Sun Java™ System

Application Server 7 System Deployment Guide

2004Q2

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

Part No: 819-1640

Contents

About This Guide

This guide describes how to create and run Java™ 2 Platform, Enterprise Edition (J2EE™ platform) applications that follow the new open Java standards model for Java™ Servlet, JavaServer Pages™ (JSP™), Enterprise JavaBeans™ (EJB™), and other J2EE components in the Sun Java™ System Application Server 7 Enterprise Edition environment.

This preface addresses the following topics:

Who Should Use This Guide

The information in this guide is intended for system administrators who want to deploy and support a large system that has high availability requirements.

This guide assumes you are familiar with the following:

- Installation of enterprise-level software products
- UNIX® operating system
- Client/server programming model
- Internet and World Wide Web
- Availability and clustering concepts

Using the Documentation

The Sun Java System Application Server Standard and Enterprise Edition manuals are available as online files in Portable Document Format (PDF) and Hypertext Markup Language (HTML).

The following table lists tasks and concepts described in the Sun Java System Application Server manuals. The manuals marked *(updated for 2004Q2)* have been updated for the Sun Java System Application Server 7 Standard and Enterprise Edition release. The manuals not marked in this way have not been updated since the version 7.0 Enterprise Edition release.

Table 1 Sun Java System Application Server Documentation Roadmap

For information about	See the following
(Updated for 7 2004Q2) Late-breaking information about the software and the documentation. Includes a comprehensive, table-based summary of supported hardware, operating system, JDK, and JDBC/RDBMS.	*Release Notes*
Sun Java System Application Server 7 overview, including the features available with each product edition.	*Product Overview*
Diagrams and descriptions of server architecture and the benefits of the Sun Java System Application Server architectural approach.	*Server Architecture*
(Updated for 7 2004Q2) How to get started with the Sun Java System Application Server product. Includes a sample application tutorial. There are two guides, one for Standard Edition and one for Enterprise Edition.	*Getting Started Guide*
(Updated for 7 2004Q2) Installing the Sun Java System Application Server Standard Edition and Enterprise Edition software and its components, such as sample applications and the Administration interface. For the Enterprise Edition software, instructions are provided for implementing the high-availability configuration.	*Installation Guide*
(Updated for 7 2004Q2) Evaluating your system needs and enterprise to ensure that you deploy Sun Java System Application Server in a manner that best suits your site. General issues and concerns that you must be aware of when deploying an application server are also discussed.	*System Deployment Guide*
Creating and implementing Java™ 2 Platform, Enterprise Edition (J2EE™ platform) applications intended to run on the Sun Java System Application Server that follow the open Java standards model for J2EE components such as servlets, Enterprise JavaBeans™ (EJBs™), and JavaServer Pages™ (JSPs™). Includes general information about application design, developer tools, security, assembly, deployment, debugging, and creating lifecycle modules. A comprehensive Sun Java System Application Server glossary is included.	*Developer's Guide*

Table 1 Sun Java System Application Server Documentation Roadmap *(Continued)*

For information about	See the following
(Updated for 7 2004Q2) Creating and implementing J2EE web applications that follow the Java™ Servlet and JavaServer Pages (JSP) specifications on the Sun Java System Application Server. Discusses web application programming concepts and tasks, and provides sample code, implementation tips, and reference material. Topics include results caching, JSP precompilation, session management, security, deployment, SHTML, and CGI.	*Developer's Guide to Web Applications*
(Updated for 7 2004Q2) Creating and implementing J2EE applications that follow the open Java standards model for enterprise beans on the Sun Java System Application Server. Discusses Enterprise JavaBeans (EJB) programming concepts and tasks, and provides sample code, implementation tips, and reference material. Topics include container-managed persistence, read-only beans, and the XML and DTD files associated with enterprise beans.	*Developer's Guide to Enterprise JavaBeans Technology*
(Updated for 7 2004Q2) Creating Application Client Container (ACC) clients that access J2EE applications on the Sun Java System Application Server.	*Developer's Guide to Clients*
Creating web services in the Sun Java System Application Server environment.	*Developer's Guide to Web Services*
(Updated for 7 2004Q2) Java™ Database Connectivity (JDBC™), transaction, Java Naming and Directory Interface™ (JNDI), Java™ Message Service (JMS), and JavaMail™ APIs.	*Developer's Guide to J2EE Services and APIs*
Creating custom NSAPI plug-ins.	*Developer's Guide to NSAPI*
(Updated for 7 2004Q2) Information and instructions on the configuration, management, and deployment of the Sun Java System Application Server subsystems and components, from both the Administration interface and the command-line interface. Topics include cluster management, the high-availability database, load balancing, and session persistence. A comprehensive Sun Java System Application Server glossary is included.	*Administration Guide*
(Updated for 7 2004Q2) Editing Sun Java System Application Server configuration files, such as the `server.xml` file.	*Administrator's Configuration File Reference*
Configuring and administering security for the Sun Java System Application Server operational environment. Includes information on general security, certificates, and SSL/TLS encryption. HTTP server-based security is also addressed.	*Administrator's Guide to Security*
Configuring and administering service provider implementation for J2EE™ Connector Architecture (CA) connectors for the Sun Java System Application Server. Topics include the Administration Tool, Pooling Monitor, deploying a JCA connector, and sample connectors and sample applications.	*J2EE CA Service Provider Implementation Administrator's Guide*
(Updated for 7 2004Q2) Migrating your applications to the new Sun Java System Application Server programming model, specifically from iPlanet Application Server 6.x and Sun ONE Application Server 7.0. Includes a sample migration.	*Migrating and Redeploying Server Applications Guide*
(Updated for 7 2004Q2) How and why to tune your Sun Java System Application Server to improve performance.	*Performance Tuning Guide*

Table 1 Sun Java System Application Server Documentation Roadmap *(Continued)*

For information about	See the following
(Updated for 7 2004Q2) Information on solving Sun Java System Application Server problems.	*Troubleshooting Guide*
(Updated for 7 2004Q2) Information on solving Sun Java System Application Server error messages.	*Error Message Reference*
(Updated for 7 2004Q2) Utility commands available with the Sun Java System Application Server; written in manpage style.	*Utility Reference Manual*
Using the Sun™ Java System Message Queue 3.5 software.	The Sun Java System Message Queue documentation at: `http://docs.sun.com/db?p=prod/s1.s1msgqu`
For information about	**See the following**
(Updated for 7 2004Q2) Late-breaking information about the software and the documentation. Includes a comprehensive, table-based summary of supported hardware, operating system, JDK, and JDBC/RDBMS.	*Release Notes*

How This Guide is Organized

This guide is divided into three chapters. Read the chapters in the order they are presented as each chapter builds on the previous one.

- Chapter 1, "Overview of Deployment" provides an introduction to deployment and discusses phases of deployment.
- Chapter 2, "Planning your Environment" discusses the steps you need to determine the environment that best suits your business needs.
- Chapter 3, "Selecting a Topology" contains examples of application server topologies, and helps you determine the topology that best suits your business needs.
- Appendix A, "Checklist for Deployment" provides a list of tasks to get started with Sun Java System Application Server.

Documentation Conventions

This section describes the types of conventions used throughout this guide:

General Conventions

The following general conventions are used in this guide:

- **File and directory paths** are given in UNIX® format (with forward slashes separating directory names). For Windows versions, the directory paths are the same, except that backslashes are used to separate directories.
- **URLs** are given in the format:

  http:// *server.domain* / *path* / *file*.html

  In these URLs, *server* is the server name where applications are run; *domain* is your Internet domain name; *path* is the server's directory structure; and *file* is an individual filename. Italic items in URLs are placeholders.
- **Font conventions** include:
  - The `monospace` font is used for sample code and code listings, API and language elements (such as function names and class names), file names, pathnames, directory names, and HTML tags.
  - *Italic* type is used for code variables.
  - *Italic* type is also used for book titles, emphasis, variables and placeholders, and words used in the literal sense.
  - **Bold** type is used as either a paragraph lead-in or to indicate words used in the literal sense.
- **Installation root directories** for most platforms are indicated by *install_dir* in this document. Exceptions are noted in "Conventions Referring to Directories" on page 10.

  By default, the location of *install_dir* on **most** platforms is:
  - Solaris 8 and 9 and Linux file-based installations:

    user's home directory`/sun/appserver7`

- Windows, all installations:

  `C:\Sun\AppServer7`

  For the platforms listed above, *default_config_dir* and *install_config_dir* are identical to *install_dir*. See "Conventions Referring to Directories" on page 10. for exceptions and additional information.

- **Instance root directories** are indicated by *instance_dir* in this document, which is an abbreviation for the following:

  default_config_dir`/domains/`*domain*`/`*instance*

- **UNIX-specific descriptions** throughout this manual apply to the Linux operating system as well, except where Linux is specifically mentioned.

Conventions Referring to Directories

By default, when using the Solaris 8 and 9 package-based or Linux RPM-based installation, the application server files are spread across several root directories. This guide uses the following document conventions to correspond to the various default installation directories provided:

- *install_dir* refers to `/opt/SUNWappserver7`, which contains the static portion of the installation image. All utilities, executables, and libraries that make up the application server reside in this location.
- *default_config_dir* refers to `/var/opt/SUNWappserver7/domains`, which is the default location for any domains that are created.
- *install_config_dir* refers to `/etc/opt/SUNWappserver7/config`, which contains installation-wide configuration information such as licenses and the master list of administrative domains configured for this installation.

