus, a management agent must be running on a host before issuing any `hadbm` management commands to that host. The management agent can be configured as a system service that starts up automatically.

Ensuring availability of management agents

The management agent process ensures the availability of the HADB node supervisor processes by restarting them if they fail. Thus, for deployment, you must ensure the availability of the `ma` process to maintain the overall availability of HADB. After restarting, the management agent recovers the domain and database configuration data from other agents in the domain.the system.

Use the host operating system (OS) to ensure the availability of the management agent. On Solaris or Linux, `init.d` ensures the availability of the `ma` process after a process failure and reboot of the operating system. On Windows, the management agent runs as a Windows service. Thus, the OS restarts the management agent if the agent fails or the OS reboots.

Management Domains

An HADB management domain is a set of hosts, each of which has a management agent running on the same port number. The hosts in a domain can contain one or more HADB database instances. A management domain is defined by the common port number the agents use and an identifier (called a *domainkey*) generated when you create or the domain or add an agent to it. The *domainkey* provides a unique identifier for the domain, crucial since management agents communicate using multicast. You can set up an HADB management domain to match with an Application Server domain.

Having multiple database instances in one domain can be useful in a development environment, since it enables different developer groups to use their own database instance. In some cases, it may also be useful in production environments.

All agents belonging to a domain coordinate their management operations. When you change the database configuration through an `hadbm` command, all agents will change the configuration accordingly. You cannot stop or restart a node unless the management agent on the node's host is running. However, you can execute `hadbm` commands that read HADB state or configuration variable values even if some agents are not available.

Use the following management client commands to work with management domains:

- **hadbm createdomain**: creates a management domain with the specified hosts. For more information, see `hadbm-createdomain`(1).
- **hadbm extenddomain**: adds hosts to an existing management domain. For more information, see `hadbm-extenddomain`(1).
- **hadbm deletedomain**: removes a management domain. For more information, see `hadbm-deletedomain`(1).
- **hadbm reducedomain**: removes hosts from the management domain. For more information, see `hadbm-reducedomain`(1).
- **hadbm listdomain**: lists all hosts defined in the management domain. For more information, see `hadbm-listdomain`(1).

Repository

Management agents store the database configuration in a *repository*. The repository is highly fault-tolerant, because it is replicated over all the management agents. Keeping the configuration on the server enables you to perform management operations from any computer that has a management client installed.

A majority of the management agents in a domain must be running to perform any changes to the repository. Thus, if there are M agents in a domain, at least M/2 + 1 agents (rounded down to a whole number) must be running to make a change to the repository.

If some of the hosts in a domain are unavailable, for example due to hardware failures, and you cannot perform some management commands because you don't have a quorum, use the `hadbm disablehost` command to remove the failed hosts from the domain. For more information on this command, see `hadbm-disablehost`(1).

Setup and Configuration Roadmap

▼ To Set Up and Configure Application Server for High Availability

1 Determine your performance and QoS requirements and goals, as described in Chapter 1, "Product Concepts"

2. **Size your system, as described in "Design Decisions" on page 58 in Chapter 1, "Product Concepts"**

   - Number of Application Server Instances
   - Number of HADB Nodes and Hosts
   - HADB Storage Capacity

3. **Determine system topology, as described in Chapter 3, "Selecting a Topology."**

   This determines whether you are going to intall HADB on the same host machines as Application Server or on different machines.

4. **Install Application Server instances, along with related subcomponents such as HADB and a web server.**

5. **Create domains and clusters.**

6. **Configure your web server software.**

7. **Install the Load Balancer Plug-in.**

8. **Setup and configure load balancing.**

9. **Setup and configure HADB nodes and DRUs.**

10. **Configure AS Web container and EJB container for HA session persistence.**

11. **Deploy applications and configure them for high availability and session failover.**

12. **Configure JMS clusters for failover if you are using messaging extensively.**

    For more information, see Chapter 9, "Working With Broker Clusters," in *Sun Java System Message Queue 3.7 UR1 Administration Guide*

◆ ◆ ◆ CHAPTER 2

Planning your Deployment

Before deploying the Application Server, first determine the performance and availability goals, and then make decisions about the hardware, network, and storage requirements accordingly.

This chapter contains the following sections:

Establishing Performance Goals

At its simplest, high performance means maximizing throughput and reducing response time. Beyond these basic goals, you can establish specific goals by determining the following:

- What types of applications and services are deployed, and how do clients access them?
- Which applications and services need to be highly available?
- Do the applications have session state or are they stateless?
- What request capacity or throughput must the system support?
- How many concurrent users must the system support?
- What is an acceptable average response time for user requests?
- What is the average think time between requests?

You can calculate some of these metrics using a remote browser emulator (RBE) tool, or web site performance and benchmarking software that simulates expected application activity. Typically, RBE and benchmarking products generate concurrent HTTP requests and then report the response time for a given number of requests per minute. You can then use these figures to calculate server activity.

The results of the calculations described in this chapter are not absolute. Treat them as reference points to work against, as you fine-tune the performance of the Application Server and your applications.

This section discusses the following topics:

Estimating Throughput

In broad terms, *throughput* measures the amount of work performed by Application Server. For Application Server, throughput can be defined as the number of requests processed per minute per server instance. High availability applications also impose throughput requirements on HADB, since they save session state data periodically. For HADB, throughput can be defined as volume of session data stored per minute, which is the product of the number of HADB requests per minute, and the average session size per request.

As described in the next section, Application Server throughput is a function of many factors, including the nature and size of user requests, number of users, and performance of Application Server instances and back-end databases. You can estimate throughput on a single machine by benchmarking with simulated workloads.

High availability applications incur additional overhead because they periodically save data to HADB. The amount of overhead depends on the amount of data, how frequently it changes, and how often it is saved. The first two factors depend on the application in question; the latter is also affected by server settings.

HADB throughput can be defined as the number of HADB requests per minute multiplied by the average amount of data per request. Larger throughput to HADB implies that more HADB nodes are needed and a larger store size.

Estimating Load on Application Server Instances

Consider the following factors to estimate the load on Application Server instances:

Maximum Number of Concurrent Users

Users interact with an application through a client, such as a web browser or Java program. Based on the user's actions, the client periodically sends requests to the Application Server. A user is considered *active* as long as the user's session has neither expired nor been terminated. When estimating the number of concurrent users, include all active users.

The following figure illustrates a typical graph of requests processed per minute (throughput) versus number of users. Initially, as the number of users increases, throughput increases correspondingly. However, as the number of concurrent requests increases, server performance begins to saturate, and throughput begins to decline.

Identify the point at which adding concurrent users reduces the number of requests that can be processed per minute. This point indicates when optimal performance is reached and beyond which throughput start to degrade. Generally, strive to operate the system at optimal throughput as much as possible. You might need to add processing power to handle additional load and increase throughput.

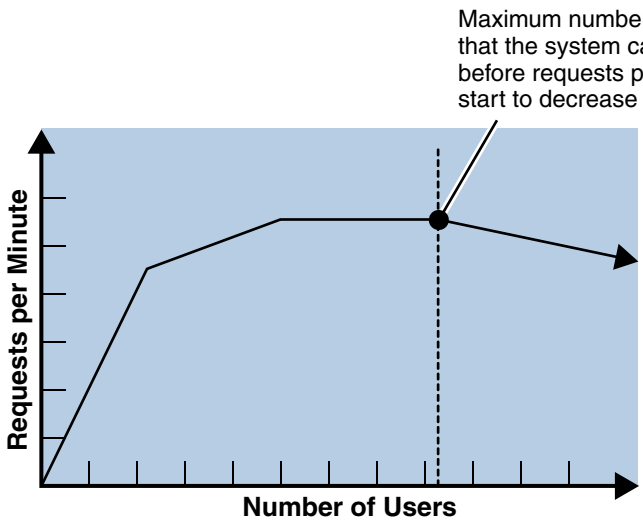


FIGURE 2–1 Typical Profile of Throughput Versus Concurrent Users

Think Time

A user does not submit requests continuously. A user submits a request, the server receives and processes the request, and then returns a result, at which point the user spends some time before submitting a new request. The time between one request and the next is called *think time*.

Think times are dependent on the type of users. For example, machine-to-machine interaction such as for a web service typically has a lower think time than that of a human user. You may have to consider a mix of machine and human interactions to estimate think time.

Determining the average think time is important. You can use this duration to calculate the number of requests that need to be completed per minute, as well as the number of concurrent users the system can support.

Average Response Time

Response time refers to the amount of time Application Server takes to return the results of a request to the user. The response time is affected by factors such as network bandwidth, number of users, number and type of requests submitted, and average think time.

In this section, response time refers to the mean, or average, response time. Each type of request has its own minimal response time. However, when evaluating system performance, base the analysis on the average response time of all requests.

The faster the response time, the more requests per minute are being processed. However, as the number of users on the system increases, the response time starts to increase as well, even though the number of requests per minute declines, as the following diagram illustrates:

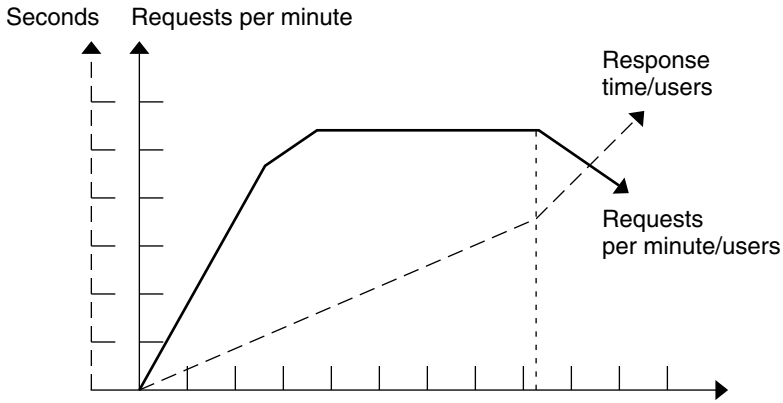


FIGURE 2–2 Response Time with Increasing Number of Users

A system performance graph similar to this figure indicates that after a certain point, requests per minute are inversely proportional to response time. The sharper the decline in requests per minute, the steeper the increase in response time (represented by the dotted line arrow).

