Sun GlassFish Communications Server 2.0 Deployment Planning Guide

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

Part No: 821–0190
October 2009

091103@22749

Contents

Figures

Tables

Examples

Preface

Deployment Planning Guide explains how to build a production deployment.

This preface contains information about and conventions for the entire Sun GlassFish™ Communications Server documentation set.

Communications Server Documentation Set

The Uniform Resource Locator (URL) for Communications Server documentation is `http://docs.sun.com/coll/1343.10`. For an introduction to Communications Server, refer to the books in the order in which they are listed in the following table.

TABLE P–1 Books in the Communications Server Documentation Set

Book Title	Description
Documentation Center	Communications Server documentation topics organized by task and subject.
Release Notes	Late-breaking information about the software and the documentation. Includes a comprehensive, table-based summary of the supported hardware, operating system, Java™ Development Kit (JDK™), and database drivers.
Quick Start Guide	How to get started with the Communications Server product.
Installation Guide	Installing the software and its components.
Application Deployment Guide	Deployment of applications and application components to the Communications Server. Includes information about deployment descriptors.
Developer's Guide	Creating and implementing Java Platform, Enterprise Edition (Java EE platform) applications intended to run on the Communications Server that follow the open Java standards model for Java EE components and APIs. Includes information about developer tools, security, debugging, and creating lifecycle modules.
Java EE 5 Tutorial	Using Java EE 5 platform technologies and APIs to develop Java EE applications.
Java WSIT Tutorial	Developing web applications using the Web Service Interoperability Technologies (WSIT). Describes how, when, and why to use the WSIT technologies and the features and options that each technology supports.

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

Book Title	Description
Administration Guide	System administration for the Communications Server, including configuration, monitoring, security, resource management, and web services management.
High Availability Administration Guide	Setting up clusters, working with node agents, and using load balancers.
Administration Reference	Editing the Communications Server configuration file, `domain.xml`.
Performance Tuning Guide	Tuning the Communications Server to improve performance.
Reference Manual	Utility commands available with the Communications Server; written in man page style. Includes the `asadmin` command line interface.

Related Documentation

For documentation about other stand-alone Sun GlassFish server products, go to the following:

- Message Queue documentation (`http://docs.sun.com/coll/1343.4`)
- Identity Server documentation (`http://docs.sun.com/app/docs/prod/ident.mgmt#hic`)
- Directory Server documentation (`http://docs.sun.com/coll/1224.1`)
- Web Server documentation (`http://docs.sun.com/coll/1308.3`)

A Javadoc™ tool reference for packages provided with the Communications Server is located at `http://glassfish.dev.java.net/nonav/javaee5/api/index.html`. Additionally, the following resources might be useful:

- The Java EE 5 Specifications (`http://java.sun.com/javaee/5/javatech.html`)
- The Java EE Blueprints (`http://java.sun.com/reference/blueprints/index.html`)

For information on creating enterprise applications in the NetBeans™ Integrated Development Environment (IDE), see `http://www.netbeans.org/kb/55/index.html`.

For information about the Java DB database included with the Communications Server, see `http://developers.sun.com/javadb/`.

The GlassFish Samples project is a collection of sample applications that demonstrate a broad range of Java EE technologies. The GlassFish Samples are bundled with the Java EE Software Development Kit (SDK), and are also available from the GlassFish Samples project page at `https://glassfish-samples.dev.java.net/`.

Default Paths and File Names

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

TABLE P–2 Default Paths and File Names

Placeholder	Description	Default Value
as-install	Represents the base installation directory for Communications Server.	Solaris™ and Linux installations, non-root user: *user's-home-directory*`/SUNWappserver` Solaris and Linux installations, root user: `/opt/SUNWappserver` Windows, all installations: *SystemDrive*`:\Sun\AppServer`
domain-root-dir	Represents the directory containing all domains.	All installations: *as-install*`/domains/`
domain-dir	Represents the directory for a domain. In configuration files, you might see *domain-dir* represented as follows: `${com.sun.aas.instanceRoot}`	*domain-root-dir*`/`*domain-dir*
instance-dir	Represents the directory for a server instance.	*domain-dir*`/`*instance-dir*
samples-dir	Represents the directory containing sample applications.	*as-install*`/samples`
docs-dir	Represents the directory containing documentation.	*as-install*`/docs`

Typographic Conventions

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

TABLE P–3 Typographic Conventions

Typeface	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories, and onscreen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% you have mail.`

TABLE P–3 Typographic Conventions *(Continued)*

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

Symbol Conventions

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

TABLE P–4 Symbol Conventions

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

Documentation, Support, and Training

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

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

Third-Party Web Site References

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

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

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions.