Contacting Sun

You might want to contact Sun Microsystems in order to:

- Give Us Feedback
- Obtain Training
- Contact Product Support

Give Us Feedback

If you have general feedback on the product or documentation, please send this to:

http://www.sun.com/hwdocs/feedback/

Obtain Training

Application Server training courses are available at:

http://training.sun.com/US/catalog/enterprise/web_application.html/

Visit this site often for new course availability on the Sun Java System Application Server.

Contact Product Support

If you have problems with your system, contact customer support using one of the following mechanisms:

- The online support web site at:

  http://www.sun.com/supportraining/

- The telephone dispatch number associated with your maintenance contract

Please have the following information available prior to contacting support. This helps to ensure that our support staff can best assist you in resolving problems:

- Description of the problem, including the situation where the problem occurs and its impact on your operation
- Machine type, operating system version, and product version, including any patches and other software that might be affecting the problem. Here are some of the commonly used commands:
  - **Solaris:** `pkginfo, showrev`
  - **Linux:** `rpm`
  - **All:** `asadmin version --verbose`
- Detailed steps on the methods you have used to reproduce the problem
- Any error logs or core dumps
- Configuration files such as:

- *instance_dir*/config/server.xml
- a web application's web.xml file,
  when a web application is involved in the problem

- For an application, whether the problem appears when it is running in a cluster or standalone

Chapter 1

Overview of Deployment

This chapter describes what you need to know to set up your Sun Java™ System Application Server, Standard and Enterprise Edition the way that best meets your requirements.

This chapter contains the following sections:

About Deployment

Successful deployment of complex applications on the Application Server requires that you consider practical aspects of the environment. In general, you should begin by assessing your goals for performance and availability. You should then plan the hardware, network, and resource configuration accordingly.

Here are the important goals that you should consider while planning the deployment:

- Throughput
- Response time
- Availability

You gather information related to these goals and then analyze it to establish a processing threshold for your site.

In considering performance, consider both—application server instances and the High Availability Database (HADB).

The HADB uses the patented Always-On technology and works as the persistence store to provide high availability for web applications. It offers an ideal platform for delivering all types of session state persistence within an enterprise application server environment. For more information on configuring HADB, see *Sun Java™ System Application Server Administration Guide.*

Throughput

Throughput is the number of requests that Sun Java System Application Server can service in a given time period. For example requests per minute. You should estimate the maximum number of operations and transactions that the system needs to perform under peak load conditions. It is also useful to determine the operations and transactions per minute under steady state (typical) load conditions. This will help you to determine the network bandwidth needed, the number of application server instances required, and the number of HADB nodes required.

You should also consider plans to increase capacity in the future.

Response Time

You should determine the acceptable response time from the system under heavy load. This has a direct bearing on hardware capacity planning.

Availability

Will your system be running 24 x 7? If there is a failure in the system, will your users notice it? Do you have a subset of applications that need to be available all the time whereas other applications will run only periodically? The answers to these questions determine your availability needs. You will have to build redundancy into the system to meet availability needs and avoid single points of failure.

Phases of the Deployment Process

The deployment process primarily comprises the following three phases, each one building on the previous one.

- Planning Your Environment

- Selecting a Topology
- Running Tests

Planning Your Environment

In the first phase of planning your deployment, determine how the Application Server fits into your overall enterprise. Central to planning your environment is the assessment of the goals discussed in “About Deployment” on page 13. You should establish performance goals related to throughput and response time. You also determine your availability goals.

Based on the performance and availability goals, you consider the network requirements and the infrastructure requirements including hardware, storage, and network requirements.

You may realize during this process that you should change the structure and components of your network to accommodate the needs of your Application Server. If your network structure cannot be changed at this time, use the environment planning process to determine how you can best deploy Application Server to fit in with your existing network.

For more details on this phase of the planning process, see Chapter 2, “Planning your Environment.”

Selecting a Topology

Once you have determined the performance, availability, network, and infrastructure requirements, you then select a topology that best meets your performance needs. A topology is the schematic arrangement of Application Server components and the communication flow between these components. The two recommended topologies (and their variations) are:

- Co-located: The application server instance and the HADB node are on the same machine.
- Separate Tier: The application server instance and the HADB node are on different machines.

For more details on these topologies, see Chapter 3, “Selecting a Topology.”

Running Tests

Once you configure the Application Server, you should deploy a representative sampling of applications and run tests to check whether your performance goals are met. If you are not able to reach your stated performance goals, use this phase to identify bottlenecks, fine-tune the system, and improve performance.

As implementation of this phase is completely dependent on your particular environment, it is not covered in this guide.

Understanding Session Persistence

J2EE applications typically have significant amounts of session state data. A web shopping cart is the classic example of 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 session state. Both HTTP sessions and stateful session beans (SFSBs) have session state data.

While session state is not as important as transactional state stored in a database, preserving 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.

Sun Java System Application Server supports the following types of session persistence stores:

- High Availability (HA)
- Memory
- File

When you set the persistence type to HA, Application Server uses HADB as the persistence store for both HTTP and SFSB sessions. With Memory persistence, the state is always kept in memory and does not survive failure. 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 enabling SFSB checkpointing, see the *Sun Java System Application Server Developer's Guide to Enterprise JavaBeans Technology*.

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

The persistence settings can be defined for each application server instance. However, all application server instances in a single cluster must have the same persistence configuration. If you have deployed more than one Application Server cluster, it is not necessary for all clusters to have the same persistence configuration settings.

For more information on configuring session persistence and its effect on performance, see "HTTP Session Persistence Frequency".

NOTE Use the command `cladmin` to ensure that the session persistence settings are homogeneous for all instances in a cluster. For more information on using the `cladmin` command, see *Sun Java System Application Server Administration Guide*.

Chapter 2

Planning your Environment

Planning your environment is one of the first phases of deployment. In this phase, you should first decide your performance and availability goals, and then accordingly make decisions about the hardware, network, and storage requirements.

The main objective of this phase is to determine the environment that best meets your business requirements.

This chapter contains the following sections:

Introducing HADB

The High Availability Database (HADB) provides a highly available persistence store for the Application Server for HTTP sessions, stateful session beans, and remote references of EJB look-ups on the RMI/IIOP path.

This section contains the following topics:

Overview

J2EE applications' need for session persistence was previously described in the section "Understanding Session Persistence" on page 16. The Application Server uses the HADB as a highly available session store. The HADB is included with the Application Server Enterprise Edition, but in deployment can be run on separate 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.
- Starting or stopping server instances does not affect other servers or their availability.
- The 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 the 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" and "Separate Tier Topology" in Chapter 3, "Selecting a Topology."
- As state management requirements change, you can add resources to the HADB system without affecting existing clusters or their applications.

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

For the HADB hardware and network system requirements, see the *Sun Java System Application Server Release Notes.* For additional system configuration steps required for HADB, see *Sun Java System Application Server Administration Guide.*

System Requirements

The recommended system requirements for the HADB hosts are the following:

- At least one CPU per HADB node.

- At least 512 MB memory per node.
- Network configuration requirements (see *Sun Java System Application Server Installation Guide*).

For additional requirements for very high availability, see "Mitigating Double Failures" on page 25.

HADB Architecture

HADB is a distributed system comprising pairs of nodes, which are divided into two data redundancy units (DRUs). Each node consists of the following:

- 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 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.

For HADB hardware requirements, see the *Sun Java System Application Server Release Notes*.

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 can 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 numerous child processes. The parent process, called the node supervisor (NSUP) is started by the management agent and 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 like sorts and joins.

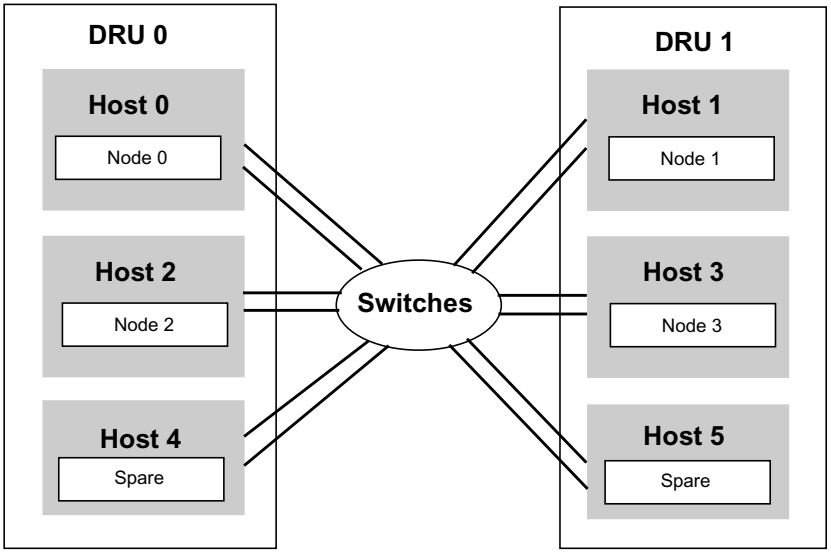
- SQL shared memory server process (SQLSHM) that maintains the SQL dictionary cache.
- SQL server process (SQLC), that receives client queries, compiles them into local HADB instructions, sends the instructions to TRANS, receives the results and conveys them to the client. Each node has one main SQL 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

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.

Figure 2-1 shows an example HADB architecture with six nodes: four active nodes and two spare nodes. Nodes 0 and 1 are a mirror node 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 "System Requirements" on page 20).

Figure 2-1 Sample HADB Configuration with Double Interconnects



NOTE Machines that host HADB nodes must be added in pairs, with one machine in each DRU.

