# Sun Java System Directory Server Enterprise Edition 6.3 Deployment Planning Guide

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

Part No: 820–2760
April 2008

080404@19860

# Contents

# Figures

# Preface

The *Sun Java System Directory Server Enterprise Edition 6.3 Deployment Planning Guide* contains the information that you need to plan a directory service deployment. This guide describes the initial decisions that you need to make on issues such as data types, access control, and sizing. It also provides high-level examples and strategies that you can use for the specific requirements of your enterprise.

## Who Should Use This Book

This guide is primarily intended for deployment architects and business planners responsible for the analysis and design of directory service deployments. This guide is also useful for system integrators and other people responsible for the design and implementation of enterprise applications.

## Before You Read This Book

This guide assumes that you are familiar with the basic concepts of LDAP directory servers and that you have read these documents:

- *Sun Java System Directory Server Enterprise Edition 6.3 Release Notes*
- *Sun Java System Directory Server Enterprise Edition 6.3 Evaluation Guide*

## How This Book Is Organized

This guide is based on a solution life cycle that describes the various phases of deployment planning.

Part I provides an introduction to Directory Server Enterprise Edition and explains the steps involved in planning a deployment (solution life cycle).

Part II describes the technical requirements analysis that must be performed before you can begin drawing up a logical deployment architecture. Technical requirements analysis requires an understanding of the business domain, business objectives, and the underlying system technology.

Part III describes how to create logical architectures for Directory Server Enterprise Edition deployments. It also provides sample logical architectures based on typical Directory Server Enterprise Edition deployment scenarios.

Part IV discusses specialized deployment topics including the use of LDAP-based naming services on the Solaris Operating System, Identity Synchronization for Windows, and the deployment of a virtual directory.

# Directory Server Enterprise Edition Documentation Set

This Directory Server Enterprise Edition documentation set explains how to use Sun Java System Directory Server Enterprise Edition to evaluate, design, deploy, and administer directory services. In addition, it shows how to develop client applications for Directory Server Enterprise Edition. The Directory Server Enterprise Edition documentation set is available at `http://docs.sun.com/coll/1224.4`.

For an introduction to Directory Server Enterprise Edition, review the following documents in the order in which they are listed.

**TABLE P–1** Directory Server Enterprise Edition Documentation

| Document Title | Contents |
|---|---|
| *Sun Java System Directory Server Enterprise Edition 6.3 Release Notes* | Contains the latest information about Directory Server Enterprise Edition, including known problems. |
| *Sun Java System Directory Server Enterprise Edition 6.3 Documentation Center* | Contains links to key areas of the documentation set. |
| *Sun Java System Directory Server Enterprise Edition 6.3 Evaluation Guide* | Introduces the key features of this release. Demonstrates how these features work and what they offer in the context of a fictional deployment that you can implement on a single system. |
| *Sun Java System Directory Server Enterprise Edition 6.3 Deployment Planning Guide* | Explains how to plan and design highly available, highly scalable directory services based on Directory Server Enterprise Edition. Presents the basic concepts and principles of deployment planning and design. Discusses the solution life cycle, and provides high-level examples and strategies to use when planning solutions based on Directory Server Enterprise Edition. |