In the figure, the point of the peak load is the point at which requests per minute start to decline. Prior to this point, response time calculations are not necessarily accurate because they do not use peak numbers in the formula. After this point, (because of the inversely proportional relationship between requests per minute and response time), the administrator can more accurately calculate response time using maximum number of users and requests per minute.

Use the following formula to determine $T_{response}$, the response time (in seconds) at peak load:

$$T_{response} = n/r - T_{think}$$

where

- n is the number of concurrent users
- r is the number requests per second the server receives
- T_{think} is the average think time (in seconds)

  To obtain an accurate response time result, always include think time in the equation.

EXAMPLE 2–1 Calculation of Response Time

If the following conditions exist:

- Maximum number of concurrent users, n, that the system can support at peak load is 5,000.
- Maximum number of requests, r, the system can process at peak load is 1,000 per second.

Average think time, T_{think}, is three seconds per request.

Thus, the calculation of response time is:

$$T_{response} = n/r - T_{think} = (5000/ 1000) - 3 \text{ sec.} = 5 - 3 \text{ sec.}$$

Therefore, the response time is two seconds.

After the system's response time has been calculated, particularly at peak load, compare it to the acceptable response time for the application. Response time, along with throughput, is one of the main factors critical to the Application Server performance.

Requests Per Minute

If you know the number of concurrent users at any given time, the response time of their requests, and the average user think time, then you can calculate the number of requests per minute. Typically, start by estimating the number of concurrent users that are on the system.

For example, after running web site performance software, the administrator concludes that the average number of concurrent users submitting requests on an online banking web site is 3,000. This number depends on the number of users who have signed up to be members of the online bank, their banking transaction behavior, the time of the day or week they choose to submit requests, and so on.

Therefore, knowing this information enables you to use the requests per minute formula described in this section to calculate how many requests per minute your system can handle for this user base. Since requests per minute and response time become inversely proportional at peak load, decide if fewer requests per minute is acceptable as a trade-off for better response time, or alternatively, if a slower response time is acceptable as a trade-off for more requests per minute.

Experiment with the requests per minute and response time thresholds that are acceptable as a starting point for fine-tuning system performance. Thereafter, decide which areas of the system require adjustment.

Solving for *r* in the equation in the previous section gives:

$r = n/(T_{response} + T_{think})$

EXAMPLE 2–2 Calculation of Requests Per Second

For the values:

- $n = 2{,}800$ concurrent users
- $T_{response} = 1$ (one second per request average response time)
- $T_{think} = 3$, (three seconds average think time)

The calculation for the number of requests per second is:

```
r = 2800 / (1+3) = 700
```

Therefore, the number of requests per second is 700 and the number of requests per minute is 42000.

Estimating Load on the HADB

To calculate load on the HADB, consider the following factors:

- "HTTP Session Persistence Frequency" on page 49
- "HTTP Session Size and Scope" on page 50
- "Stateful Session Bean Checkpointing" on page 51

For instructions on configuring session persistence, see Chapter 9, "Configuring High Availability Session Persistence and Failover," in *Sun Java System Application Server Enterprise Edition 8.2 High Availability Administration Guide*.

HTTP Session Persistence Frequency

The number of requests per minute received by the HADB depends on the *persistence frequency*. Persistence Frequency determines how often Application Server saves HTTP session data to the HADB.

The persistence frequency options are:

- **web-method** (default): the server stores session data with every HTTP response. This option guarantees that stored session information will be up to date, but leads to high traffic to HADB.
- **time-based**: the session is stored at the specified time interval. This option reduces the traffic to HADB, but does not guarantee that the session information will be up to date.

The following table summarizes the advantages and disadvantages of persistence frequency options.

TABLE 2–1 Comparison of Persistence Frequency Options

Persistence Frequency Option	Advantages	Disadvantages
web-method	Guarantees that the most up-to-date session information is available.	Potentially increased response time and reduced throughput.
time-based	Better response time and potentially better throughput.	Less guarantee that the most updated session information is available after the failure of an application server instance.

HTTP Session Size and Scope

The session size per request depends on the amount of session information stored in the session.

Tip – To improve overall performance, reduce the amount of information in the session as much as possible.

It is possible to fine-tune the session size per request through the `persistence scope` settings. Choose from the following options for HTTP session persistence scope:

- **session**: The server serializes and saves the entire session object every time it saves session information to HADB.
- **modified-session**: The server saves the session only if the session has been modified. It detects modification by intercepting calls to the bean's `setAttribute()` method. This option will not detect direct modifications to inner objects, so in such cases the SFSB must be coded to call `setAttribute()` explicitly.
- **modified-attribute**: The server saves only those attributes that have been modified (inserted, updated, or deleted) since the last time the session was stored. This has the same drawback as modified-session but can significantly reduce HADB write throughput requirements if properly applied.

To use this option, the application must:

- Call `setAttribute()` or `removeAttribute()` every time it modifies session state.
- Make sure there are no cross references between attributes.
- Distribute the session state across multiple attributes, or at least between a read-only attribute and a modifiable attribute.

  The following table summarizes the advantages and disadvantages of the persistence scope options.

TABLE 2–2 Comparison of Persistence Scope Options

Persistence Scope Option	Advantages	Disadvantages
modified-session	Provides improved response time for requests that do not modify session state.	During the execution of a web method, typically `doGet()` or `doPost()`, the application must call a session method: ■ `setAttribute()` if the attribute was changed ■ `removeAttribute()` if the attribute was removed.
session	No constraint on applications.	Potentially poorer throughput and response time as compared to the `modified-session` and the `modified-attribute` options.
modified-attribute	Better throughput and response time for requests in which the percentage of session state modified is low.	As the percentage of session state modified for a given request nears 60%, throughput and response time degrade. In such cases, the performance is worse than the other options because of the overhead of splitting the attributes into separate records.

Stateful Session Bean Checkpointing

For SFSB session persistence, the load on HADB depends on the following:

- Number of SFSBs enabled for checkpointing.
- Which SFSB methods are selected for checkpointing, and how often they are used.
- Size of the session object.
- Which methods are transactional.

Checkpointing generally occurs after any transaction involving the SFSB is completed (even if the transaction rolls back).

For better performance, specify a small set of methods for checkpointing. The size of the data that is being checkpointed and the frequency of checkpointing determine the additional overhead in response time for a given client interaction.

Planning the Network Configuration

When planning how to integrate the Application Server into the network, estimate the bandwidth requirements and plan the network in such a way that it can meet users' performance requirements.

The following topics are covered in this section:

Estimating Bandwidth Requirements

To decide on the desired size and bandwidth of the network, first determine the network traffic and identify its peak. Check if there is a particular hour, day of the week, or day of the month when overall volume peaks, and then determine the duration of that peak.

During peak load times, the number of packets in the network is at its highest level. In general, if you design for peak load, scale your system with the goal of handling 100 percent of peak volume. Bear in mind, however, that any network behaves unpredictably and that despite your scaling efforts, it might not always be able handle 100 percent of peak volume.

For example, assume that at peak load, five percent of users occasionally do not have immediate network access when accessing applications deployed on Application Server. Of that five percent, estimate how many users retry access after the first attempt. Again, not all of those users might get through, and of that unsuccessful portion, another percentage will retry. As a result, the peak appears longer because peak use is spread out over time as users continue to attempt access.

Calculating Bandwidth Required

Based on the calculations made in "Establishing Performance Goals" on page 43, determine the additional bandwidth required for deploying the Application Server at your site.

Depending on the method of access (T-1 lines, ADSL, cable modem, and so on), calculate the amount of increased bandwidth required to handle your estimated load. For example, suppose your site uses T-1 or higher-speed T-3 lines. Given their bandwidth, estimate how many lines are needed on the network, based on the average number of requests generated per second at your site and the maximum peak load. Calculate these figures using a web site analysis and monitoring tool.

EXAMPLE 2–3 Calculation of Bandwidth Required

A single T-1 line can handle 1.544 Mbps. Therefore, a network of four T-1 lines can handle approximately 6 Mbps of data. Assuming that the average HTML page sent back to a client is 30 kilobytes (KB), this network of four T-1 lines can handle the following traffic per second:

6,176,000 bits/8 bits = 772,000 bytes per second

772,000 bytes per second/30 KB = approximately 25 concurrent response pages per second.

With traffic of 25 pages per second, this system can handle 90,000 pages per hour (25 x 60 seconds x 60 minutes), and therefore 2,160,000 pages per day maximum, assuming an even load throughout the day. If the maximum peak load is greater than this, increase the bandwidth accordingly.

Estimating Peak Load

Having an even load throughout the day is probably not realistic. You need to determine when the peak load occurs, how long it lasts, and what percentage of the total load is the peak load.

EXAMPLE 2–4 Calculation of Peak Load

If the peak load lasts for two hours and takes up 30 percent of the total load of 2,160,000 pages, this implies that 648,000 pages must be carried over the T-1 lines during two hours of the day.

Therefore, to accommodate peak load during those two hours, increase the number of T-1 lines according to the following calculations:

648,000 pages/120 minutes = 5,400 pages per minute

5,400 pages per minute/60 seconds = 90 pages per second

If four lines can handle 25 pages per second, then approximately four times that many pages requires four times that many lines, in this case 16 lines. The 16 lines are meant for handling the realistic maximum of a 30 percent peak load. Obviously, the other 70 percent of the load can be handled throughout the rest of the day by these many lines.

Configuring Subnets

If you use the separate tier topology, where the application server instances and HADB nodes are on separate host machines, you can improve performance by having all HADB nodes on a separate subnet. This is because HADB uses the User Datagram Protocol (UDP). Using a separate subnet reduces the UDP traffic on the machines outside of that subnet. Note, however, that all HADB nodes must be on the same subnet.

You can still run the management client from a different subnet as long as all the nodes and management agents are on the same subnet. All hosts and ports should be accessible within all node agents and node must not be blocked by firewalls, blocking of UDP, and so on.

HADB uses UDP multicast, so any subnet containing HADB nodes must be configured for multicast.

Choosing Network Cards

For greater bandwidth and optimal network performance, use at least 100 Mbps Ethernet cards or, preferably, 1 Gbps Ethernet cards between servers hosting the Application Server and the HADB nodes.