HADB achieves high availability by replicating data and services. The data replicas on mirror 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 active node in a DRU has a mirror node on the other DRU. The mirror node pair contains the replicas of the same data fragment. Since a spare node does not have data, it does not have a mirror node. Due to the mirroring, each DRU contains a complete copy of the database.When a mirror node takes over the functions of a failed node, it has to perform double the 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 non-mirror 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 25.

Spare Nodes

When a node fails, its mirror node takes over. 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 spare nodes, since the original spare nodes are now active.

Spare nodes are not mandatory, 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, you should have a spare machine with enough Application Server instances and HADB nodes to replace any machine that becomes unavailable.

Examples of Spare Node Configurations

The following examples illustrate using spare nodes in HADB deployments. There are two fundamental categories of deployment topology: *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 about deployment topologies, see Chapter 3, "Selecting a Topology."

Example: Co-located Spare Node Configuration

Suppose you have a co-located deployment, with four Sun Fire™ V480 servers where each server has one Application Server instance and two HADB data nodes.

In this scenario, you should allocate two more servers as spare machines (one machine per DRU). Each spare machine should run one application server instance and two spare HADB nodes.

Example: Separate-tier Spare Node Configuration

Suppose you have a separate-tier deployment 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 should have as many instances as the other machines in the Application Server instances tier. Similarly, the spare machine for the HADB tier should have as many HADB nodes as the other machines in the HADB tier.

For more information about the co-located and the separate tier deployment topologies, see Chapter 3, “Selecting a Topology.”

Mitigating Double Failures

HADB’s built-in data replication enables it to tolerate failure of a single node or DRU. However, by default, HADB will not survive a double failure, when a mirror node pair or both DRUs fail. In such cases, HADB becomes unavailable.

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

- **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 as shown in Figure 2-1.

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

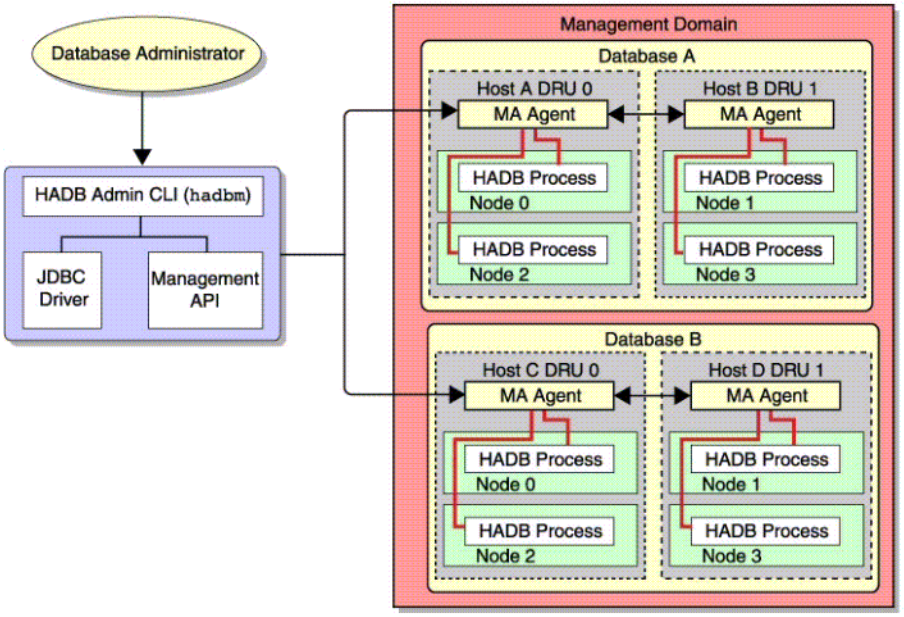
HADB Management System

The HADB management system provides built-in security and facilitates multi-platform management. As illustrated in Figure 2-2, the HADB management architecture contains the following components:

- Management Client
- Management Agent
- Management Domains
- Repository

As shown in Figure 2-2, 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 general there must be an even number of machines to form the DRU pairs. Thus, a domain contains many management agents.

Figure 2-2 HADB Management Architecture



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.

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 continuously—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 the *Sun Java System Application Server 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 the *Sun Java System Application Server 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 instance connects to a management agent by specifying the address and port number of the agent. Once connected, the management client sends commands to the 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.

Use the host operating system 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 operating system restarts the management agent if the agent fails or the operating system 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*) that is generated when you create or the domain or add an agent to it. The domainkey provides a unique identifier for the domain, which is crucial because management agents communicate using multicast. You can set up an HADB management domain to match 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.
- **hadbm extenddomain**: adds hosts to an existing management domain.
- **hadbm deletedomain**: removes a management domain.
- **hadbm reducedomain**: removes hosts from the management domain.
- **hadbm listdomain**: lists all hosts defined in the management domain.

For more information on these commands, see the *Sun Java Application Server Reference Manual* (or the corresponding man pages).

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 the nearest integer) must be running to make a change to the repository.

If you cannot perform some management commands because a majority of the hosts in a domain are unavailable (for example due to hardware failures), use the `hadbm disablehost` command to remove failed hosts from the domain until you have a majority. For more information on this command, see the *Sun Java System Application Server Utility Reference Guide*.

Setup and Configuration Roadmap

Follow this procedure to setup and configure your Application Server system for high availability:

1. Determine your performance and QoS requirements and goals, as described later in this chapter.
2. Size your system, as described in "Design Decisions" later in this chapter. In particular, determine:
    - 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," that is, whether you are going to install HADB on the same host machines as Application Server or on different machines.
4. Install Application Server instances
5. Create domains and clusters
6. Install and configure your web server software.
7. Install the Load Balancer Plug-in.
8. Set up and configure load balancing
9. Set up and configure HADB nodes and DRUs
10. Configure Application Server Web container and EJB container for HA session persistence.
11. Deploy applications and configure them for high availability and session failover.
12. Configure JMS cluster for failover. For more information, see the *Sun Java System Message Queue Administration Guide*.

Establishing Performance Goals

As explained in Chapter 1, "Overview of Deployment," one of your main goals is to maximize performance. This essentially translates into maximizing throughput and reducing response time.

Beyond these basic goals, you should establish specific goals by determining the following:

- What capacity of requests, or throughput, can the system support?
- How many concurrent users can the system support?

- What is an acceptable average response time for requests submitted by your users?
- What is the average think time between requests?

These factors are interrelated. If you know the answer to any three of these four factors, you can calculate the fourth.

Some of the metrics described in this chapter can be calculated using a remote browser emulator (RBE) tool, or web site performance and benchmarking software, that simulates your enterprise's web application activity. Typically, RBE and benchmarking products generate concurrent HTTP requests and then report back the response time and 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 Sun Java System Application Server.

This section describes the following topics:

Estimating Throughput

Throughput, as measured for application server instances and for HADB, has different implications.

A good measure of the throughput for Application Server instances is the number of requests precessed per minute. A good measure of throughput for the HADB is the number of requests processed per minute by HADB, and the session size per request. The session size per request is important because the size of session data stored varies from request to request.

For more information session persistence, see Chapter 1, "Overview of Deployment."

Estimating Load on Application Server Instances

Consider the following factors to estimate the load on application server instances:

- Calculating Maximum Number of Concurrent Users
- Calculating Think Time
- Calculating Average Response Time
- Calculating Requests Per Minute

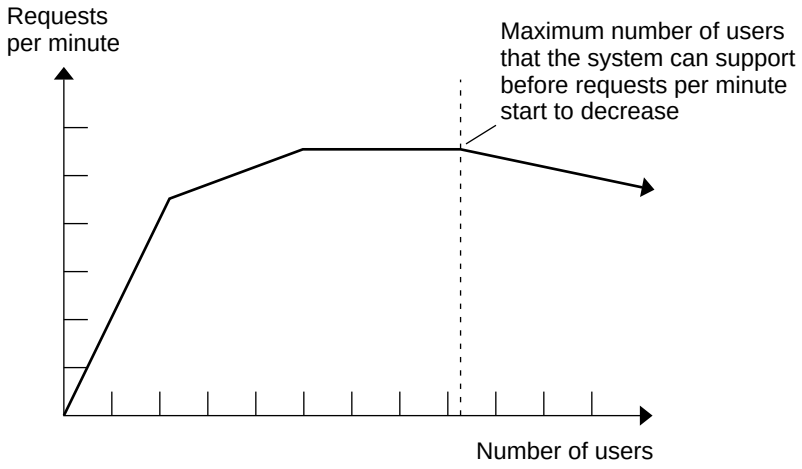
Calculating Maximum Number of Concurrent Users

A user runs a process (for example through a web browser) that periodically sends requests from a client machine to Application Server. When estimating the number of concurrent users, include all users currently active. A user is considered active as long as the session that user is running is active (for example, the session has neither expired nor terminated).

A user is concurrent for as long as the user is on the system as a running process submitting requests, receiving results of requests from the server, and viewing the results.

Eventually, as the number of concurrent users submitting requests increases, requests processed per minute begins to decline (and the response time begins to increase). The following diagram illustrates this situation.

Figure 2-3 Performance Pattern with Increasing Number of Users.



You should identify the point at which adding more concurrent users reduces the number of requests that can be processed per minute. This point indicates when performance starts to degrade.

Calculating Think Time

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

Determining the typical duration of think time is important. You can use the duration to calculate more accurately the number of requests per minute, as well as the number of concurrent users your system can support. Essentially, when a user is on the system but not submitting a request, a gap opens for another user to submit a request without altering system load. This implies that you can support more concurrent users.