To share your comments, go to http://docs.sun.com and click Feedback. In the online form, provide the document title and part number. The part number is a seven-digit or nine-digit number that can be found on the title page of the book or at the top of the document.

◆ ◆ ◆ CHAPTER 1

Product Concepts

The Sun GlassFish Communications Server provides a robust platform for the development, deployment, and management of Java EE , converged, and SIP applications. Key features include scalable transaction management, web services performance, clustering, security, and integration capabilities.

This chapter covers the following topics:

Java EE Platform Overview

The Communications Server implements Java 2 Enterprise Edition (Java EE) 1.4 technology. The Java EE platform is a set of standard specifications that describe application components, APIs, and the runtime containers and services of an application server.

Java EE Applications

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

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

Sorry: the graphics are not currently available.

Containers

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

Java EE Services

The Java EE platform provides services for applications, including:

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

Web Services

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

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

Client Access

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

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

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

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

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

External Systems and Resources

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

- *Java Database Connectivity (JDBC)* - A database management system (DBMS) provides facilities for storing, organizing, and retrieving data. Most business applications store data in relational databases, which applications access via JDBC. The Application Server includes the PointBase DBMS for use sample applications and application development and prototyping, though it is not suitable for deployment. The Application Server provides certified JDBC drivers for connecting to major relational databases. These drivers are suitable for deployment.
- *Java Message* Service - Messaging is a method of communication between software components or applications. A messaging client sends messages to, and receives messages from, any other client via a messaging provider that implements the Java Messaging Service (JMS) API. The Application Server includes a high-performance JMS broker, the Sun Java System Message Queue. The Platform Edition of Application Server includes the free Platform Edition of Message Queue. Sun GlassFishCommunications Server includes Message Queue Enterprise Edition, which supports clustering and failover.
- **Java EE Connectors** - The Java EE Connector architecture enables integrating Java EE applications and existing Enterprise Information Systems (EIS). An application accesses an EIS through a portable Java EE component called a *connector* or *resource adapter*, analogous to using JDBC driver to access an RDBMS. Resource adapters are distributed as standalone

Resource Adapter Archive (RAR) modules or included in Java EE application archives. As RARs, they are deployed like other Java EE components. The Application Server includes evaluation resource adapters that integrate with popular EIS.

- **JavaMail** - Through the JavaMail API, applications can connect to a Simple Mail Transport Protocol (SMTP) server to send and receive email.

Application Server Components

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

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

Server Instances

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

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

Administrative Domains

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

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

Domain Administration Server (DAS)

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

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

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

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

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

Administrative commands are provided to backup and restore domain configuration and applications. With the standard backup and restore procedures, you can quickly restore working configurations. If the DAS host fails, you must create a new DAS installation to restore the previous domain configuration. For instructions, see "Recreating the Domain Administration Server" in *Sun GlassFish Communications Server 2.0 Administration Guide*.

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

Clusters

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

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

Clusters, domains, and instances are related as follows:

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

Node Agents

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

- Starts and stops server instances as instructed by the DAS.
- Restarts failed server instances.
- Provides a view of the log files of failed servers and assists in remote diagnosis
- Synchronizes each server instance's local configuration repository with the DAS's central repository, as it starts up the server instances under its watch.
- When an instance is initially created, creates directories the instance needs and synchronizes the instance's configuration with the central repository.
- Performs appropriate cleanup when a server instance is deleted.

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

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

For more information on node agents, see Chapter 4, "Configuring Node Agents," in *Sun GlassFish Communications Server 2.0 High Availability Administration Guide*.