**TABLE P–1** Directory Server Enterprise Edition Documentation *(Continued)*

| Document Title | Contents |
|---|---|
| *Sun Java System Directory Server Enterprise Edition 6.3 Installation Guide* | Explains how to install the Directory Server Enterprise Edition software. Shows how to select which components to install, configure those components after installation, and verify that the configured components function properly.<br><br>For instructions on installing Directory Editor, go to `http://docs.sun.com/coll/DirEdit_05q1`.<br><br>Make sure you read the information in *Sun Java System Directory Server Enterprise Edition 6.3 Release Notes* concerning Directory Editor before you install Directory Editor. |
| *Sun Java System Directory Server Enterprise Edition 6.3 Migration Guide* | Provides migration instructions from the earlier versions of Directory Server, Directory Proxy Server, and Identity Synchronization for Windows. |
| *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide* | Provides command-line instructions for administering Directory Server Enterprise Edition.<br><br>For hints and instructions on using the Directory Service Control Center, DSCC, to administer Directory Server Enterprise Edition, see the online help provided in DSCC.<br><br>For instructions on administering Directory Editor, go to `http://docs.sun.com/coll/DirEdit_05q1`.<br><br>For instructions on installing and configuring Identity Synchronization for Windows, see Part II, "Installing Identity Synchronization for Windows," in *Sun Java System Directory Server Enterprise Edition 6.3 Installation Guide*. |
| *Sun Java System Directory Server Enterprise Edition 6.3 Developer's Guide* | Shows how to develop directory client applications with the tools and APIs that are provided as part of Directory Server Enterprise Edition. |
| *Sun Java System Directory Server Enterprise Edition 6.3 Reference* | Introduces the technical and conceptual foundations of Directory Server Enterprise Edition. Describes its components, architecture, processes, and features. Also provides a reference to the developer APIs. |
| *Sun Java System Directory Server Enterprise Edition 6.3 Man Page Reference* | Describes the command-line tools, schema objects, and other public interfaces that are available through Directory Server Enterprise Edition. Individual sections of this document can be installed as online manual pages. |
| *Sun Java System Directory Server Enterprise Edition 6.3 Troubleshooting Guide* | Provides information for defining the scope of the problem, gathering data, and troubleshooting the problem areas using various tools. |
| *Sun Java System Identity Synchronization for Windows 6.0 Deployment Planning Guide* | Provides general guidelines and best practices for planning and deploying Identity Synchronization for Windows |

# Related Reading

The SLAMD Distributed Load Generation Engine is a Java™ application that is designed to stress test and analyze the performance of network-based applications. It was originally developed by Sun Microsystems, Inc. to benchmark and analyze the performance of LDAP directory servers. SLAMD is available as an open source application under the Sun Public License, an OSI-approved open source license. To obtain information about SLAMD, go to `http://www.slamd.com/`. SLAMD is also available as a java.net project. See `https://slamd.dev.java.net/`.

Java Naming and Directory Interface (JNDI) technology supports accessing the Directory Server using LDAP and DSML v2 from Java applications. For information about JNDI, see `http://java.sun.com/products/jndi/`. The *JNDI Tutorial* contains detailed descriptions and examples of how to use JNDI. This tutorial is at `http://java.sun.com/products/jndi/tutorial/`.

Directory Server Enterprise Edition can be licensed as a standalone product, as a component of Sun Java Enterprise System, as part of a suite of Sun products, such as the Sun Java Identity Management Suite, or as an add-on package to other software products from Sun. Java Enterprise System is a software infrastructure that supports enterprise applications distributed across a network or Internet environment. If Directory Server Enterprise Edition was licensed as a component of Java Enterprise System, you should be familiar with the system documentation at `http://docs.sun.com/coll/1286.3`.

Identity Synchronization for Windows uses Message Queue with a restricted license. Message Queue documentation is available at `http://docs.sun.com/coll/1307.2`.

Identity Synchronization for Windows works with Microsoft Windows password policies.

- Information about password policies for Windows 2003 is available in the Microsoft documentation online.
- Information about the Microsoft Certificate Services Enterprise Root certificate authority is available in the Microsoft support documentation online.
- Information about configuring LDAP over SSL on Microsoft systems is available in the Microsoft support documentation online.

# Redistributable Files

Directory Server Enterprise Edition does not provide any files that you can redistribute.

# Default Paths and Command Locations

This section explains the default paths used in the documentation, and gives the locations of commands on different operating systems and deployment types.

## Default Paths

The table in this section describes the default paths that are used in this document. For complete descriptions of the files installed, see the following product documentation.

- Chapter 14, "Directory Server File Reference," in *Sun Java System Directory Server Enterprise Edition 6.3 Reference*
- Chapter 25, "Directory Proxy Server File Reference," in *Sun Java System Directory Server Enterprise Edition 6.3 Reference*
- Appendix A, "Directory Server Resource Kit File Reference," in *Sun Java System Directory Server Enterprise Edition 6.3 Reference*