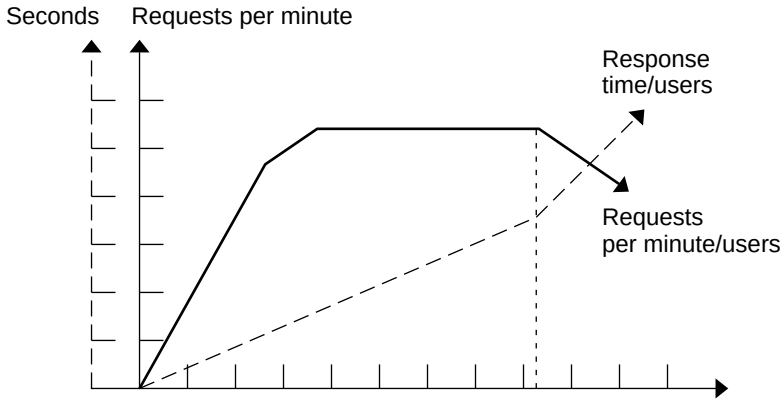
Calculating Average Response Time

Response time refers to the amount of time it takes for results of a request to be returned to the user. The response time is affected by a number of factors including 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, you should base your 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 your system increases, response time starts to increase as well, even though the number of requests per minute declines, as the following diagram illustrates:

Figure 2-4 Response Time with Increasing Number of Users



A system performance graph similar to Figure 2-4 on page 33, indicates that after a certain point (point A in this diagram), 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 Figure 2-4 on page 33, point A represents peak load, that 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), you can more accurately calculate response time using maximum number of users and requests per minute.

To determine response time at peak load, use the following formula:

```
Response time = (concurrent users / requests per second) - think time in
seconds
```

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

Example Calculation of Response Time

For example, if the following conditions exist:

- Maximum number of concurrent users that your system can support at peak load is 5,000.
- Maximum number of requests the system can process at peak load is 1,000 per second.
- Average think time equals 3 seconds per request.

```
Response time = (5000/ 1000) - 3 seconds think time
```

Therefore, the response time is 2 seconds.

After you have calculated your system's response time, particularly at peak load, decide what is an acceptable response time for your enterprise. Response time, along with throughput, is one of the main factors critical to Sun Java System Application Server performance. Improving the response time should be one of your goals.

If there is a response time beyond which you do not want to wait, and performance is such that you get response times over that level, then work towards improving your response time or redefine your response time threshold.

Calculating 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 at that time, you can determine requests per minute. Typically, you start by knowing how many concurrent users are on your system.

For example, after running a few web site performance calculation software, you conclude that the average number of concurrent users submitting requests on your online banking web site is 3,000. This number is dependent on the number of users who have signed up to be members of your online bank, their banking transaction behavior, the times 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 are 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.

Essentially, you should experiment with the requests per minute and response time thresholds that is acceptable as a starting point for fine-tuning system performance. Thereafter, decide which areas of your system you want to adjust.

The formula for obtaining the requests per second is as follows:

```
requests per second = concurrent users / (response time in seconds +
think time in seconds)
```

Example Calculation of Requests per Second

For example, if the following conditions exists:

- Concurrent users equals 2,800.

- Average response time equals 1 second per request.
- Average think time equals 3 seconds.

```
Requests per second = 2800 / (1+3)
```

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

Estimating Load on HADB

To calculate load on HADB, consider the following factors:

- HTTP Session Persistence Frequency
- HTTP Session Size and Scope
- SFSB Checkpointing

For more information on configuring session persistence, see *Sun Java System Application Server 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 it leads to high traffic to the HADB.
- **time-based**: the session is stored at the specified time interval. This option reduces the traffic to the HADB, but does not guarantee that the session information will be up to date.

Table 2-1 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 certain 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.

You can further 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.

Table 2-2 summarizes the advantages and disadvantages of the persistence scope options.

Table 2-2 Comparison of Persistence Scope Options

Persistence Scope Option	Advantage(s)	Disadvantage(s)
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.	1. As the percentage of session state that gets modified for a given request grows to around 60%, the throughput and the response time degrade. In such cases, the performance gets worse than the `session` or `modified-session` persistence scope because of the overhead of splitting the attributes into separate records.

SFSB 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.

Design Decisions

Depending on the load on the application server instances, the load on the HADB, and the failover requirements, you should make the following decisions at this stage:

- Number of Application Server Instances Required
- Number of HADB Nodes Required
- Number of HADB Hosts
- HADB Storage Capacity
- Designing for Peak Load Compared to Steady State Load

Number of Application Server Instances Required

To determine the number of applications server instances needed, evaluate your environment on the basis of the factors explained in "Estimating Load on Application Server Instances" on page 31. Each application server instance can use more than one Central Processing Unit (CPU) and should have at least one CPU allocated to it.

Number of HADB Nodes Required

As a general guideline, you should plan to have one HADB node for each CPU in your 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 such as, 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}, using the following formula:

   $V_{data} = n_{users} . s$
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:

   $R_{dt} = n_{users} . s . NTPS$
4. Determine the number of nodes based on data volume considerations, $N_{NODES,}$ using the following formula:

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

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

Number of HADB Hosts

Determine the number of 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 based on data transfer considerations, follow this procedure:

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

2. 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. The number of hosts needed to accommodate this data is determined by using the following formula:

   $N_{HOSTS} = 4 . R_{dt} / R_{max}$

   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 you exceed the network capacity. 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.

For example, suppose the expected session data is X MB. The HADB replicates the data on mirror nodes, and therefore needs 2X MB of storage.Further, the HADB uses indexes to enable fast access to data. An additional 2X MB is required (for both nodes together) for indexes (assuming a less than 100% fillings rate). This implies that a storage capacity of 4X is required.Therefore, the expected storage capacity needed by the HADB 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 the expected data volume.

Additionally, HADB uses disk space for internal use as follows:

- Space for temporary storage of log buffer. This space is four times the `logBufferSize`. The `logBufferSize` is the size of the log buffer, which keeps track of operations related to data.

  NOTE The default value of logBufferSize is 48 MB.

- Space for internal administration purpose. This space is one percent of the storage device size.

For more information, see *Sun Java System Application Server Administration Guide* and *Sun Java System Application Server Performance Tuning Guide.*

The following table summarizes the HADB storage space requirements for a session data of X MB.

Table 2-3 HADB Storage Space Requirement for Session Size of X MB

Condition	HADB Storage Space Required
Addition or removal of HADB nodes while online is *not* required.	(4X MB) + (4*logBufferSize) + (1% of Device Size)
Addition or removal of HADB nodes while online is required.	(8X MB) + (4*logBufferSize) + (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, see *Sun Java System Application Server Troubleshooting Guide.*

Setting Data Device Size

Use the following command to set the size of the data devices of the HADB:

```
hadbm set TotalDatadeviceSizePerNode
```

The `hadbm` command restarts all the nodes, one by one, for the change to take effect. For more information on configuring the HADB, see *Sun Java System Application Server Administration Guide.*

NOTE The current version of the `hadbm` command does not add data devices to a running HADB database.

Designing for Peak Load Compared to Steady State Load

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

If you design for peak load, you must deploy a system that can sustain the expected maximum load of users and requests without a degradation in response time. This implies that your 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 may mean that you are spending on resources that will be idle for a significant amount of time.

If you design for steady state load, then you don't have to deploy a system with all the resources required to handle the server's expected peak load. However a system designed to support steady load will have slower response time when peak load occurs.

Frequency and Duration of Peak Load

The factor that may affect whether you want to design for peak load or for steady state is how often your system is expected to handle the peak load. If peak load occurs several times a day or even per week, you may decide that this is enough time to warrant expanding capacity to handle this load. If the system operates at steady state 90 percent of the time, and at peak only 10 percent of the time, then you may prefer to deploy a system designed around steady state load.

This implies that your 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 your system (should this be required to handle peak load).

Planning the Network Configuration

When planning how to integrate Sun Java System Application Server into your network for optimal performance, you should estimate the bandwidth requirements and plan your network in such a way that it can meet your performance requirements.

The following topics are covered in this section:

- Estimating Bandwidth Requirements
- Calculating Bandwidth Required

Estimating Bandwidth Requirements

When you decide on the desired size and bandwidth of your network, first determine your 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.

TIP	At all times consult network experts at your site about the size and type of network components you are considering.

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 your users occasionally do not have immediate Internet access when accessing applications deployed on Application Server. Of that five percent, determine how many users retry access after the first attempt. Again, not all of those users may 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.

To ensure optimal access during times of peak load, start by verifying that your Internet service provider (ISP) has a backbone network connection that can reach an Internet hub without degradation.

Calculating Bandwidth Required

Based on the calculations you made in “Establishing Performance Goals” on page 29, you should determine the additional bandwidth required for deploying Sun Java System Application Server at your site.

Depending on your method of access (T-1 lines, ISDN, and so on), you can calculate the amount of increased bandwidth you require to handle your estimated load. For example, suppose your site uses T-1 or higher-speed T-3 links for Internet access. Given their bandwidth, you can estimate how many lines you will need on your network, based on the average number of requests generated per second at your site and the maximum peak load. You can calculate these figures using a web site analysis and monitoring-tool.

Example Calculation of Bandwidth Required

A single T-1 line can handle 1.544 Mbps. Therefore, a network of four T-1 lines carrying 1.544 Mbps each 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 client requests for pages per second.

With a 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, you will have to increase the bandwidth accordingly.

Estimating Peak Load

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

Example Calculation of Peak Load

If 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, you should 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 your 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 tiers, you can achieve a performance improvement by keeping 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.

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 Sun Java System Application Server and the HADB nodes, as well as among other resources such as HADB databases that are hosted on other machines.