Named Configurations

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

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

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

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

Converged Load Balancer

The load balancer distributes the workload among multiple physical machines, thereby increasing the overall throughput of the system.

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

IIOP Load Balancing in a Cluster

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

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

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

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

Message Queue and JMS Resources

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

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

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

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

For information on deploying JMS applications and MQ clustering for message failover, see "Planning Message Queue Broker Deployment" on page 32.

◆ ◆ ◆ CHAPTER 2

Planning your Deployment

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

This chapter contains the following sections:

Establishing Performance Goals

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

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

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

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

This section discusses the following topics:

Estimating Throughput

In broad terms, *throughput* measures the amount of work performed by Communications Server. For Communications Server, throughput can be defined as the number of requests processed per minute per server instance.

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

Estimating Load on Application Server Instances

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

Maximum Number of Concurrent Users

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

Initially, as the number of users increases, throughput increases correspondingly. However, as the number of concurrent requests increases, server performance begins to saturate, and throughput begins to decline.

Identify the point at which adding concurrent users reduces the number of requests that can be processed per minute. This point indicates when optimal performance is reached and beyond

which throughput start to degrade. Generally, strive to operate the system at optimal throughput as much as possible. You might need to add processing power to handle additional load and increase throughput.

Think Time

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

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

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

Average Response Time

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

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

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

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

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

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

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

where

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

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

EXAMPLE 2–1 Calculation of Response Time

If the following conditions exist:

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

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

Thus, the calculation of response time is:

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

Therefore, the response time is two seconds.

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

Requests Per Minute

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

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

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

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

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

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

EXAMPLE 2–2 Calculation of Requests Per Second

For the values:

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

The calculation for the number of requests per second is:

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

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

Planning the Network Configuration

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

The following topics are covered in this section:

Estimating Bandwidth Requirements

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

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

For example, assume that at peak load, five percent of users occasionally do not have immediate network access when accessing applications deployed on Communications Server. Of that five

percent, estimate how many users retry access after the first attempt. Again, not all of those users might get through, and of that unsuccessful portion, another percentage will retry. As a result, the peak appears longer because peak use is spread out over time as users continue to attempt access.

Calculating Bandwidth Required

Based on the calculations made in "Establishing Performance Goals" on page 25, determine the additional bandwidth required for deploying the Application Server at your site.

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

EXAMPLE 2–3 Calculation of Bandwidth Required

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

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

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

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

Estimating Peak Load

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

EXAMPLE 2–4 Calculation of Peak Load

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

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

EXAMPLE 2–4 Calculation of Peak Load *(Continued)*

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

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

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

Planning for Availability

This section contains the following topics:

Rightsizing Availability

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

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

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

Using Clusters to Improve Availability

At the most basic level, a cluster is a group of application server instances—often hosted on multiple physical servers—that appear to clients as a single instance. This provides horizontal scalability as well as higher availability than a single instance on a single machine. This basic

level of clustering works in conjunction with the converged load balancer that accepts HTTP/HTTPS and SIP/SIPS requests and forwards them to one of the instances in the cluster. The ORB and integrated JMS brokers also perform load balancing to application server clusters. If an instance fails, become unavailable (due to network faults), or becomes unresponsive, requests are redirected only to existing, available machines. The load balancer can also recognize when an failed instance has recovered and redistribute load accordingly.

Adding Redundancy to the System

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

Identifying Failure Classes

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

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

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

Planning Message Queue Broker Deployment

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

Sun Java System Message Queue (MQ) is integrated with Communications Server using a *connector module,* also known as a resource adapter, as defined by the Java EE Connector Architecture Specification (JCA) 1.5. A connector module is a standardized way to add

functionality to the Communications Server. Java EE components deployed to the Communications Server exchange JMS messages using the JMS provider integrated via the connector module. By default, the JMS provider is the Sun Java System Message Queue, but if you wish you can use a different JMS provider, as long as it implements JCA 1.5.

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

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

Multi-Broker Clusters

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

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

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

- Provide messaging services despite hardware failures on a single machine.
- Minimize downtime while performing system maintenance.
- Accommodate workgroups having different user repositories.
- Deal with firewall restrictions.