Network Settings for HADB

Note – HADB uses UDP multicast and thus you must enable multicast on your system's routers and host network interface cards. If HADB spans multiple sub-networks, you must also enable multicast on the routers between the sub-networks. For best results, put HADB nodes all on same network. Application server instances may be on a different sub network.

The following suggestions will enable HADB to work optimally in the network:

- Use switched routers so that each network interface has a dedicated 100 Mbps or better Ethernet channel.
- When running HADB on a multi-CPU machine hosting four or more HADB nodes, use 1 Gbps Ethernet cards. If the average session size is greater than 50 KB, use 1 Gbps Ethernet cards even if there are less than four HADB nodes per machine.
- If you suspect network bottlenecks within HADB nodes:
  - Run network monitoring software on your HADB servers to diagnose the problem.
  - Consider replacing any 100 Mbps Ethernet cards in the network with 1 Gbps Ethernet cards.

Planning for Availability

This section contains the following topics:

Rightsizing Availability

To plan availability of systems and applications, assess the availability needs of the user groups that access different applications. For example, external fee-paying users and business partners often have higher quality of service (QoS) expectations than internal users. Thus, it may be more acceptable to internal users for an application feature, application, or server to be unavailable than it would be for paying external customers.

The following figure illustrates the increasing cost and complexity of mitigating against decreasingly probable events. At one end of the continuum, a simple load-balanced cluster can tolerate localized application, middleware, and hardware failures. At the other end of the scale, geographically distinct clusters can mitigate against major catastrophes affecting the entire data center.

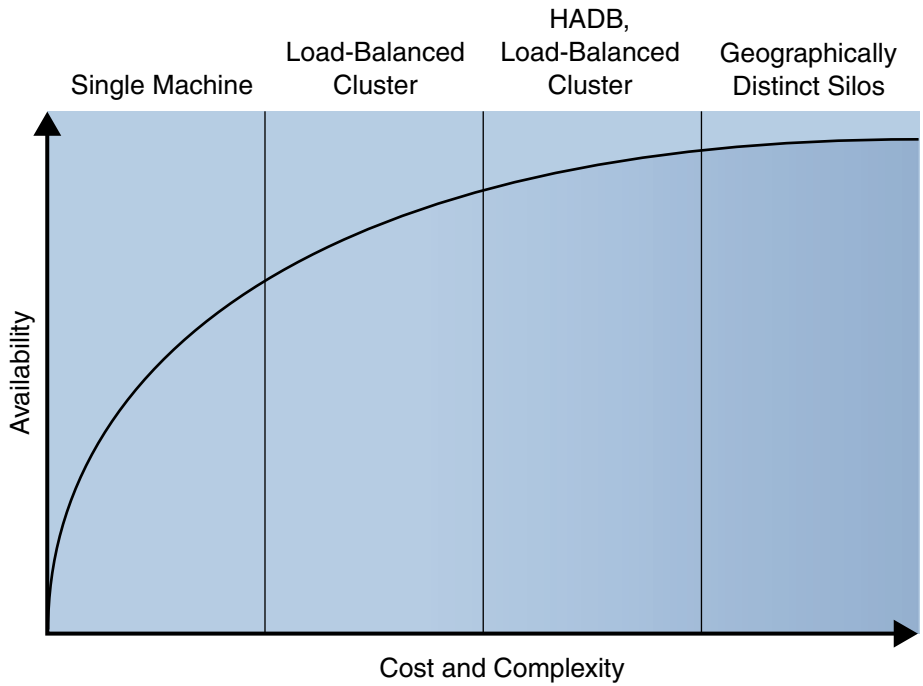


FIGURE 2–3 Availability versus Cost and Complexity

To realize a good return on investment, it often makes sense identify availability requirements of features within an application. For example, it may not be acceptable for an insurance quotation system to be unavailable (potentially turning away new business), but brief unavailability of the account management function (where existing customers can view their current coverage) is unlikely to turn away existing customers.

Using Clusters to Improve Availability

At the most basic level, a cluster is a group of application server instances—often hosted on multiple physical servers—that appear to clients as a single instance. This provides horizontal scalability as well as higher availability than a single instance on a single machine. This basic level of clustering works in conjunction with the Application Server's HTTP load balancer plug-in, which accepts HTTP and HTTPS requests and forwards them to one of the application server instances in the cluster. The ORB and integrated JMS brokers also perform load balancing to application server clusters. If an instance fails, become unavailable (due to network faults), or becomes unresponsive, requests are redirected only to existing, available machines. The load balancer can also recognize when an failed instance has recovered and redistribute load accordingly.

The HTTP load balancer also provides a *health checker* program that can monitor servers and specific URLs to determine whether they are available. You must carefully manage the overhead of health checking so that it does not become a large processing burden itself.

For stateless applications or applications that only involve low-value, simple user transactions, a simple load balanced cluster is often all that is required. For stateful, mission-critical applications, consider using HADB for session persistence. For an overview of HADB, see "High-Availability Database" on page 32 in Chapter 1, "Product Concepts" Application Server Administration Guide.

To perform online upgrades of applications, it is best to group the application server instances into multiple clusters. The Application Server has the ability to *quiesce* both applications and instances. Quiescence is the ability to take an instance (or group of instances) or a specific application offline in a controlled manner without impacting the users currently being served by the instance or application. As one instance is quiesced, new users are served by the upgraded application on another instance. This type of application upgrade is called a *rolling upgrade*. For more information on upgrading live applications, see "Upgrading Applications Without Loss of Availability" in *Sun Java System Application Server Enterprise Edition 8.2 High Availability Administration Guide*.

Adding Redundancy to the System

One way to achieve high availability is to add hardware and software redundancy to the system. When one unit fails, the redundant unit takes over. This is also referred to as fault tolerance. In general, to maximize high availability, determine and remove every possible point of failure in the system.

Identifying Failure Classes

The level of redundancy is determined by the failure classes (types of failure) that the system needs to tolerate. Some examples of failure classes are:

- System process
- Machine
- Power supply
- Disk
- Network failures
- Building fires or other preventable disasters
- Unpredictable natural catastrophes

Duplicated system processes tolerate single system process failures, as well as single machine failures. Attaching the duplicated mirrored (paired) machines to different power supplies tolerates single power failures. By keeping the mirrored machines in separate buildings, a single-building fire can be tolerated. By keeping them in separate geographical locations, natural catastrophes like earthquakes can be tolerated.

Using HADB Redundancy Units to Improve Availability

To improve availability, HADB nodes are always used in Data Redundancy Units (DRUs) as explained in "Establishing Performance Goals" on page 43.

Using HADB Spare Nodes to Improve Fault Tolerance

Using spare nodes improves fault tolerance. Although spare nodes are not mandatory, they provide maximum availability.

Planning Failover Capacity

Failover capacity planning implies deciding how many additional servers and processes you need to add to the Application Server deployment so that in the event of a server or process failure, the system can seamlessly recover data and continue processing. If your system gets overloaded, a process or server failure might result, causing response time degradation or even total loss of service. Preparing for such an occurrence is critical to successful deployment.

To maintain capacity, especially at peak loads, add spare machines running Application Server instances to the existing deployment.

For example, consider a system with two machines running one Application Server instance each. Together, these machines handle a peak load of 300 requests per second. If one of these machines becomes unavailable, the system will be able to handle only 150 requests, assuming an even load distribution between the machines. Therefore, half the requests during peak load will not be served.

Design Decisions

Design decisions include whether you are designing the system for peak or steady-state load, and the number of machines in various roles and their sizes.

Designing for Peak or Steady State Load

In a typical deployment, there is a difference between steady state and peak workloads:

- If the system is designed to handle peak load, it can sustain the expected maximum load of users and requests without degrading response time. This implies that the system can handle extreme cases of expected system load. If the difference between peak load and steady state load is substantial, designing for peak loads can mean spending money on resources that are often idle.
- If the system is designed to handle steady state load, it does not have all the resources required to handle the expected peak load. Thus, the system has a slower response time when peak load occurs.

How often the system is expected to handle peak load will determine whether you want to design for peak load or for steady state.

If peak load occurs often—say, several times per day—it may be worthwhile to expand capacity to handle it. If the system operates at steady state 90 percent of the time, and at peak only 10 percent of the time, then it may be preferable to deploy a system designed around steady state load. This implies that the system's response time will be slower only 10 percent of the time. Decide if the frequency or duration of time that the system operates at peak justifies the need to add resources to the system.

System Sizing

Based on the load on the application server instances, the load on the HADB, and failover requirements, you can determine:

- "Number of Application Server Instances" on page 59
- "Number of HADB Nodes" on page 59
- "Number of HADB Hosts" on page 60
- "HADB Storage Capacity" on page 61

Number of Application Server Instances

To determine the number of applications server instances (hosts) needed, evaluate your environment on the basis of the factors explained in "Estimating Load on Application Server Instances" on page 44 to each application server instance, although each instance can use more than one Central Processing Unit (CPU).

Number of HADB Nodes

As a general guideline, plan to have one HADB node for each CPU in the system. For example, use two HADB nodes for a machine that has two CPUs.

Note – If you have more than one HADB node per machine (for example, if you are using bigger machines), then you must ensure that there is enough redundancy and scalability on the machines; for example multiple uninterruptible power supplies and independent disk controllers.

Alternatively, use the following procedure.

▼ To determine the required number of HADB nodes

1 Determine the following parameters:

- Maximum number of concurrent users, n_{users}.
- Average BLOB size, s.

- Maximum transaction rate per user, referred to as NTPS.

2 **Determine the size in Gigabytes of the maximum primary data volume, V_{data}.**

Use the following formula:

$V_{data} = n_{users} \cdot s$

3 **Determine the maximum HADB data transfer rate, R_{dt}.**

This reflects the data volume shipped into HADB from the application side. Use the following formula:

$R_{dt} = n_{users} \cdot s \cdot \text{NTPS}$

4 **Determine the number of nodes, $N_{NODES,}$.**

Use the following formula:

$N_{NODES} = V_{data} / 5\text{GB}$

Round this value up to an even number, since nodes work in pairs.