Network Settings for HADB

HADB uses UDP multicast and hence 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 all the HADB nodes on the same network. Application server instances may be on a different sub network.

Use the following suggestions to make HADB work optimally in the network:

- Use switched routers so that each network interface has a dedicated 100 Mbps or better Ethernet channel.
- If you are 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

Availability must be planned according to the application and customer requirements.

There are two ways to achieve high availability:

- Adding Redundancy to the System
- Using Multiple Clusters to Improve Availability

Adding Redundancy to the System

One way to achieve high availability is to add redundancy to the system—redundancy of hardware and software. When one unit fails, the redundant unit takes over. This is also referred to as fault tolerance.

In general, to achieve high availability, you should determine and remove every possible point of failure in the system.

This section discusses the following topics:

- Identifying Failure Classes
- Using Redundancy Units to Improve Availability
- Using Spare Nodes to Improve Fault Tolerance
- Planning Failover Capacity

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 and catastrophes.

Duplicated system processes tolerate single system process failures. Duplicated machines tolerate 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 and by keeping them in separate geographical locations, natural catastrophes like earth quake in a location can be tolerated.

When planning availability, you should determine the failure classes covered by the system.

Using Redundancy Units to Improve Availability

To improve availability, HADB nodes are always used in Data Redundancy Units (DRUs) as explained in "Introducing HADB" on page 19.

Using Spare Nodes to Improve Fault Tolerance

The use of spare nodes as explained in "Spare Nodes" on page 23 improves fault tolerance. Although spare nodes are not mandatory, their use is recommended for maximum availability.

Planning Failover Capacity

Failover capacity planning implies deciding how many additional servers and processes you need to add to Sun Java System Application Server installation 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, we recommended that you add spare machines running Application Server instances to your existing Application Server installation. For example, assume you have a system with two machines running one Application Server instance each. Together, these machines can 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 would not be served.

Using Multiple Clusters to Improve Availability

To improve availability, instead of using a single cluster, you should group the application server instances into multiple clusters. This way, you can perform online upgrades for clusters (one by one) without loss of service.

For more information on setting up multiple clusters and using multiple clusters to perform online upgrades without loss of service, see *Sun Java System Application Server Administration Guide.*

Chapter 3

Selecting a Topology

After estimating the factors related to performance as explained in Chapter 2, "Planning your Environment," you should decide the *topology* that you will use to deploy Sun Java™ System Application Server 7 Enterprise Edition. A topology is the schematic arrangement of Application Server components (machines, application server instances, and HADB nodes), and the communication flow between these components.

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 the following:

- Common Requirements
- Co-located Topology
- Separate Tier Topology
- Comparison of Topologies
- Determining Which Topology to Use

Common Requirements

The following topics in this section describe the requirements that are common to both the topologies:

- General Requirements
- HADB Nodes and Machines
- Load Balancer Configuration

General Requirements

Both topologies must meet the following general requirements:

- Machines that host HADB nodes must be provided in pairs.
- Each DRU must have the same number of machines. You must 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, which is used to store all persisted information in the HADB.
- Machines that host the HADB nodes must run the same operating system. These machines should be as identical as possible 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 *Sun Java System Application Server Administration Guide.*
- The machines hosting the application server instances should 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 you have machines of different classes hosting instances, then the load will not be balanced in the most optimum way across these machines.
- Each DRU should preferably have a separate Uninterruptible Power Supply (UPS).

HADB Nodes and Machines

Each DRU contains a complete copy of the data in the 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 your data is lost. For this reason, it is important that you do not set up your system in a way that both DRUs can be impacted by a single failure, such as a power failure or a 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 and thereby balance the 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 (failures related to disk, memory, CPU, power, operating system crashes and so on).

Load Balancer Configuration

Both the topologies comprise application server instances in a cluster. These instances persist session information to the HADB. You must 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 it, see *Sun Java System Application Server 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.

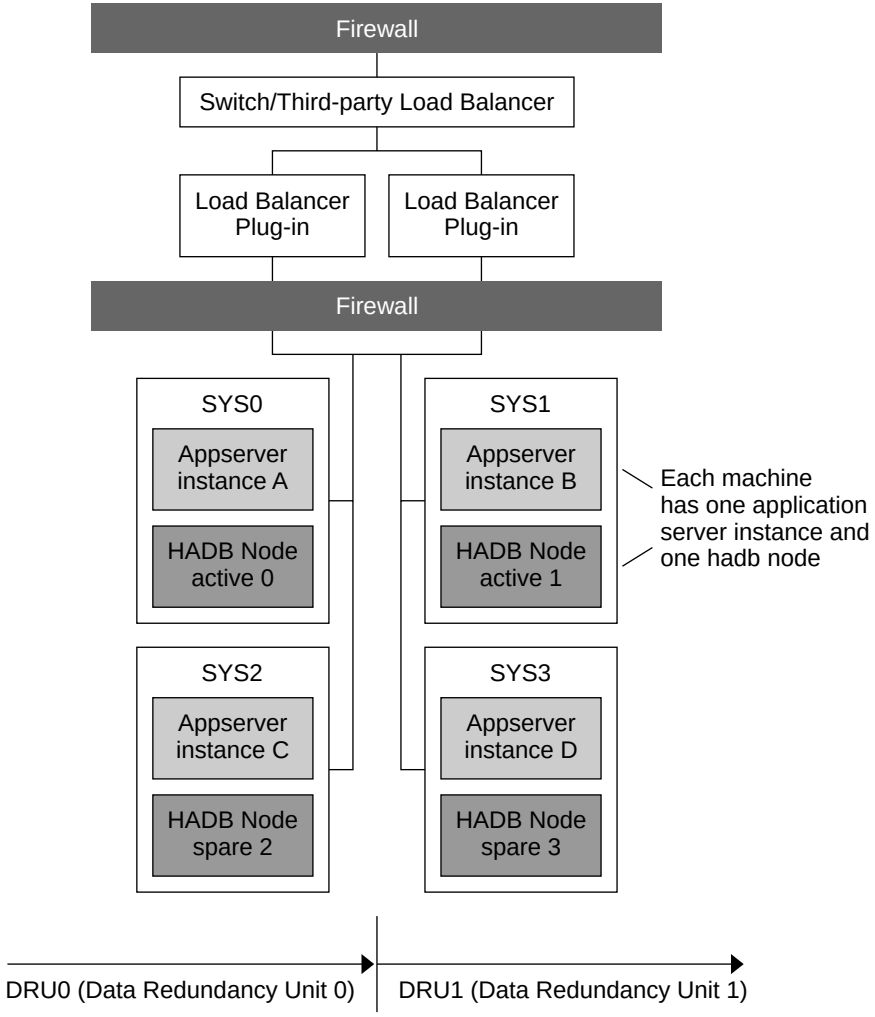
A minimum of two machines are required for this topology. To improve throughput, more machines can be added in pairs.

NOTE The co-located topology is a good fit for large, Symmetric Multiprocessing (SMP) machines as you can take full advantage of the processing power of these machines.

Example of Co-located Topology

Figure 3-1 shows an example of co-located topology.

Figure 3-1 Example of Co-located Topology



Application Server instances are hosted as follows:

- Application Server instance A is hosted on machine SYS0,

- Application Server instance B hosted is on machine SYS1,
- Application Server instance C is hosted on machine SYS2, and
- Application Server instance D is hosted on machine SYS3.

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.

NOTE The configuration settings described in this document assume that the host names for the machines correspond to the machine names as described in the topologies. For example, for the reference co-located topology, the host names are SYS0, SYS1, SYS2, and SYS3. This applies to both the topologies (and their variations).

Configuration Settings for Reference Co-located Topology

Use the `clsetup` command for configuring the cluster (as part of the cluster configuration, the `clsetup` command creates an HADB database and sets up the JDBC connection pool and the JDBC resource for the HADB). For information on using `clsetup`, see *Sun Java System Application Server Installation Guide*.

The `clsetup` command uses the following input files:

- `clresource.conf`: This is the resource configuration file for the application server instances and the HADB.
- `clinstance.conf`: This file contains information about application server instances.

NOTE Make changes to these input files as described in the subsequent sections before you run `clsetup` command.

Changes to clresource.conf File

For configuration related to the topologies described in this guide, the following properties should be changed in the `clresource.conf` file:

- `hosts`: A comma separated list of host names for the machines that host HADB active nodes. For each HADB active node, include the host name of the machine. Therefore, if a machine hosts two HADB nodes, the host name of the machine must appear twice.

- steadypoolsize: The value of the steadypoolsize property is calculated using the following formula:

  8*(number of HADB nodes)/(number of application server instances)

  If the resulting number is a decimal, round it off to the next even number.

- maxpoolsize: The value of the maxpoolsize property is calculated using the following formula:

  16*(number of HADB nodes)/(number of application server instances)

  If the resulting number is a decimal, round it off to the next even number.

NOTE

- The HADB nodes include both active and spare nodes.
- The description and the calculation of values described here apply to both the topologies (and their variations).