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

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

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

Master Broker and Client Synchronization

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

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

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

Configuring Application Server to Use Message Queue Brokers

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

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

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

Managing JMS with Admin Console

In the Admin Console, you can set JMS properties using the Java Message Service node for a particular configuration. You can set properties such as Reconnect Interval and Reconnect Attempts. For more information, see Chapter 4, "Configuring Java Message Service Resources," in *Sun GlassFish Communications Server 2.0 Administration Guide*.

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

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

Managing JMS with asadmin

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

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

    For more information on these commands, see *Sun GlassFish Communications Server 2.0 Reference Manual*or the corresponding man pages.

Java Message Service Type

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

Local Java Message Service

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

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

Remote Java Message Service

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

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

Default JMS Host

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

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

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

Example Deployment Scenarios

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

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

Default Deployment

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

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

"Default Deployment" on page 36 illustrates an example default deployment with an Communications Server cluster containing three instances.

Using an MQ Broker Cluster with an Application Server Cluster

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

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

Specifying an Application-Specific MQ Broker Cluster

In some cases, an application may need to use a different MQ broker cluster than the one used by the Communications Server cluster. "Specifying an Application-Specific MQ Broker Cluster" on page 37 illustrates an example of such a scenario. To do so, use the `AddressList` property of a JMS connection factory or the `activation-config` element in an MDB deployment descriptor to specify the MQ broker cluster.

For more information about configuring connection factories, see "JMS Connection Factories" in *Sun GlassFish Communications Server 2.0 Administration Guide*. For more information about MDBs, see "Using Message-Driven Beans" in *Sun GlassFish Communications Server 2.0 Developer's Guide*.

Application Clients

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

◆ ◆ ◆ CHAPTER 3

Selecting a Topology

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

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

This chapter discusses:

- "Common Requirements" on page 39 for both topologies.
- The two topologies:
  - "Co-located Topology" on page 41 - Communications Server instances and HADB nodes are on the same machine.
  - "Separate Tier Topology" on page 45 - Communications Server instances and HADB nodes are on different machines.
- "Determining Which Topology to Use" on page 49

Common Requirements

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

- "General Requirements" on page 40
- "HADB Nodes and Machines" on page 40
- "Load Balancer Configuration" on page 41

General Requirements

Both topologies must meet the following general requirements:

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

HADB Nodes and Machines

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

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

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

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

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

Load Balancer Configuration

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

Co-located Topology

In the co-located topology, the Communications 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 Communications Server instance and an HADB node share one machine and the processing is distributed evenly among them.