Number of HADB Hosts

Determine the number of HADB hosts based on data transfer requirements. This calculation assumes all hosts have similar hardware configurations and operating systems, and have the necessary resources to accommodate the nodes they run.

▼ To calculate the number of hosts

1 **Determine the maximum host data transfer rate, $R_{max.}$.**

Determine this value empirically, because it depends on network and host hardware. Note this is different from the maximum HADB data transfer rate, R_{dt}, determined in the previous section.

2 **Determine the number of hosts needed to accommodate this data**

Updating a volume of data V distributed over a number of hosts N_{HOSTS} causes each host to receive approximately $4V/N_{HOSTS}$ of data. Determine the number of hosts needed to accommodate this volume of data with the following formula:

$N_{HOSTS} = 4 \cdot R_{dt} / R_{max}$

Round this value up to the nearest even number to get the same number of hosts for each DRU.

3 **Add one host on each DRU for spare nodes.**

If each of the other hosts run N data nodes, let this host run N spare nodes. This allows for single-machine failure taking down N data nodes.

Each host needs to run at least one node, so if the number of nodes is less than the number of hosts ($N_{NODES} < N_{HOSTS}$), adjust N_{NODES} to be equal to N_{HOSTS}. If the number of nodes is greater than the number of hosts, (N_{NODES} \> N_{HOSTS}), several nodes can be run on the same host.

HADB Storage Capacity

The HADB provides near-linear scaling with the addition of more nodes until the network capacity is exceeded. Each node must be configured with storage devices on a dedicated disk or disks. All nodes must have equal space allocated on the storage devices. Make sure that the storage devices are allocated on local disks.

Suppose the expected size session data is *x* MB. Where *x* MB is the total amount of storage for the entire system (i.e., *x* = N users * s). HADB replicates data on mirror nodes, and therefore requires 2*x* MB of storage. Further, HADB uses indexes to enable fast access to data. The two nodes will require an additional 2*x* MB for indexes, for a total required storage capacity of 4*x*. Therefore, HADB's expected storage capacity requirement is four times the expected data volume.

To account for future expansion without loss of data from HADB, you must provide additional storage capacity for online upgrades because you might want to refragment the data after adding new nodes. In this case, a similar amount (4*x*) of additional space on the data devices is required. Thus, the expected storage capacity is eight times (8x) the expected data volume.

Additionally, HADB uses disk space as follows:

- Space for temporary storage of log buffer. This space is four times the log buffer size. The log buffer keeps track of operations related to data. The default value of the log buffer size is 48 MB.
- Space for internal administration purpose. This space is one percent of the storage device size.

The following table summarizes the HADB storage space requirements for the total session data of *x* MB. Note that this is the total amount of session storage data spread across all the nodes of the HADB database. Device size per node will be shared by all devices specified for that node.

TABLE 2–3 HADB Storage Space Requirement for Total Session Data Size of x MB

Condition	HADB Storage Space Required
Addition or removal of HADB nodes while online is *not* required.	(4*x*/N nodes) MB + (4*log buffer size) + 1% of device size
Addition or removal of HADB nodes while online is required.	(8*x*/N nodes) MB + (4*log buffer size) + 1% of device size

If the HADB runs out of device space, it will not accept client requests to insert or update data. However, it will accept delete operations. If the HADB runs out of device space, it returns error codes 4593 or 4592 and writes corresponding error messages to the history files. For more

information on these messages, seeChapter 14, "HADB Error Messages," in *Sun Java System Application Server Enterprise Edition 8.2 Error Message Reference*.

Planning Message Queue Broker Deployment

The Java Message Service (JMS) API is a messaging standard that allows J2EE applications and components to create, send, receive, and read messages. It enables distributed communication that is loosely coupled, reliable, and asynchronous. The Sun Java System Message Queue, which implements JMS, is integrated with Application Server, enabling you to create components such as message-driven beans (MDBs).

Sun Java System Message Queue (MQ) is integrated with Application Server using a *connector module,* also known as a resource adapter, as defined by the J2EE Connector Architecture Specification (JCA) 1.5. A connector module is a standardized way to add functionality to the Application Server. J2EE components deployed to the Application Server exchange JMS messages using the JMS provider integrated via the connector module. By default, the JMS provider is the Sun Java System Message Queue, but if you wish you can use a different JMS provider, as long as it implements JCA 1.5.

Creating a JMS resource in Application Server creates a connector resource in the background. So, each JMS operation invokes the connector runtime and uses the MQ resource adapter in the background.

In addition to using resource adapter APIs, Application Server uses additional MQ APIs to provide better integration with MQ. This tight integration enables features such as connector failover, load balancing of outbound connections, and load balancing of inbound messages to MDBs. These features enable you to make messaging traffic fault-tolerant and highly available.

Multi-Broker Clusters

MQ Enterprise Edition supports using multiple interconnected broker instances known as a *broker cluster*. With broker clusters, client connections are distributed across all the brokers in the cluster. Clustering provides horizontal scalability and improves availability.

A single message broker scales to about eight CPUs and provides sufficient throughput for typical applications. If a broker process fails, it is automatically restarted. However, as the number of clients connected to a broker increases, and as the number of messages being delivered increases, a broker will eventually exceed limitations such as number of file descriptors and memory.

Having multiple brokers in a cluster rather than a single broker enables you to:

- Provide messaging services despite hardware failures on a single machine.
- Minimize downtime while performing system maintenance.

- Accommodate workgroups having different user repositories.
- Deal with firewall restrictions.

However, having multiple brokers does not ensure that transactions in progress at the time of a broker failure will continue on the alternate broker. While MQ will re-establish a failed connection with a different broker in a cluster, it will lose transactional messaging and roll back transactions in progress. User applications will not be affected, except for transactions that could not be completed. Service failover is assured since connections continue to be usable.

Thus, MQ does not support high availability persistent messaging in a cluster. If a broker restarts after failure, it will automatically recover and complete delivery of persistent messages. Persistent messages may be stored in a database or on the file system. However if the machine hosting the broker does not recover from a hard failure, messages may be lost.

The Solaris platform with Sun Cluster Data Service for Sun Message Queue supports transparent failover of persistent messages. This configuration leverages Sun Cluster's global file system and IP failover to deliver true high availability and is included with Java Enterprise System.

Master Broker and Client Synchronization

In a multi-broker configuration, each destination is replicated on all of the brokers in a cluster. Each broker knows about message consumers that are registered for destinations on all other brokers. Each broker can therefore route messages from its own directly-connected message producers to remote message consumers, and deliver messages from remote producers to its own directly-connected consumers.

In a cluster configuration, the broker to which each message producer is directly connected performs the routing for messages sent to it by that producer. Hence, a persistent message is both stored and routed by the message's *home broker*.

Whenever an administrator creates or destroys a destination on a broker, this information is automatically propagated to all other brokers in a cluster. Similarly, whenever a message consumer is registered with its home broker, or whenever a consumer is disconnected from its home broker—either explicitly or because of a client or network failure, or because its home broker goes down—the relevant information about the consumer is propagated throughout the cluster. In a similar fashion, information about durable subscriptions is also propagated to all brokers in a cluster.

Configuring Application Server to Use Message Queue Brokers

The Application Server's Java Message Service represents the connector module (resource adapter) for the Message Queue. You can manage the Java Message Service through the Admin Console or the `asadmin` command-line utility.

MQ brokers (JMS hosts) run in a separate JVM from the Application Server process. This allows multiple Application Server instances or clusters to share the same set of MQ brokers.

In Application Server, a *JMS host* refers to an MQ broker. The Application Server's Java Message Service configuration contains a JMS Host List (also called AddressList) that contains all the JMS hosts that will be used.

Managing JMS with Admin Console

In the Admin Console, you can set JMS properties using the Java Message Service node for a particular configuration. You can set properties such as Reconnect Interval and Reconnect Attempts. For more information, see Chapter 4, "Configuring Java Message Service Resources," in *Sun Java System Application Server Enterprise Edition 8.2 Administration Guide*.

The JMS Hosts node under the Java Message Service node contains a list of JMS hosts. You can add and remove hosts from the list. For each host, you can set the host name, port number, and the administration user name and password. By default, the JMS Hosts list contains one MQ broker, called "default_JMS_host," that represents the local MQ broker integrated with the Application Server.

Configure the JMS Hosts list to contain all the MQ brokers in the cluster. For example, to set up a cluster containing three MQ brokers, add a JMS host within the Java Message Service for each one. Message Queue clients use the configuration information in the Java Message Service to communicate with MQ broker.

Managing JMS with asadmin

In addition to the Admin Console, you can use the `asadmin` command-line utility to manage the Java Message Service and JMS hosts. Use the following `asadmin` commands:

- Configuring Java Message Service attributes: `asadmin set`
- Managing JMS hosts:
  - `asadmin create-jms-host`
  - `asadmin delete-jms-host`
  - `asadmin list-jms-hosts`

  Managing JMS resources:
  - `asadmin create-jms-resource`
  - `asadmin delete-jms-resource`
  - `asadmin list-jms-resources`

    For more information on these commands, see *Sun Java System Application Server Enterprise Edition 8.2 Reference Manual*or the corresponding man pages.

Java Message Service Type

There are two types of integration between Application Server and MQ brokers: local and remote. You can set this type attribute on the Admin Console's Java Message Service page.

Local Java Message Service

If the Type attribute is LOCAL, the Application Server will start and stop the MQ broker. When Application Server starts up, it will start the MQ broker specified as the Default JMS host. Likewise, when the Application Server instance shuts down, it shuts down the MQ broker. LOCAL type is most suitable for standalone Application Server instances.

With LOCAL type, use the Start Arguments attribute to specify MQ broker startup parameters.

Remote Java Message Service

If the Type attribute is REMOTE, Application Server will use an externally configured broker or broker cluster. In this case, you must start and stop MQ brokers separately from Application Server, and use MQ tools to configure and tune the broker or broker cluster. REMOTE type is most suitable for Application Server clusters.

With REMOTE type, you must specify MQ broker startup parameters using MQ tools. The Start Arguments attribute is ignored.