Table 3-1 describes the changes needed to the clresource.conf file for the reference co-located topology. The left column lists the section in the file where the property to be changed is listed, the middle column lists the property name, and the right column lists the value of the property

Table 3-1 Changes Needed to the clresource.conf File for Reference Co-located Topology

Section of clresource.conf File in Which the Property Appears	Property Name	Value
HADBINFO	hosts	SYS0,SYS1,SYS2,SYS3
JDBC_CONNECTION_POOL	steadypoolsize	8
JDBC_CONNECTION_POOL	maxpoolsize	16

Changes to clinstance.conf File

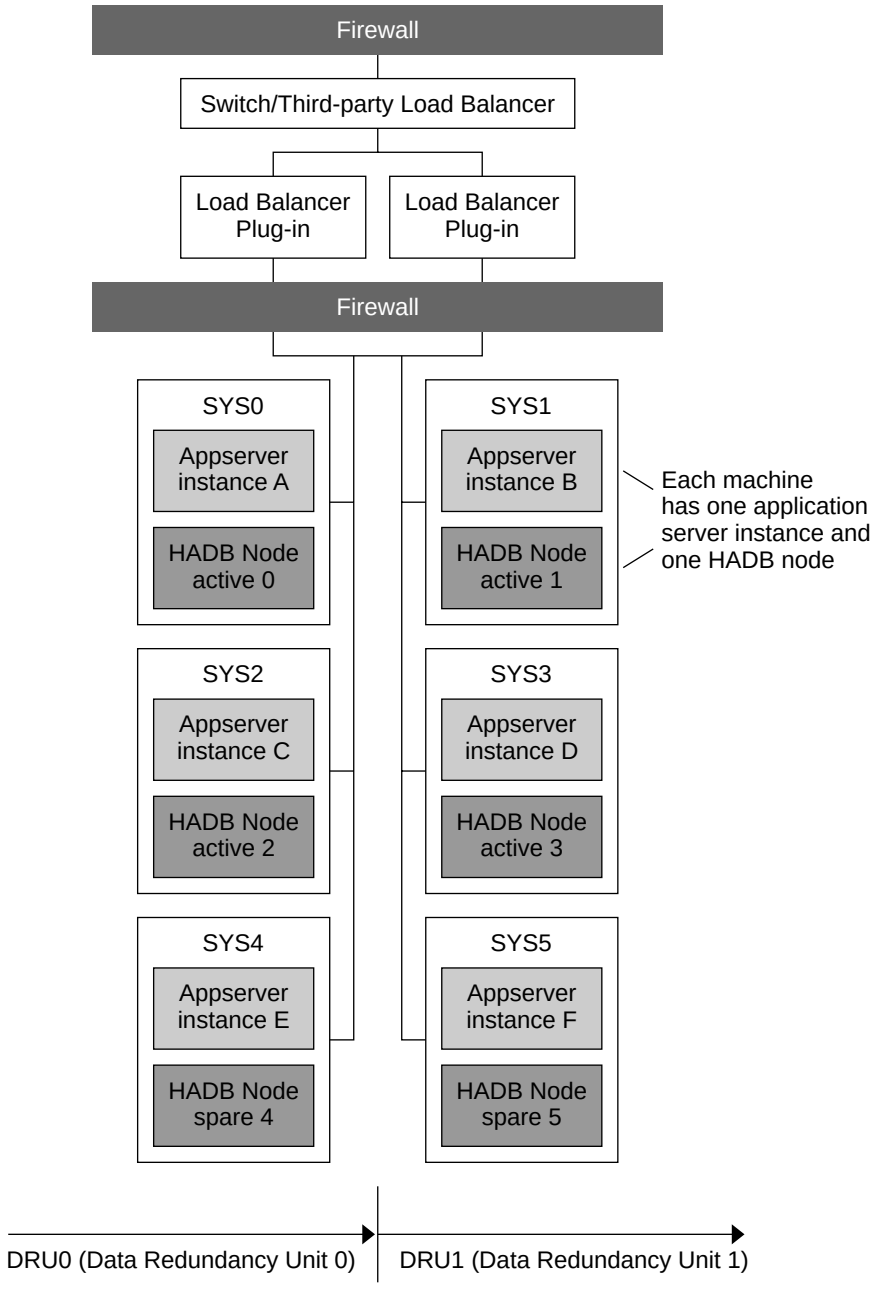
In the clinstance.conf file, include information for each instance. For detailed information, see *Sun Java System Application Server Installation Guide*. This applies to both topologies and their variations.

Variation of Co-located Topology

For better scalability and throughput, you can increase the number of application server instances and HADB nodes by adding more machines.

For example, you can add two machines, each with one application server instance and one HADB node. Make sure that you add the HADB nodes in pairs, assigning one node for each DRU. This configuration is shown in Figure 3-2.

Figure 3-2 Variation of Co-located Topology



In this variation, the machines SYS4 and SYS5 have been added to the sample co-located topology described in "Example of Co-located Topology" on page 52.

Application Server instances are hosted as follows:

- Application Server instance A is hosted on machine SYS0,
- Application Server instance B is hosted on machine SYS1,
- Application Server instance C is hosted on machine SYS2,
- Application Server instance D is hosted on machine SYS3,
- Application Server instance E is hosted on machine SYS4, and
- Application Server instance F is hosted on machine SYS5.

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

- DRU0 comprises the 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.

Configuration Settings for Variation to the Reference Co-located Topology

Make the changes as described in the subsequent sections before you run the `clsetup` command.

Changes to clresource.conf File

Table 3-2 describes the changes needed to the `clresource.conf` file for the variation to the reference co-located topology as described in this section. The left column lists the section in the file in which the property to be changed is listed, the middle column lists the property name, and the right column lists the value of the property. For more information on the values of these properties, see "Changes to clresource.conf File" on page 53.

Table 3-2 Changes Needed to clresource.conf File for Variation to Reference Co-located Topology

Section of clresource.conf File in Which the Property Appears	Property Name	Value
HADBINFO	`hosts`	SYS0,SYS1,SYS2,SYS3,SYS4,SYS5

Table 3-2 Changes Needed to clresource.conf File for Variation to Reference Co-located Topology

JDBC_CONNECTION_POOL	`steadypoolsize`	8
JDBC_CONNECTION_POOL	`maxpoolsize`	16

Changes to clinstance.conf File

In the `clinstance.conf` file, include the information for each instance. For more information, see *Sun Java System Application Server Installation Guide.*

Separate Tier Topology

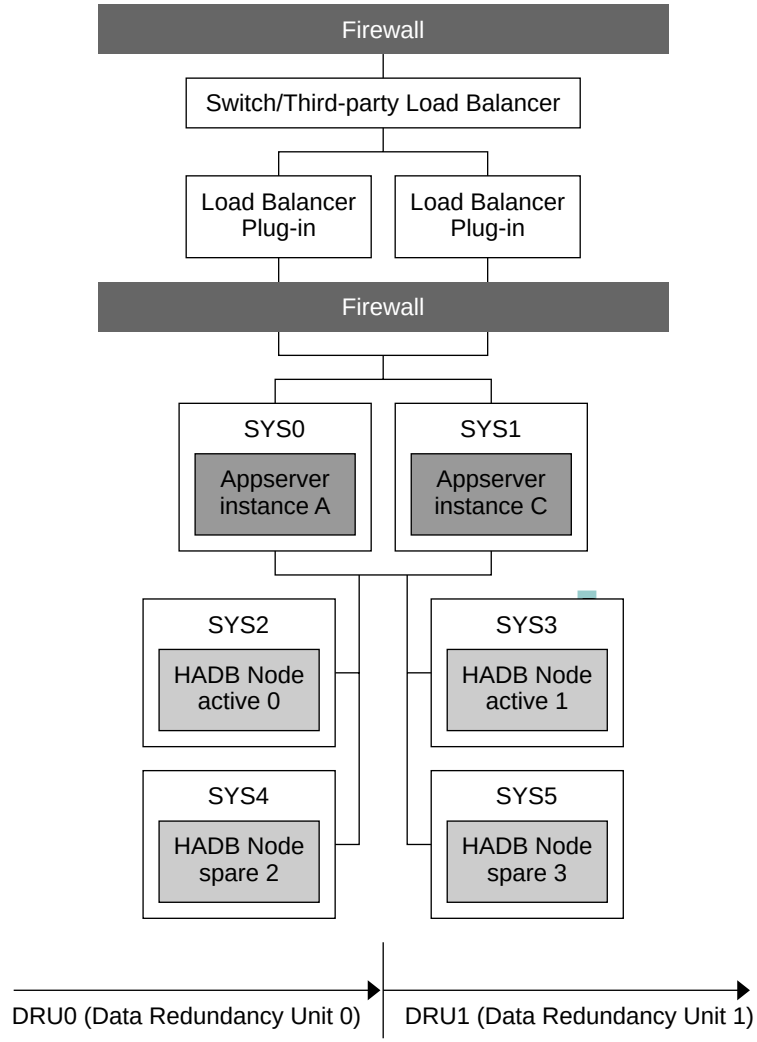
In this topology, Application Server instances and the HADB nodes are on different machines (hence the name *separate tier*).

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

Sample Configuration

Figure 3-3 shows an example of the separate tier topology.

Figure 3-3 Reference Separate Tier Topology



In this reference topology, the Application Server instance A is hosted on machine SYS0 and the Application Server instance B is hosted on the machine SYS1.

These two instances form a cluster that persists session information to DRUs as follows:

- DRU0 comprises two machines: SYS2 and SYS4. The 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. The 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 becomes unavailable, the complete data for any DRU continues to be available on other machines.

Configuration Settings for Reference Separate Tier Topology

Make the changes as described in the subsequent sections to these input files before you run the `clsetup` command.

Changes to clresource.conf File

Table 3-3 describes the changes needed to the `clresource.conf` file for the reference separate tier topology. The left column lists the section in the file in which the property to be changed is listed, the middle column lists the property name, and the right column lists the value of the property. For more information on the values of these properties, see "Changes to clresource.conf File" on page 53.