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

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

Example Configuration

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

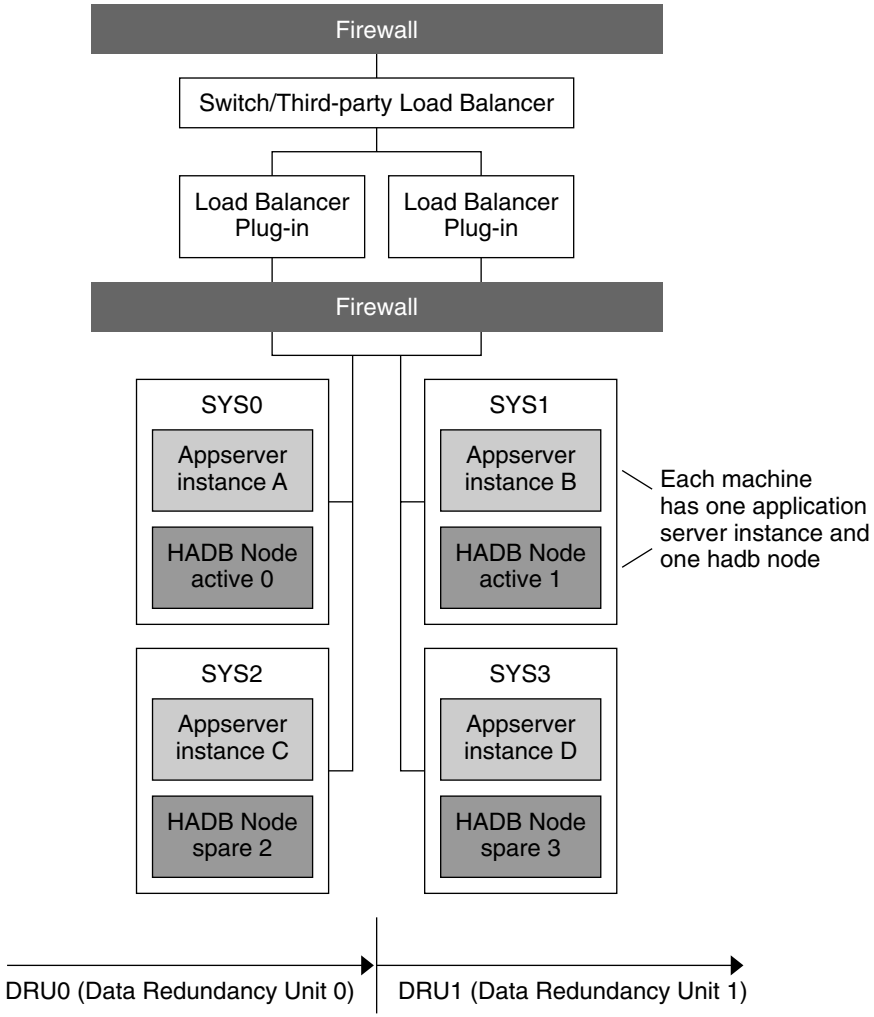

FIGURE 3–1 Example Co-located Topology

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

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

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

Variation of Co-located Topology

For better scalability and throughput, increase the number of Communications Server instances and HADB nodes by adding more machines. For example, you could add two machines, each with one Communications Server instance and one HADB node. Make sure to add the HADB nodes in pairs, assigning one node for each DRU. "Variation of Co-located Topology" on page 43 illustrates this configuration.