Default JMS Host

You can specify the default JMS Host in the Admin Console Java Message Service page. If the Java Message Service type is LOCAL, then Application Server will start the default JMS host when the Application Server instance starts.

To use an MQ broker cluster, delete the default JMS host, then add all the MQ brokers in the cluster as JMS hosts. In this case, the default JMS host becomes the first JMS host in the JMS host list.

You can also explicitly set the default JMS host to one of the JMS hosts. When the Application Server uses a Message Queue cluster, the default JMS host executes MQ-specific commands. For example, when a physical destination is created for a MQ broker cluster, the default JMS host executes the command to create the physical destinations, but all brokers in the cluster use the physical destination.

Example Deployment Scenarios

To accommodate your messaging needs, modify the Java Message Service and JMS host list to suit your deployment, performance, and availability needs. The following sections describe some typical scenarios.

For best availability, deploy MQ brokers and Application Servers on different machines, if messaging needs are not just with Application Server. Another option is to run an Application Server instance and an MQ broker instance on each machine until there is sufficient messaging capacity.

Default Deployment

Installing the Application Server automatically creates a domain administration server (DAS). By default, the Java Message Service type for the DAS is LOCAL. So, starting DAS will also start its default MQ broker.

Creating a new domain will also create a new broker. By default, when you add a standalone server instance or a cluster to the domain, its Java Message Service will be configured as REMOTE and its default JMS host will be the broker started by DAS.

The figure below illustrates an example default deployment with an Application Server cluster containing three instances.

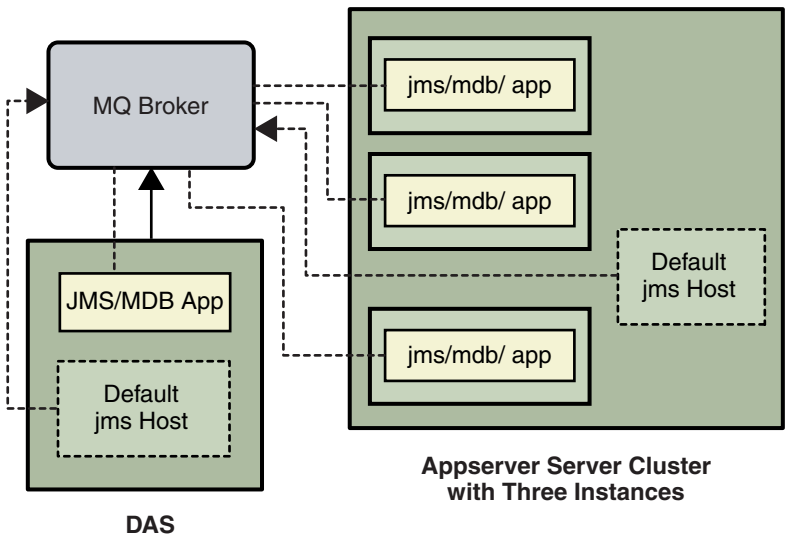


FIGURE 2–4 Default MQ Deployment

Using an MQ Broker Cluster with an Application Server Cluster

To configure an Application Server cluster to use an MQ broker cluster, add all the MQ brokers as JMS hosts in the Application Server's Java Message Service. Any JMS connection factories created and MDBs deployed will then use the JMS configuration specified.

The following figure illustrates an example deployment with three MQ brokers in an broker cluster and three Application Server instances in a cluster.

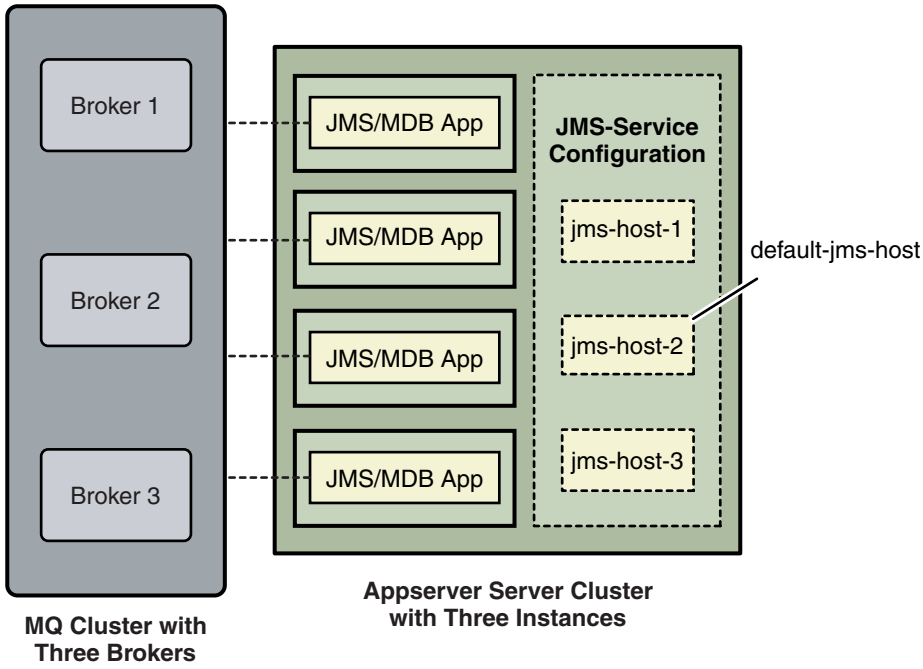


FIGURE 2–5 Application Server Cluster Using an MQ Broker Cluster

Specifying an Application-Specific MQ Broker Cluster

In some cases, an application may need to use a different MQ broker cluster than the one used by the Application Server cluster. The following figure illustrates an example of such a scenario. To do so, use the `AddressList` property of a JMS connection factory or the `activation-config` element in an MDB deployment descriptor to specify the MQ broker cluster.

For more information about configuring connection factories, see "JMS Connection Factories" in *Sun Java System Application Server Enterprise Edition 8.2 Administration Guide*. For more information about MDBs, see "Using Message-Driven Beans" in *Sun Java System Application Server Enterprise Edition 8.2 Developer's Guide*.

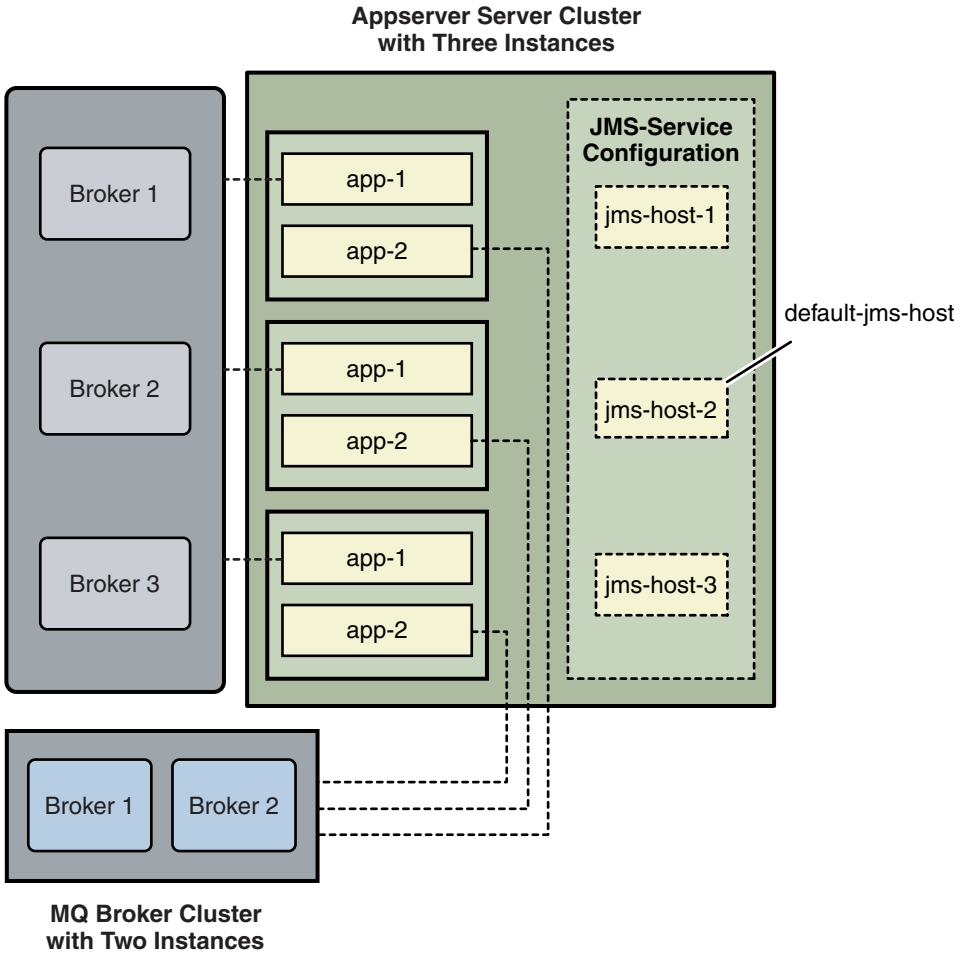


FIGURE 2–6 Application-specific MQ broker cluster

Application Clients

When an application client or standalone application accesses a JMS administered object for the first time, the client JVM retrieves the Java Message Service configuration from the server. Further changes to the JMS service will not be available to the client JVM until it is restarted.

◆ ◆ ◆ CHAPTER 3

Selecting a Topology

After estimating the factors related to performance as explained in Chapter 1, "Product Concepts" the Application Server. A topology is the arrangement of machines, Application Server instances, and HADB nodes, and the communication flow among them.

There are two fundamental deployment topologies. Both topologies have common building blocks: multiple Application Server instances in a cluster, a mirrored set of HADB nodes, and HADB spare nodes. Both of them require a set of common configuration settings to function properly.