Table 3-3 Changes Needed to clresource.conf File for Reference Separate Tier Topology

Section of clresource.conf File in Which the Property Appears	Property Name	Value
HADBINFO	hosts	SYS2,SYS3,SYS4,SYS5
JDBC_CONNECTION_POOL	steadypoolsize	16
JDBC_CONNECTION_POOL	maxpoolsize	32

Changes to clinstance.conf File

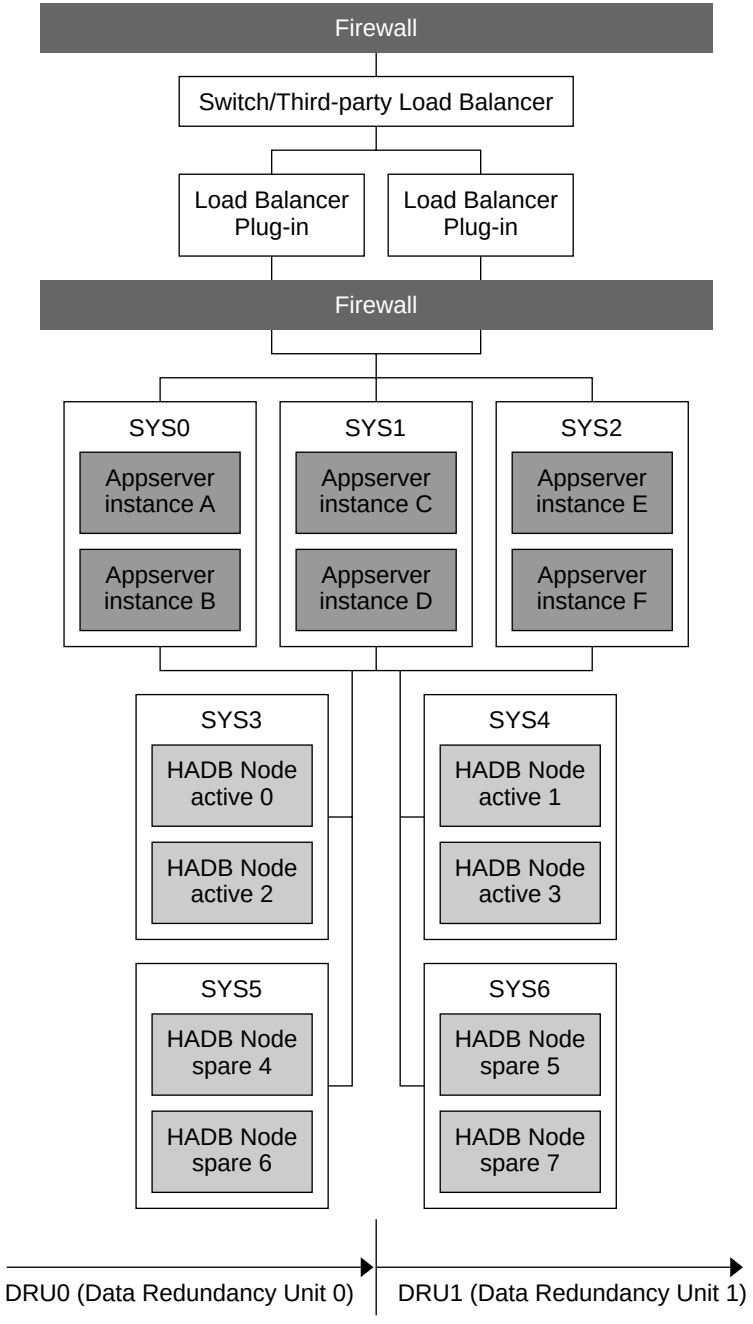
In the `clinstance.conf` file, include information for each instance. For more information, see *Sun Java System Application Server Installation Guide.*

Variation of Separate Tier Topology

You can increase the number of Application Server instances by adding more machines horizontally to the configuration. For example, you can add another machine to the reference configuration by creating a new Application Server instance. Similarly, you can increase the number of HADB nodes by adding more machines to host HADB nodes. Make sure that you add the HADB nodes in pairs with one node for each DRU.

This configuration is shown in Figure 3-4.

Figure 3-4 Variation of Separate Tier Topology



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

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

DRU0 comprises two machines:

- SYS3 that hosts the HADB Node active 0 and the HADB Node active 2
- SYS5, that hosts the HADB Node spare 4 and the HADB Node spare 6.

DRU1 comprises two machines:

- SYS4 that hosts the HADB Node active 1 and HADB Node active 3.
- SYS6 that hosts the HADB Node spare 5 and HADB Node spare 7.

Each machine hosting HADB nodes hosts two HADB nodes each. There are thus a total of eight HADB nodes (four active nodes and four spare nodes).

Configuration Settings for Variation to Reference Separate Tier Topology

Make the changes as described in the subsequent sections before you run the `clsetup` command.

Changes to clresource.conf File

Table 3-4 describes the changes needed to the `clresource.conf` file for the variation to the reference separate tier topology. The left column lists the section in the file in which the property to be changed is listed, the middle column lists the property name, and the right column lists the value of the property. For more information on the values of these properties, see “Changes to clresource.conf File” on page 53.

Table 3-4 Changes Needed to clresource.conf File for Variation to Reference Separate Tier Topology

Section of clresource.conf File in Which the Property Appears	Property Name	Value
HADBINFO	hosts	SYS3,SYS4,SYS3,SYS4, SYS5,SYS6,SYS5,SYS6
JDBC_CONNECTION_POOL	steadypoolsize	12
JDBC_CONNECTION_POOL	maxpoolsize	22

Changes to clinstance.conf File

In the `clinstance.conf` file, include information for each Application Server instance. For more information, see *Sun Java System Application Server Installation Guide.*

Comparison of Topologies

Table 3-5 presents a comparison of the co-located topology and the separate tier topology.

Table 3-5 Comparison of Topologies

Topology	Advantages	Disadvantages
Co-located Topology	• Requires fewer numbers of machines as compared to the separate tier topology. Because the HADB nodes and the application server instances are on the same tier, you can create an application server instance on each spare node to handle additional load. • Improved effectiveness of CPU utilization. An application server instance and an HADB node share one machine and the processing is distributed evenly among them. • Useful for large, Symmetric Multiprocessing (SMP) machines as you can take full advantage of the processing power of these machines.	Increased complexity of maintenance. For example, if you want to perform maintenance tasks (that require shutting down of machines) on HADB nodes, the application server instances on the machine hosting HADB nodes also become unavailable while the machine is unavailable.

Table 3-5 Comparison of Topologies

Topology	Advantages	Disadvantages
Separate Tier Topology	• Easier maintenance. For example, you can perform maintenance tasks for the machines that host application server instances without having to bring down HADB nodes. • Useful in situations where you have different types of machines. You can allocate a different set of machines to the application server instances tier and to the HADB tier. For example, you can use the more powerful machines for the application server instances tier and the less powerful machines for the HADB tier.	• Requires more machines as compared to 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 effectiveness of CPU utilization. The tier consisting of application server instances and the tier consisting of HADB nodes will likely have uneven loads. This is more significant when the number of machines is smaller (four to six).

Determining Which Topology to Use

You should test the different topologies mentioned in this chapter and experiment with different combinations of machines and CPUs to determine which topology (or its variation) best meets your performance and availability requirements.

Determine what trade offs you want to make to serve your needs the best. For example, if ease of maintenance is a critical requirement for you, the separate tier topology is more suitable. However, you will have to use a higher number of machines as compared to the co-located topology.

An important factor in the choice of topology is the type of machines you have in your setup. If you have large, Symmetric Multiprocessing (SMP) machines in your system, the co-located topology is an attractive option because you can take full advantage of the processing power of these machines. If you have different types of machines, separate tier topology may be more useful because you can allocate a different set of machines to the application server instances tier and to the HADB tier. For example, you can use more powerful machines for the application server instances tier and the less powerful machines for the HADB tier.

Appendix A

Checklist for Deployment

This appendix provides a checklist to get started on the evaluation and production with Sun Java System Application Server 7 2004Q2.

Table 0-1 Checklist

Component/Feature	Description
Application	Determine the following requirements for the application to be deployed. 1. The required/acceptable response time. 2. Peak load characteristics. 3. Necessary persistence scope and frequency. 4. Session timeout in web.xml. 5. The failover and availability requirements. For more information on planning for your requirements, see Chapter 2, “Planning your Environment” and “About Application Server Performance” in *Sun Java System Application Server 7 Performance and Tuning Guide*.
Hardware	Determine the following hardware requirements for deploying the application: 1. The same type of hardware should be used to host HADB nodes. 2. Have necessary amounts of hard disk space and memory installed. 3. Use the sizing exercise to identify the requirements for your deployment. For detailed information on the minimum system requirements, see *Sun Java System Application Server 7 Release Notes* at the product documentation web site at http://docs.sun.com/db/prod/s1appsrv#hic

Table 0-1 Checklist

Component/Feature	Description
Operating System	1. Ensure that the product is being installed on a supported platform. 2. Ensure that the patch levels are up-to-date and accurate. For more information on the supported platforms, see *Sun Java System Application Server 7 Release Notes* at the product documentation web site at http://docs.sun.com/db/prod/s1appsrv#hic
Network Infrastructure	1. Identify single points of failures and address them. 2. Make sure that the NIC cards and other network components are correctly configured. 3. Run `ttcp` benchmark test to determine if the throughput meets the requirements/expected result. 4. Setup `rsh/ssh` based on your preference so that HADB nodes can be installed. For more information on the required network infrastructure, setting up `rsh/ssh`, see *Sun Java System Application Server 7 Installation Guide*.
Back-ends and other external datasources	Check with your domain expert/vendor to ensure that these datasources are configured appropriately.
System Changes/Configuration	1. Make sure that changes to `/etc/system` and its equivalent on Linux are completed before running any performance/stress tests. 2. Make sure you have completed changes to TCP/IP settings. 3. By default, the system comes with lots of services pre-configured. Not all of them are required to be running. Turning off services that you do not need and thereby conserve system resources. 4. On Solaris, use `Setoolkit` to determine the behavior of the system. Resolve any flags that show up. For more information on optimum system configuration, see “About Application Server Performance” in *Sun Java System Application Server 7 Performance and Tuning Guide*.