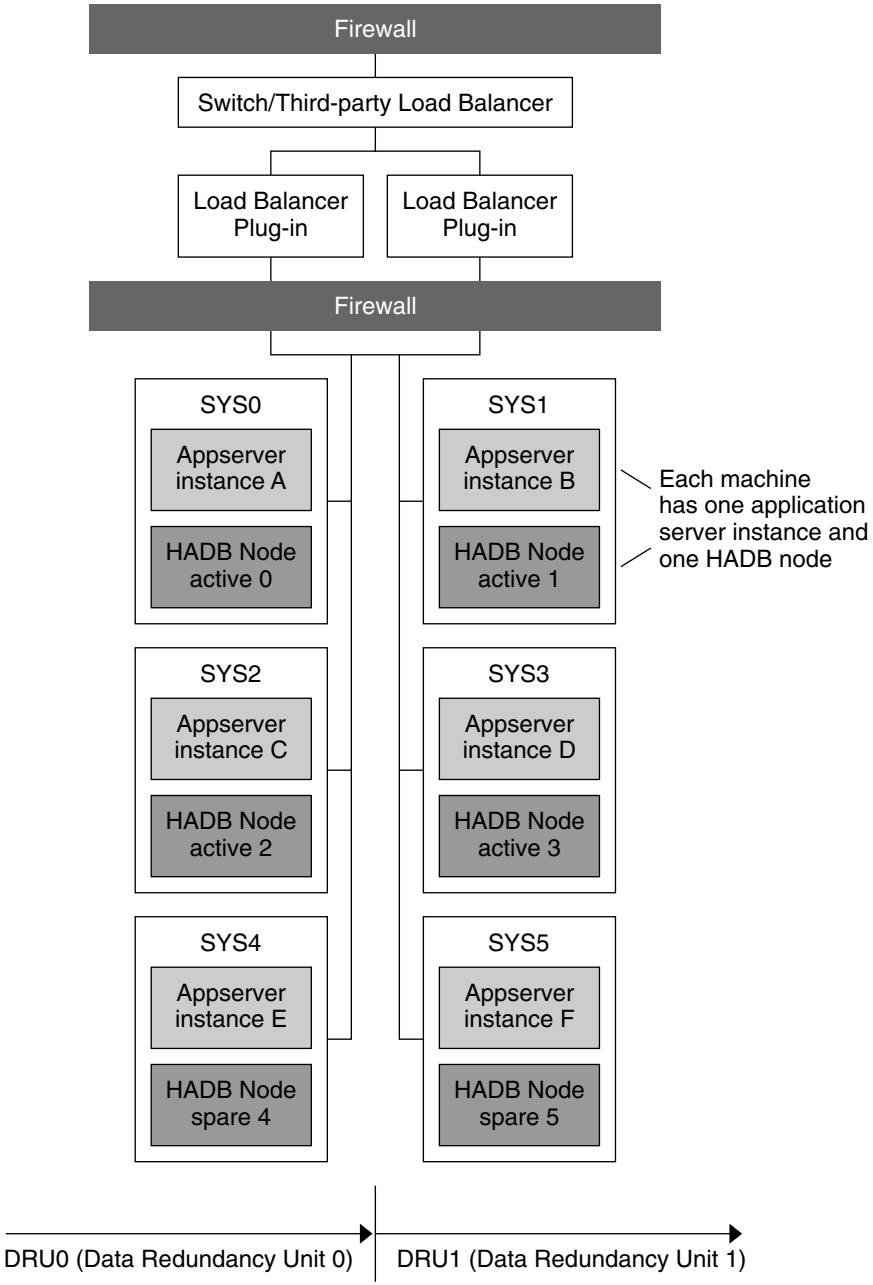

FIGURE 3–2 Variation of Co-located Topology

In this variation, the machines SYS4 and SYS5 have been added to the co-located topology described in "Example Configuration" on page 41.

Communications Server instances are hosted as follows:

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

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

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

Separate Tier Topology

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

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

Example Configuration

The following figure illustrates the separate tier topology.

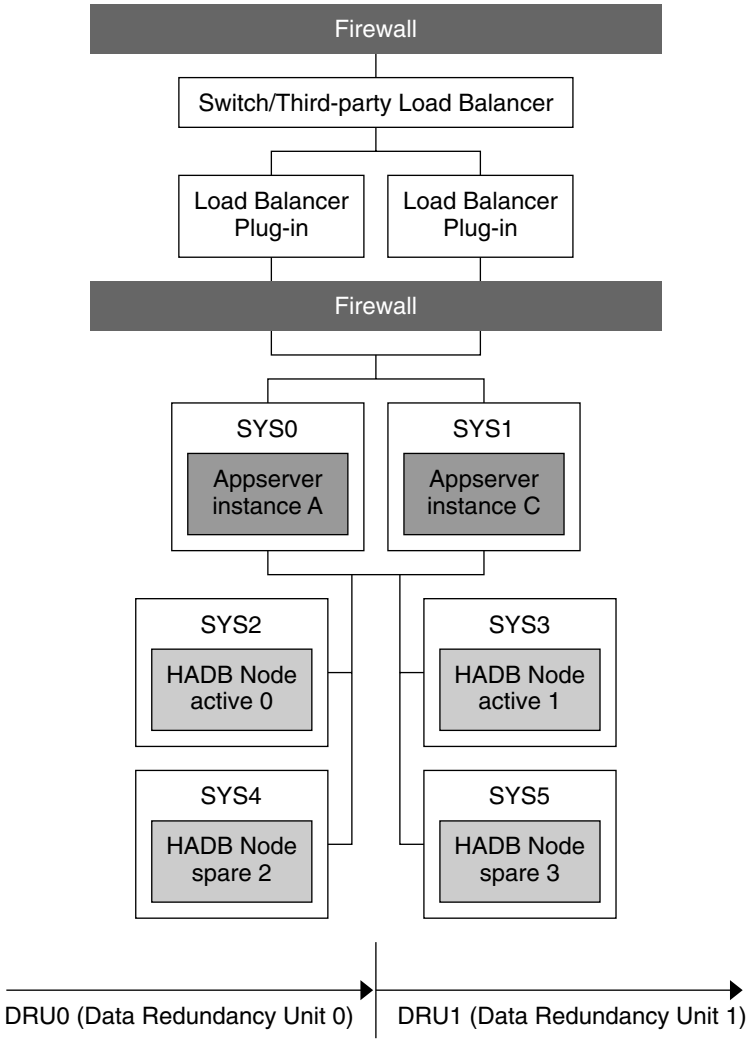

FIGURE 3–3 Example Separate Tier Topology

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

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

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

Variation of Separate Tier Topology

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

"Variation of Separate Tier Topology" on page 47 illustrates this configuration.