This chapter discusses:

- "Common Requirements" on page 69 for both topologies.
- The two topologies:
  - "Co-located Topology" on page 71 - Application Server instances and HADB nodes are on the same machine.
  - "Separate Tier Topology" on page 75 - Application Server instances and HADB nodes are on different machines.
- "Determining Which Topology to Use" on page 79

Common Requirements

This section describes the requirements that are common to both topologies:

- "General Requirements" on page 69
- "HADB Nodes and Machines" on page 70
- "Load Balancer Configuration" on page 71

General Requirements

Both topologies must meet the following general requirements:

- Machines that host HADB nodes must be in pairs. That is, there must be an even number of them.
- Each data redundancy unit (DRU) must have the same number of machines. Create the HADB database in such a way that the mirrored (paired) nodes are on a different DRU than the primary nodes.
- Each machine that hosts HADB nodes must have local disk storage, used to store all persisted information in the HADB.
- Machines that host the HADB nodes must run the same operating system. It is best to use identical or nearly identical machines, in terms of configuration and performance.
- For HTTP and SFSB session information to be persisted to the HADB, the Application Server instances must be in a cluster and satisfy all related requirements. For more information on configuring clusters, see Chapter 6, "Using Application Server Clusters," in *Sun Java System Application Server Enterprise Edition 8.2 High Availability Administration Guide*.
- Machines hosting the Application Server instances must be as identical as possible, in terms of configuration and performance. This is because the load balancer plug-in uses a round-robin policy for load balancing, and if machines of different classes host instances, then the load will not be balanced in the most optimum way across these machines.
- Preferably have a separate uninterruptible power supply (UPS) for each DRU.

HADB Nodes and Machines

Each DRU contains a complete copy of the data in HADB and can continue servicing requests if the other DRU becomes unavailable. However, if a node in one DRU and its mirror in another DRU fail at the same time, some portion of data is lost. For this reason, it is important that the system is not set up so that both DRUs can be affected by a single failure such as a power failure or disk failure.

Note – Each DRU must run on a completely independent, redundant system.

Follow these guidelines when setting up the HADB nodes and machines:

- To increase capacity and throughput, add nodes in pairs with one node for each DRU.
- Set up each DRU with a number of spare nodes equal to the number of nodes running on each machine. This is because if each machine in the configuration runs *n* data nodes, the failure of a single machine brings down *n* nodes.
- Run the same number of HADB nodes on all machines to balance load as evenly as possible.

Caution – Do not run nodes from different DRUs on the same machine. If you must run nodes from different DRUs on the same machine, ensure that the machine can handle any single point of failure (for failures related to disk, memory, CPU, power, operating system crashes, and so on).

Load Balancer Configuration

Both the topologies have Application Server instances in a cluster. These instances persist session information to the HADB. Configure the load balancer to include configuration information for all the Application Server instances in the cluster.

For more information on setting up a cluster and adding Application Server instances to clusters, see Chapter 6, "Using Application Server Clusters," in *Sun Java System Application Server Enterprise Edition 8.2 High Availability Administration Guide*.

Co-located Topology

In the co-located topology, the Application Server instance and the HADB nodes are on the same machine (hence the name *co-located*). This topology requires fewer machines than the separate tier topology. The co-located topology uses CPUs more efficiently—an Application Server instance and an HADB node share one machine and the processing is distributed evenly among them.

This topology requires a minimum of two machines. To improve throughput, add more machines in pairs.

Note – The co-located topology is a good for large, symmetric multiprocessing (SMP) machines, since you can take full advantage of the processing power of these machines.

Example Configuration

The following figure illustrates an example configuration of the co-located topology.

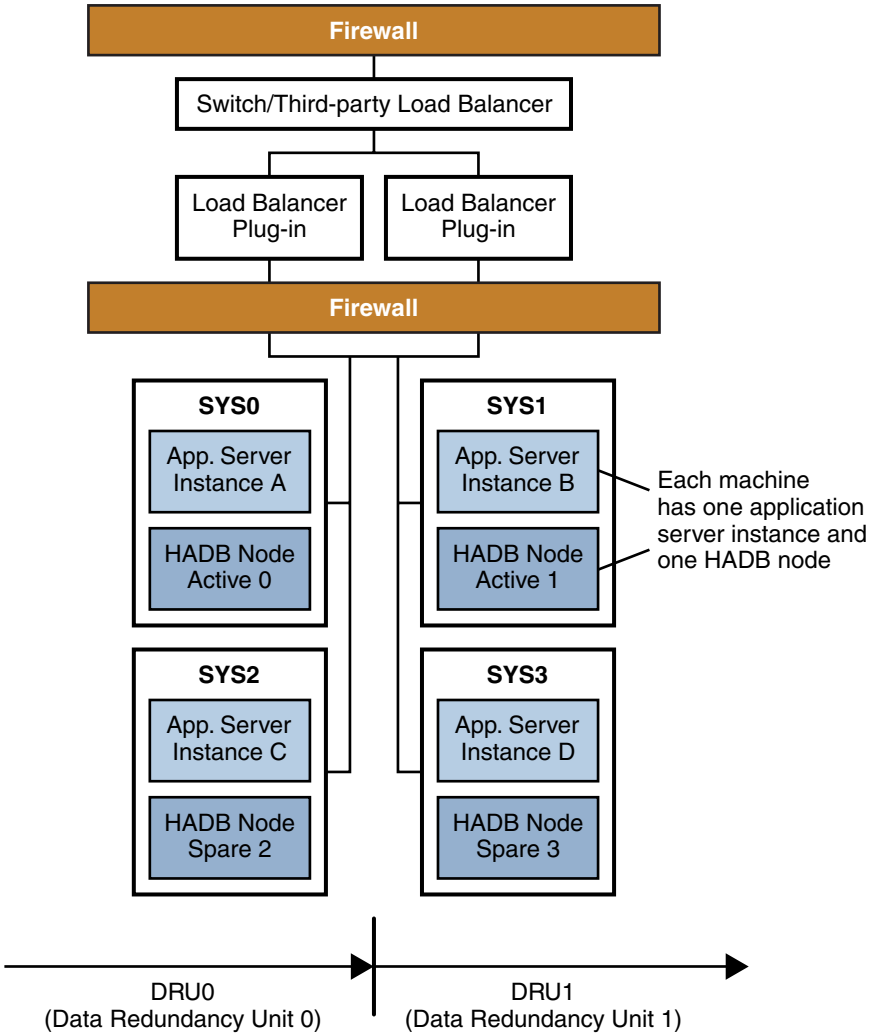


FIGURE 3–1 Example Co-located Topology

Machine SYS0 hosts Application Server instance A, machine SYS1 hosts Application Server instance B, machine SYS2 hosts Application Server instance C, and machine SYS3 hosts Application Server instance D.

These four instances form a cluster that persists information to the two DRUs:

- **DRU0** comprises two machines, SYS0 and SYS2. HADB node active 0 is on the machine SYS0. HADB node spare 2 is on the machine SYS2.

- **DRU1** comprises two machines, SYS1 and SYS3. HADB node active 1 is on the machine SYS1. HADB node spare 3 is on the machine SYS3.

Variation of Co-located Topology

For better scalability and throughput, increase the number of Application Server instances and HADB nodes by adding more machines. For example, you could add two machines, each with one Application Server instance and one HADB node. Make sure to add the HADB nodes in pairs, assigning one node for each DRU. "Variation of Co-located Topology" on page 73 illustrates this configuration.

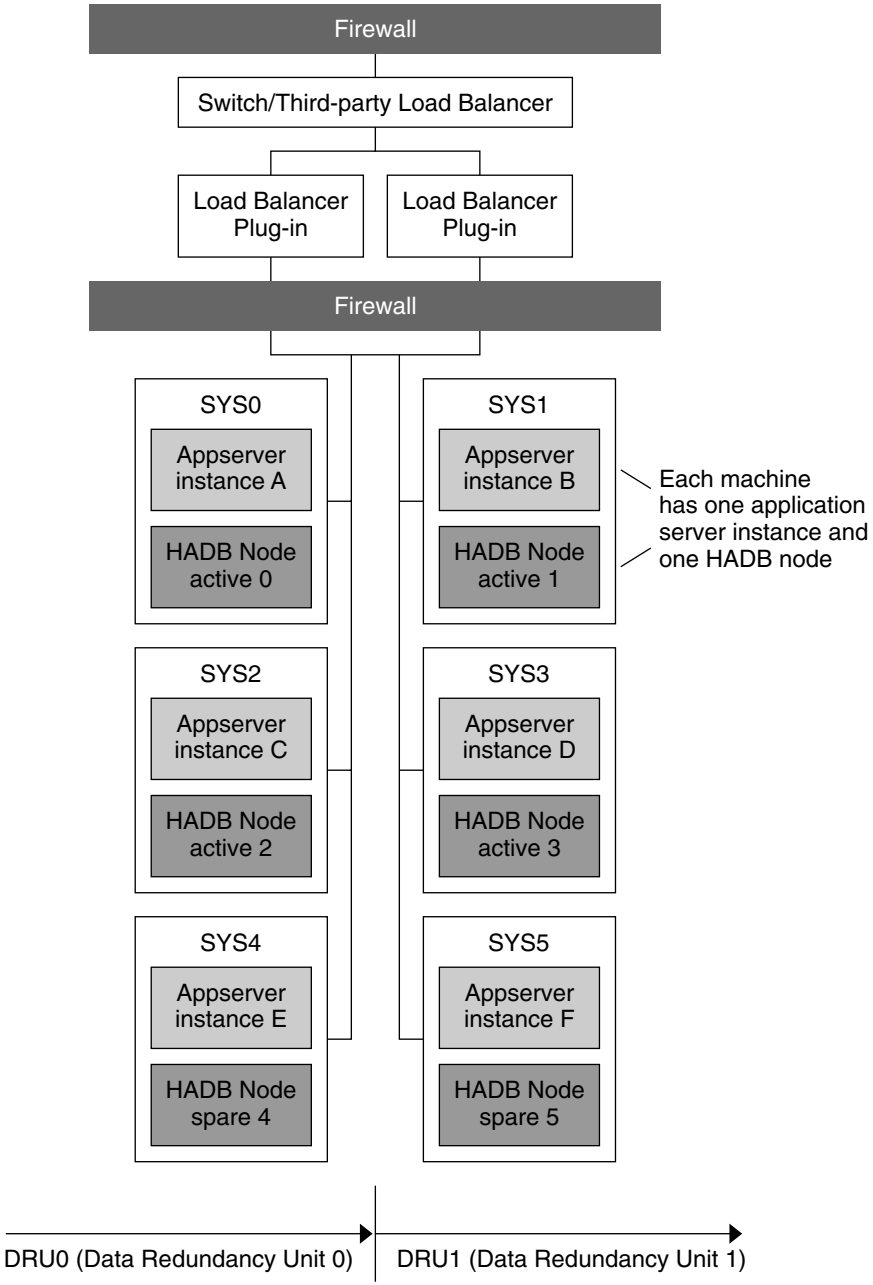


FIGURE 3–2 Variation of Co-located Topology

In this variation, the machines SYS4 and SYS5 have been added to the co-located topology described in "Example Configuration" on page 71.