Table 0-1 Checklist

Component/Feature	Description
Application Server and HADB Installation	1. Ensure that these servers are not installed on NFS mounted volumes. 2. Check for enough disk space and RAM when installing both Application Server and the HADB nodes on the same machine. 3. Check for enough independent disks when installing multiple HADB nodes on the same system. For more information on installing the Application Server and the HADB, see *Sun Java System Application Server 7 Installation Guide*.
HADB Configuration	1. Set the size of the HADB Data Device. 2. Define the DataBufferPoolSize. 3. Define the LogBufferSize. 4. Define the InternalBufferSize. 5. Set the NumberOfLocks. 6. Set optimum time-out values for various Application Server components. 7. Create the Physical layout of HADB nodes on the filesystem. For more information on installing the HADB, see chapter on "Preparing for HADB Setup" in *Sun Java System Application Server Installation Guide*. For more information on tuning the HADB, see chapter on "Tuning for High-Availability" in *Sun Java System Application Server Performance and Tuning Guide*.

Table 0-1 Checklist

Component/Feature	Description
Application Server Configuration	1. If sso is not required, then it can be turned off via `<sso-enabled = false />` property in `server.xml` configuration file. 2. Similarly, if MDB are not being used, then `jms-service` can be turned off. 3. Logging: Enable access log rotation. 4. Choose the right logging level, WARNING would be appropriate most of the times. 5. Configure J2EE containers using Admin Console. 6. Configure HTTP listeners using Admin Console. 7. Configure ORB threadpool using Admin Console. 8. If the application to be deployed has EJB's, then ensure that the `UtilDelegate` flag is used. For information on using this flag, see “Enabling the High Performance CORBA Util Delegate,” in *Sun Java System Application Server Performance and Tuning Guide*. 9. If using Type2 Drivers or calls involving native code, ensure that mtmalloc.so is specified in the LD_LIBRARY_PATH. 10. Consider disabling `server.policy` file if performance is critical. 11. Ensure that the appropriate persistence scope and frequency are used and they are not overridden underneath in the individual Web/EJB modules. 12. Ensure that only critical methods in the SFSB are checkpointed. For more information on tuning Application Server settings, see *Sun Java System Application Server 7 Performance and Tuning Guide*. For more information on configuring various Application Server components and services, see *Sun Java System Application Server 7 Administration Guide*.

Table 0-1 Checklist

Component/Feature	Description
Load balancer Configuration	1. Make sure you have installed the Web Server. 2. Make sure you have installed the load balancer plug-in. For more information on installing load balancer plug-in, see chapter, “Installing Standard and Enterprise Edition Software” in *Sun Java System Application Server 7 Installation Guide*. 3. Ensure to turn off unnecessary functions. 4. Make sure you have disabled patch checks. 5. Make sure you have configured `RqThrottle`, `KeepAliveQuery*` parameters. Lower the `KeepAliveQuery*` values; lower the value, latency will be lower on lightly loaded systems. Higher the values, higher will be the throughput on highly loaded systems. 6. Make sure you have enabled and configured perfdump in *instancename*`-obj.conf` file. For more information on load balancer configuration, see chapter “Configuring HTTP Load Balancing and Failover” in *Sun Java System Application Server 7 Administration Guide*.
JVM Configuration	1. At a minimum, specify the minimum and maximum heap sizes to be the same, and equal to one GB for each instance. 2. Refer to documentation on http://java.sun.com for configurable options in Java Hotspot JVM. 3. 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.

Table 0-1 Checklist

Component/Feature	Description
Configuring time-outs in Load balancer	1. Response-time-out-in-seconds: Determines how much time the load balancer will wait before declaring an Application Server instance as unhealthy. This value needs to be set based on the response time characteristics of your application. If this value is too high, then the Web Server/Load balancer plug-in is going to wait for a long time before marking that Application Server instance as unhealthy. If the value for Response-time-out-in-seconds is set too low and if the Application Server's response time crosses this threshold, the instance will be incorrectly marked as unhealthy. 2. Interval-in-seconds*:* Specifies the time interval in seconds after which unhealthy instances will be checked to find out if they have returned to a healthy state. Too low a value will generate extra traffic from the load balancer plug-in to Application Server instances and too high a value will delay the routing of requests to the instance that has turned healthy. 3. Timeout-in-seconds*:* Specifies the duration for a response to be obtained for a health check request. This value needs to be adjusted based on the traffic among the systems in the cluster to ensure that the health check succeeds. For more information on these load balancer configuration parameters, see chapter, "Configuring HTTP Load Balancing and Failover" in *Sun Java System Application Server Administration Guide*.
Configuring time-outs in Application Server	1. Max-wait-time-millis*:* Defines the wait time to get a connection from the pool before throwing an exception. Default is 60000 milli seconds. Consider changing this value in case of highly loaded systems where the size of the data being persisted is greater than 50 Kb. 2. Cache-idle-timeout-in-seconds*:* Applies to entity beans and stateful session beans. Defines the time the bean is allowed to be idle in the cache before it gets passivated. 3. Removal-timeout-in-seconds: The amount of time that the bean remains passivated, that is, idle in the backup store, is controlled by removal-timeout-in-seconds parameter. The default value is 60 minutes. This value needs to be adjusted based on the need for SFSB failover. 4. All of these values should be set by paying attention to the HADB's JDBC connection pool setting max-wait-time-in-millis. For more information on tuning the timeout parameters, see chapter, "Tuning the Application Server" in *Sun Java System Application Server Performance and Tuning Guide*.

Table 0-1 Checklist

Component/Feature	Description
HADB time-outs	1. sql_client_timeout: This variable controls the 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 may cause SQL sessions to be aborted prematurely, while setting it too high may cause SQL sessions that are not idle, but has exited, to occupy resources. This in turn may prevent other SQL clients from logging on. When tuning this variable also consider the settings of “nsessions.” The default value is 1800 seconds and it can be changed by editing the configuration file. If the HADB JDBC connection pool steady-pool-size is greater than(>) max-pool-size, then idle-timeout-in-seconds should be set lower than the sql_client_timeout, so that the Application Server itself will close the connection before HADB closes the connection. 2. lock_timeout*:* Specifies the maximum time a transaction waits for access to data. When this time is 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 (i.e. deadlocks), and causing high server load. The default value is 5000 ms. Do not set this value to below 500 ms. If you see the “transaction timed out” messages in the server log, then it is a good idea to increase this value. The lock timeout value can be set by adding a property to the ha jdbc connection pool as: `<property name=lockTimeout value=″x″ />` where x is in milliseconds. 3. Querytimeout*:* This is the maximum time in milliseconds that the HADB waits for a query to execute. The default value is 30 seconds. If you see exceptions in the server log consistently indicating the query time out, you should consider increasing this value. This value can be set by adding a property to the ha jdbc connection pool as: `<property name=QueryTimeout value=″x″ />` where x is in milliseconds. 4. loginTimeout: This is the maximum time that the client waits to login to the HADB. This can be set by adding a property to the ha jdbc connection pool as: `<property name=loginTimeout value=″x″ />` where x is in seconds. Default value is 10sec. 5. MaxTransIdle: The maximum time a transaction can be idle (msec) between sending a reply to the client and receiving the next request. The default value is 40sec. This can be changed by adding a property to the ha jdbc connection pool as: `<property name=maxtransIdle value=″x″ />` where x is in milliseconds. For more information on these timeout parameters, see chapter, “Tuning for High-Availability” in *Sun Java System Application Server Performance and Tuning Guide*.

Table 0-1 Checklist

Component/Feature	Description

Implications of GC in Application Server, Enterprise Edition	GC pauses if they are long enough, that is greater than or equal to (>=) 4sec, which can cause intermittent problems in persisting session state into HADB. To avoid this problem, It is recommended to tune the vm heap. In cases where even a single failure to persist data is not acceptable and also in cases where the system is not fully loaded, using CMS collector can help. Other option is to use the throughput collector.
	
	These can be enabled by adding:
	
	`<jvm-options>-XX:+UseConcMarkSweepGC</jvm-options>`
	
	`<jvm-options> -XX:SoftRefLRUPolicyMSPerMB=1</jvm-options>`
	
	Note that with the use of this option, you might experience a drop in throughput.
	
	For any late-breaking updates to the GC parameters, see *Sun Java System Application Server 7 Release Notes* at the product documentation web site at `http://docs.sun.com/db/prod/s1appsrv#hic`

Common documents that can help	1. JVM options: `http://java.sun.com/docs/hotspot/gc1.4.2/index.html` The following documents are available at `http://docs.sun.com/db/prod/s1appsrv#hic` 1. *Sun Java System Application Server Installation Guide* 2. *Sun Java System Application Server System Deployment Guide* 3. *Sun Java System Application Server System Performance and Tuning Guide* 4. *Sun Java System Application Server Error Messages Guide* 5. *Sun Java System Application Server Release Notes*

Index

P

R

S

T

U