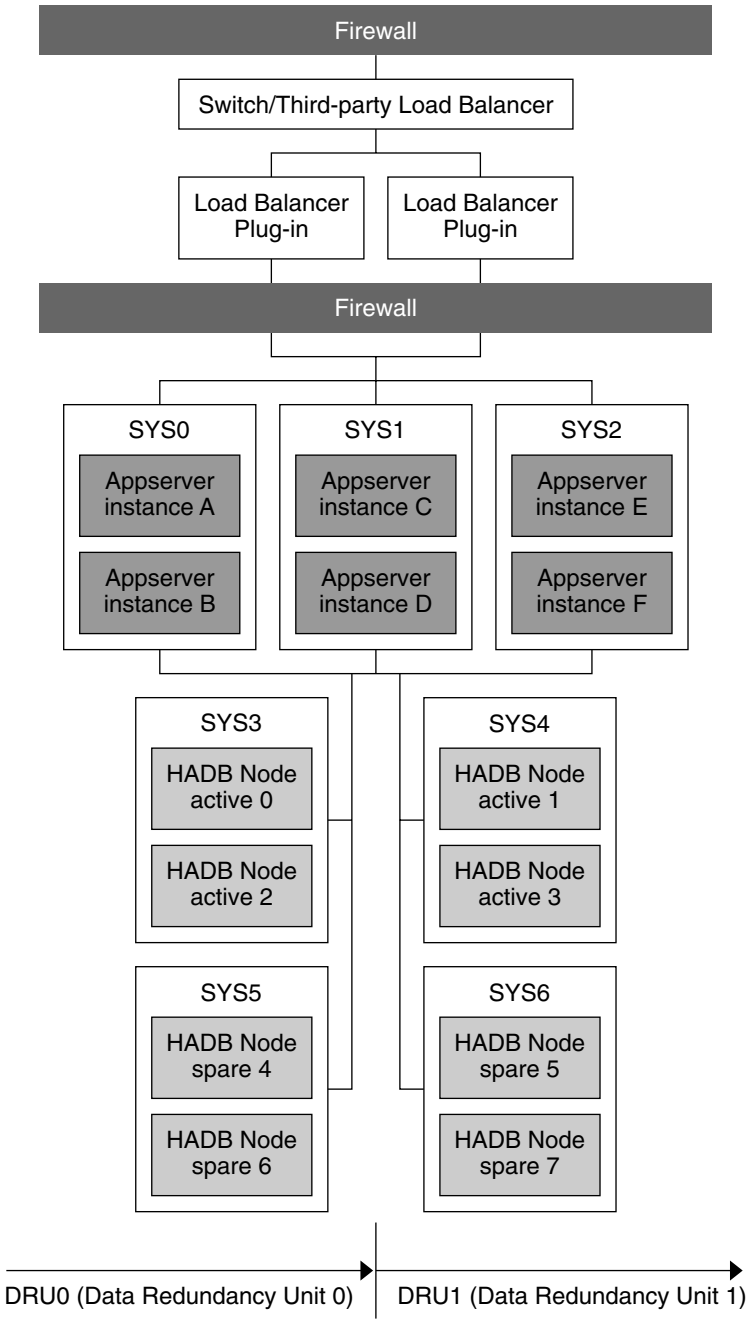

FIGURE 3–4 Variation of Separate Tier Topology

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

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

DRU0 comprises two machines:

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

DRU1 comprises two machines:

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

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

Determining Which Topology to Use

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

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

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

Comparison of Topologies

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

TABLE 3–1 Comparison of Topologies

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

◆ ◆ ◆ CHAPTER 4

Checklist for Deployment

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

Checklist for Deployment

TABLE 4–1 Checklist

Component/Feature	Description
Application	Determine the following requirements for the application to be deployed. ■ Required/acceptable response time. ■ Peak load characteristics. ■ Necessary persistence scope and frequency. ■ Session timeout in `web.xml`. ■ Failover and availability requirements. For more information see *Sun GlassFish Communications Server 2.0 Performance Tuning Guide*.
Hardware	■ Have necessary amounts of hard disk space and memory installed. ■ Use the sizing exercise to identify the requirements for deployment. For more information see *Sun GlassFish Communications Server 2.0 Release Notes*
Operating System	■ Ensure that the product is installed on a supported platform. ■ Ensure that the patch levels are up-to-date and accurate. For more information see *Sun GlassFish Communications Server 2.0 Release Notes*

TABLE 4–1 Checklist *(Continued)*

Component/Feature	Description
Network Infrastructure	■ Identify single points of failures and address them. ■ Make sure that the NICs and other network components are correctly configured. ■ Run `ttcp` benchmark test to determine if the throughput meets the requirements/expected result. ■ Setup `rsh/ssh` based your preference so that HADB nodes are properly installed. For more information see *Sun GlassFish Communications Server 2.0 Installation Guide*.
Back-ends and other external data sources	Check with the domain expert or vendor to ensure that these data sources are configured appropriately.
System Changes/Configuration	■ Make sure that changes to `/etc/system` and its equivalent on Linux are completed before running any performance/stress tests. ■ Make sure the changes to the TCP/IP settings are complete. ■ By default, the system comes with lots of services pre-configured. Not all of them are required to be running. Turn off services that are not needed to conserve system resources. ■ On Solaris, use `Setoolkit` to determine the behavior of the system. Resolve any flags that show up. For more information see *Sun GlassFish Communications Server 2.0 Performance Tuning Guide*.
Installation	■ Ensure that these servers are not installed on NFS mounted volumes.
Communications Server Configuration	■ Logging: Enable access log rotation. ■ Choose the right logging level. WARNING is usually appropriate. ■ Configure Java EE containers using Admin Console. ■ Configure HTTP listeners using Admin Console. ■ Configure SIP listeners using Admin Console. ■ Configure ORB threadpool using Admin Console. ■ If using Type2 drivers or calls involving native code, ensure that mtmalloc.so is specified in the LD_LIBRARY_PATH. ■ Ensure that the appropriate persistence scope and frequency are used and they are not overridden underneath in the individual Web/EJB modules. ■ Ensure that only critical methods in the SFSB are checkpointed. For more information on tuning, see *Sun GlassFish Communications Server 2.0 Performance Tuning Guide*. For more information on configuration, see *Sun GlassFish Communications Server 2.0 Administration Guide*.

TABLE 4–1 Checklist *(Continued)*

Component/Feature	Description
Converged Load balancer Configuration	■ Make sure that you have set the auto-apply option appropriately. ■ Verify that the configuration file for load balancer has appropriate values.
Java Virtual Machine Configuration	■ Initially set the minimum and maximum heap sizes to be the same, and at least one GB for each instance. ■ See Java Hotspot VM Options for more information. ■ When running multiple instances of Communications Server, consider creating a processor set and bind Communications Server to it. This helps in cases where the CMS collector is used to sweep the old generation.
Configuring time-outs in Communications Server	■ Max-wait-time-millis - Wait time to get a connection from the pool before throwing an exception. Default is 6 s. Consider changing this value for highly loaded systems where the size of the data being persisted is greater than 50 KB. ■ Cache-idle-timeout-in-seconds - Time an EJB is allowed to be idle in the cache before it gets passivated. Applies only to entity beans and stateful session beans. ■ Removal-timeout-in-seconds - Time that an EJB remains passivated (idle in the backup store). Default value is 60 minutes. Adjust this value based on the need for SFSB failover.
Tune VM Garbage Collection (GC)	Garbage collection pauses of four seconds or more can cause intermittent problems. To avoid this problem, tune the VM heap. In cases where even a single failure to persist data is unacceptable or when the system is not fully loaded, use the CMS collector or the throughput collector. These can be enabled by adding: `<jvm-options>-XX:+UseConcMarkSweepGC</jvm-options>` This option may decrease throughput.

Index

T

U

W