Application Server instances are hosted as follows:

- Machine SYS0 hosts instance A
- Machine SYS1 hosts instance B
- Machine SYS2 hosts instance C
- Machine SYS3 hosts instance D
- Machine SYS4 hosts instance E
- Machine SYS5 hosts instance F

These instances form a cluster that persists information to the two DRUs:

- **DRU0** comprises machines SYS0, SYS2, and SYS4. HADB node active 0 is on the machine SYS0. HADB node active 2 is on the machine SYS2. HADB node spare 4 is on the machine SYS4.
- **DRU1** comprises the machines SYS1, SYS3, and SYS5. HADB node active 1 is on the machine SYS1. HADB node active 3 is on the machine SYS3. HADB node spare 5 is on the machine SYS5.

Separate Tier Topology

In this topology, Application Server instances and the HADB nodes are on different machines (hence the name *separate tier*).

This topology requires more hardware than the co-located topology. It might be a good fit if you have different types of machines—you can allocate one set of machines to host Application Server instances and another to host HADB nodes. For example, you could use more powerful machines for the Application Server instances and less powerful machines for HADB.

Example Configuration

The following figure illustrates the separate tier topology.

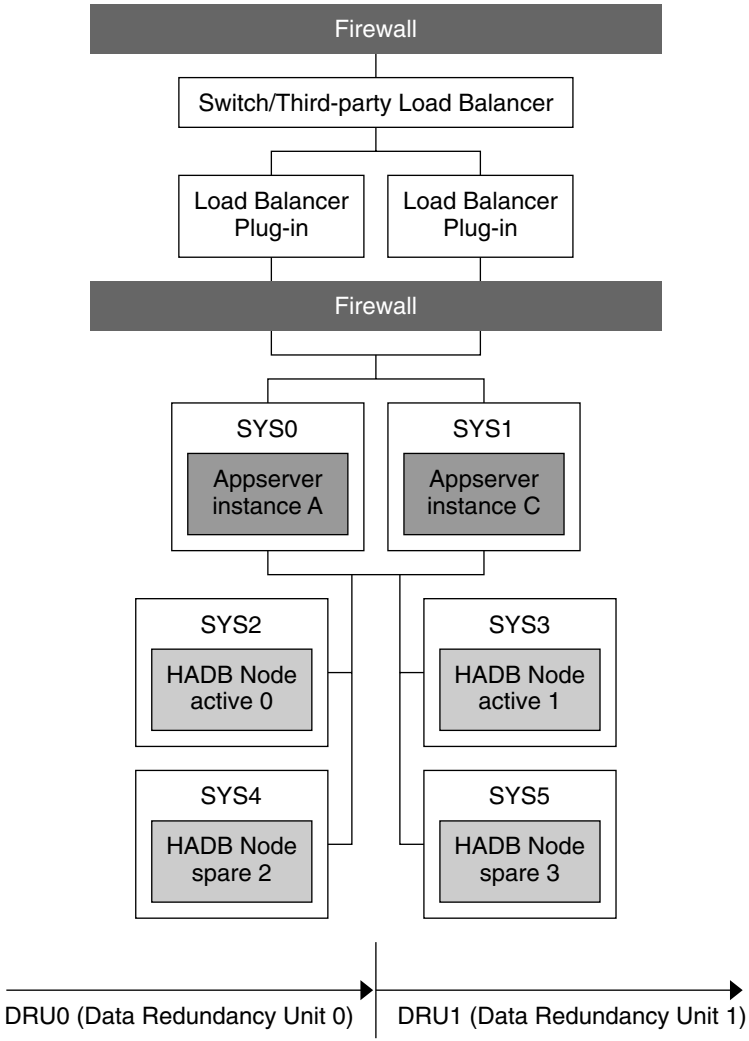


FIGURE 3–3 Example Separate Tier Topology

In this topology, machine SYS0 hosts Application Server instance A and machine SYS1 hosts Application Server instance B. These two instances form a cluster that persists session information to the two DRUs:

- **DRU0** comprises two machines, SYS2 and SYS4. HADB node active 0 is on machine SYS2 and the HADB node spare 2 is on machine SYS4.
- **DRU1** comprises two machines, SYS3 and SYS5. HADB node active 1 is on machine SYS3 and the HADB node spare 3 on machine SYS5.

All the nodes on a DRU are on different machines, so that even if one machine fails, the complete data for any DRU continues to be available on other machines.

Variation of Separate Tier Topology

A variation of the separate tier topology is to increase the number of Application Server instances by adding more machines horizontally to the configuration. For example, add another machine to the example configuration by creating a new Application Server instance. Similarly, increase the number of HADB nodes by adding more machines to host HADB nodes. Recall you must add the HADB nodes in pairs with one node for each DRU.

"Variation of Separate Tier Topology" on page 77 illustrates this configuration.

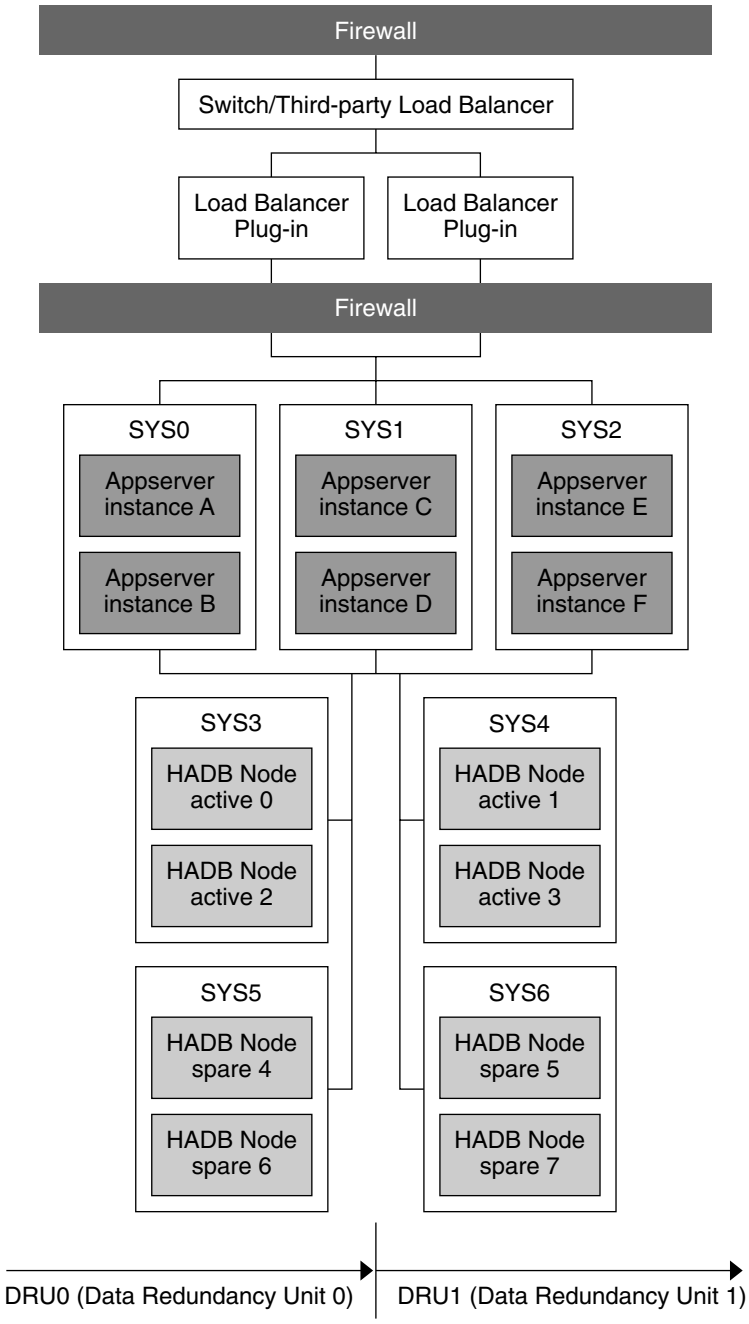


FIGURE 3–4 Variation of Separate Tier Topology

In this configuration, each machine hosting Application Server instances has two instances. There are thus a total of six Application Server instances in the cluster.

HADB nodes are on machines SYS3, SYS4, SYS5, and SYS6.

DRU0 comprises two machines:

- SYS3, which hosts HADB node active 0 and HADB node active 2.
- SYS5, with HADB node spare 4 and HADB node spare 6.

DRU1 comprises two machines:

- SYS4, which hosts HADB node active 1 and HADB node active 3.
- SYS6, which hosts HADB node spare 5 and HADB node spare 7.

Each machine hosting HADB nodes hosts two nodes. Thus, there are a total of eight HADB nodes: four active nodes and four spare nodes.

Determining Which Topology to Use

To determine which topology (or variation) best meets your performance and availability requirements, test the topologies and experiment with different combinations of machines and CPUs.

Determine what trade-offs are required to meet your goals. For example, if ease of maintenance is critical, the separate tier topology is more suitable. The trade-off is that this topology requires more machines than the co-located topology.

An important factor in the choice of topology is the type of machines available. If the system contains large, Symmetric Multiprocessing (SMP) machines, the co-located topology is attractive because you can take full advantage of the processing power of these machines. If the system contains various types of machines, the separate tier topology can be more useful because you can allocate a different set of machines to the Application Server tier and to the HADB tier. For example, you might want to use the most powerful machines for the Application Server tier and less powerful machines for the HADB tier.

Comparison of Topologies

The following table compares the co-located topology and the separate tier topology. The left column lists the name of the topology, the middle column lists the advantages of the topology, and the right column lists the disadvantages of the topology

TABLE 3–1 Comparison of Topologies

Topology	Advantages	Disadvantages
Co-located Topology	Requires fewer machines. Because the HADB nodes and the Application Server instances are on the same tier, you are able to create an Application Server instance on each spare node to handle additional load. Improved CPU utilization. Processing is distributed evenly between an Application Server instance and an HADB node sharing one machine. Useful for large, Symmetric Multiprocessing (SMP) machines since it takes full advantage of their processing power.	Increased complexity of maintenance. For example, when you have to shut down machines hosting HADB nodes to perform maintenance, application server instances on the machine also become unavailable.
Separate Tier Topology	Easier maintenance. For example, you are able to perform maintenance on the machines that host Application Server instances without having to bring down HADB nodes. Useful with different types of machines. You are able to allocate a different set of machines to the Application Server tier and the HADB tier. For example, you are able to use more powerful machines for the Application Server tier and the less powerful machines for the HADB tier.	Requires more machines than the co-located topology. Because application server instances and HADB nodes are located on separate tiers, application server instances cannot be located on the machines that host the HADB spare nodes. Reduced CPU utilization. The application server tier and the HADB tier will likely have uneven loads. This is more significant with a small number of machines (four to six).

◆ ◆ ◆ CHAPTER 4

Checklist for Deployment

This appendix provides a checklist to get started on evaluation and production with the Application Server.

Checklist for Deployment

TABLE 4–1 Checklist

Component/Feature	Description
Application	Determine the following requirements for the application to be deployed. ■ Required/acceptable response time. ■ Peak load characteristics. ■ Necessary persistence scope and frequency. ■ Session timeout in `web.xml`. ■ Failover and availability requirements. For more information see *Sun Java System Application Server Enterprise Edition 8.2 Performance Tuning Guide*.
Hardware	■ Use the same type of hardware to host HADB nodes. ■ Have necessary amounts of hard disk space and memory installed. ■ Use the sizing exercise to identify the requirements for deployment. For more information see *Sun Java System Application Server Enterprise Edition 8.2 Release Notes*

TABLE 4–1 Checklist *(Continued)*

Component/Feature	Description
Operating System	■ Ensure that the product is installed on a supported platform. ■ Ensure that the patch levels are up-to-date and accurate. For more information see *Sun Java System Application Server Enterprise Edition 8.2 Release Notes*
Network Infrastructure	■ Identify single points of failures and address them. ■ Make sure that the NICs and other network components are correctly configured. ■ Run `ttcp` benchmark test to determine if the throughput meets the requirements/expected result. ■ Setup `rsh/ssh` based your preference so that HADB nodes are properly installed. For more information see *Sun Java System Application Server Enterprise Edition 8.2 Installation Guide*.
Back-ends and other external data sources	Check with the domain expert or vendor to ensure that these data sources are configured appropriately.
System Changes/Configuration	■ Make sure that changes to `/etc/system` and its equivalent on Linux are completed before running any performance/stress tests. ■ Make sure the changes to the TCP/IP settings are complete. ■ By default, the system comes with lots of services pre-configured. Not all of them are required to be running. Turn off services that are not needed to conserve system resources. ■ On Solaris, use `Setoolkit` to determine the behavior of the system. Resolve any flags that show up. For more information see *Sun Java System Application Server Enterprise Edition 8.2 Performance Tuning Guide*.
Application Server and HADB Installation	■ Ensure that these servers are not installed on NFS mounted volumes. ■ Check for enough disk space and RAM when installing both Application Server and the HADB nodes on the same machine. ■ Check for enough independent disks when installing multiple HADB nodes on the same system. For more information see Chapter 2, "Installing and Setting Up High Availability Database," in *Sun Java System Application Server Enterprise Edition 8.2 High Availability Administration Guide*

TABLE 4–1 Checklist *(Continued)*

Component/Feature	Description
HADB Configuration	■ Set the size of the HADB Data Device. ■ Define the DataBufferPoolSize. ■ Define the LogBufferSize. ■ Define the InternalBufferSize. ■ Set the NumberOfLocks. ■ Set optimum time-out values for various Application Server components. ■ Create the Physical layout of HADB nodes on the file system. For more information, see "Configuring HADB" in *Sun Java System Application Server Enterprise Edition 8.2 High Availability Administration Guide*.
Application Server Configuration	■ Logging: Enable access log rotation. ■ Choose the right logging level. WARNING is usually appropriate. ■ Configure J2EE containers using Admin Console. ■ Configure HTTP listeners using Admin Console. ■ Configure ORB threadpool using Admin Console. ■ If using Type2 drivers or calls involving native code, ensure that mtmalloc.so is specified in the LD_LIBRARY_PATH. ■ Ensure that the appropriate persistence scope and frequency are used and they are not overridden underneath in the individual Web/EJB modules. ■ Ensure that only critical methods in the SFSB are checkpointed. For more information on tuning, see *Sun Java System Application Server Enterprise Edition 8.2 Performance Tuning Guide*. For more information on configuration, see *Sun Java System Application Server Enterprise Edition 8.2 Administration Guide*.
Load balancer Configuration	■ Make sure the Web Server is installed. ■ Make sure the load balancer plug-in into the Web Server is installed. ■ Make sure patch checks is disabled. ■ Lower the value of the `KeepAliveQuery` parameter. The lower the value, the lower the latency is on lightly loaded systems. The higher the value, the higher the throughput is on highly loaded systems. For more information, see "Keep Alive" in *Sun Java System Application Server Enterprise Edition 8.2 Performance Tuning Guide*

TABLE 4–1 Checklist *(Continued)*

Component/Feature	Description
Java Virtual Machine Configuration	■ Initially set the minimum and maximum heap sizes to be the same, and at least one GB for each instance. ■ See Java Hotspot VM Options for more information. ■ When running multiple instances of Application Server, consider creating a processor set and bind the Application Server to it. This helps in cases where the CMS collector is used to sweep the old generation.
Configuring time-outs in Load balancer	■ Response-time-out-in-seconds - How long the load balancer waits before declaring an Application Server instance unhealthy. Set this value based on the response time of the application. If set too high, the Web Server and load balancer plug-in wait a long time before marking an Application Server instance as unhealthy. If set too low and the Application Server's response time crosses this threshold, the instance will be incorrectly marked as unhealthy. ■ Interval-in-seconds - Time in seconds after which unhealthy instances are checked to find out if they have returned to a healthy state. Too low a value generates extra traffic from the load balancer plug-in to Application Server instances and too high a value delays the routing of requests to the instance that has turned healthy. ■ Timeout-in-seconds - Duration for a response to be obtained for a health check request. Adjust this value based on the traffic among the systems in the cluster to ensure that the health check succeeds. For more information, see Chapter 5, "Configuring HTTP Load Balancing," in *Sun Java System Application Server Enterprise Edition 8.2 High Availability Administration Guide*.

TABLE 4–1 Checklist *(Continued)*

Component/Feature	Description
Configuring time-outs in HADB	■ sql_client_timeout - Wait time of SQLSUB for an idle client. For example, a client which has logged on, sends some requests, and then waits for user input. A client that has been idle for more than 30 minutes is assumed to be dead, and the session is terminated. Setting the value too low can terminate SQL sessions prematurely. Setting it too high can cause SQL sessions that are not idle but have exited to occupy resources. This in turn can prevent other SQL clients from logging on. When tuning this variable, also consider the settings of `nsessions`. If the HADB JDBC connection pool steady-pool-size is greater than max-pool-size, then idle-timeout-in-seconds can be set lower than the sql_client_timeout, so that the Application Server itself closes the connection before HADB closes the connection. Default value is 1800 s. ■ lock_timeout - Maximum time in milliseconds that a transaction waits for access to data. When exceeded, the transaction generates the error message: "The transaction timed out." Such time-outs are caused by transactions waiting for locks held by other transactions (deadlocks), and causing high server load. Do not set this value to below 500 ms. If you see the "transaction timed out" messages in the server log, then increase this value. Set the lock timeout value by adding a property to the HADB's JDBC connection pool as: `<property name=lockTimeout value="x"\>`. Default value is 5000 ms. ■ Querytimeout - Maximum time in milliseconds that HADB waits for a query to execute. If you see exceptions in the server log consistently indicating the query time out, consider increasing this value. Set this value by adding the following property to HADB's JDBC connection pool: `<property name=QueryTimeout value="x"\>`. Default value is 30 s. ■ loginTimeout - Maximum time in seconds that the client waits to login to HADB. Set this value by adding the following property to HADB's JDBC connection pool: `<property name=loginTimeout value="x"\>`. Default value is 10 s. ■ MaxTransIdle - Maximum time in milliseconds that a transaction can be idle between sending a reply to the client and receiving the next request. This can be changed by adding a property to the HADB's JDBC connection pool as: `<property name=maxtransIdle value="x"\>`. Default value is 40 s. For more information: *Sun Java System Application Server Performance Tuning Guide*.

TABLE 4–1 Checklist *(Continued)*

Component/Feature	Description
Configuring time-outs in Application Server	■ Max-wait-time-millis - Wait time to get a connection from the pool before throwing an exception. Default is 6 s. Consider changing this value for highly loaded systems where the size of the data being persisted is greater than 50 KB. ■ Cache-idle-timeout-in-seconds - Time an EJB is allowed to be idle in the cache before it gets passivated. Applies only to entity beans and stateful session beans. ■ Removal-timeout-in-seconds - Time that an EJB remains passivated (idle in the backup store). Default value is 60 minutes. Adjust this value based on the need for SFSB failover. Adjust all of these values by paying attention to HADB's JDBC connection pool setting max-wait-time-in-millis. For more information, see "Configuring the JDBC Connection Pool" in *Sun Java System Application Server Enterprise Edition 8.2 High Availability Administration Guide*.
Tune VM Garbage Collection (GC)	Garbage collection pauses of four seconds or more can cause intermittent problems in persisting session state to HADB. To avoid this problem, tune the VM heap. In cases where even a single failure to persist data is unacceptable or when the system is not fully loaded, use the CMS collector or the throughput collector. These can be enabled by adding: `<jvm-options>-XX:+UseConcMarkSweepGC</jvm-options>` This option may decrease throughput.

Index

W