**TABLE P–2** Default Paths

| Placeholder | Description | Default Value |
|---|---|---|
| *install-path* | Represents the base installation directory for Directory Server Enterprise Edition software.<br>The software is installed in directories below this base *install-path*. For example, Directory Server software is installed in *install-path*`/ds6/`. | When you install from a zip distribution using `dsee_deploy`(1M), the default *install-path* is the current directory. You can set the *install-path* using the `-i` option of the `dsee_deploy` command. When you install from a native package distribution, such as you would using the Java Enterprise System installer, the default *install-path* is one of the following locations:<br>■ Solaris systems - `/opt/SUNWdsee/`.<br>■ Red Hat systems - `/opt/sun/`.<br>■ Windows systems - `C:\Program Files\Sun\JavaES5\DSEE`. |
| *instance-path* | Represents the full path to an instance of Directory Server or Directory Proxy Server.<br>The documentation uses `/local/ds/` for Directory Server and `/local/dps/` for Directory Proxy Server. | No default path exists. Instance paths must nevertheless always be found on a *local* file system.<br>The following directories are recommended:<br>`/var` on Solaris systems<br>`/global` if you are using Sun Cluster |
| *serverroot* | Represents the parent directory of the Identity Synchronization for Windows installation location | Depends on your installation. Note the concept of a *serverroot* no longer exists for Directory Server. |

**TABLE P–2** Default Paths *(Continued)*

| Placeholder | Description | Default Value |
|---|---|---|
| `isw-`*hostname* | Represents the Identity Synchronization for Windows instance directory | Depends on your installation |
| */path/to/*`cert8.db` | Represents the default path and file name of the client's certificate database for Identity Synchronization for Windows | *current-working-dir*`/cert8.db` |
| *serverroot*`/isw-`*hostname*`/logs/` | Represents the default path to the Identity Synchronization for Windows local logs for the System Manager, each connector, and the Central Logger | Depends on your installation |
| *serverroot*`/isw-`*hostname*`/logs/central/` | Represents the default path to the Identity Synchronization for Windows central logs | Depends on your installation |

# Command Locations

The table in this section provides locations for commands that are used in Directory Server Enterprise Edition documentation. To learn more about each of the commands, see the relevant man pages.

**TABLE P–3** Command Locations

| Command | Java ES, Native Package Distribution | Zip Distribution |
|---|---|---|
| `cacaoadm` | Solaris -<br>`/usr/sbin/cacaoadm` | Solaris -<br>*install-path*`/dsee6/cacao_2/usr/sbin/cacaoadm` |
| | Red Hat -<br>`/opt/sun/cacao/bin/cacaoadm` | Red Hat, HP-UX -<br>*install-path*`/dsee6/cacao_2/cacao/bin/cacaoadm` |
| | Windows -<br>*install-path*`\share\cacao_2\bin\cacaoadm.bat` | Windows -<br>*install-path*`\dsee6\cacao_2\bin\cacaoadm.bat` |

**TABLE P–3** Command Locations *(Continued)*

| Command | Java ES, Native Package Distribution | Zip Distribution |
|---|---|---|
| certutil | Solaris -<br>/usr/sfw/bin/certutil | *install-path*/dsee6/bin/certutil |
| | Red Hat -<br>/opt/sun/private/bin/certutil | |
| dpadm(1M) | *install-path*/dps6/bin/dpadm | *install-path*/dps6/bin/dpadm |
| dpconf(1M) | *install-path*/dps6/bin/dpconf | *install-path*/dps6/bin/dpconf |
| dsadm(1M) | *install-path*/ds6/bin/dsadm | *install-path*/ds6/bin/dsadm |
| dsccmon(1M) | *install-path*/dscc6/bin/dsccmon | *install-path*/dscc6/bin/dsccmon |
| dsccreg(1M) | *install-path*/dscc6/bin/dsccreg | *install-path*/dscc6/bin/dsccreg |
| dsccsetup(1M) | *install-path*/dscc6/bin/dsccsetup | *install-path*/dscc6/bin/dsccsetup |
| dsconf(1M) | *install-path*/ds6/bin/dsconf | *install-path*/ds6/bin/dsconf |
| dsee_deploy(1M) | Not provided | *install-path*/dsee6/bin/dsee_deploy |
| dsmig(1M) | *install-path*/ds6/bin/dsmig | *install-path*/ds6/bin/dsmig |
| entrycmp(1) | *install-path*/ds6/bin/entrycmp | *install-path*/ds6/bin/entrycmp |
| fildif(1) | *install-path*/ds6/bin/fildif | *install-path*/ds6/bin/fildif |
| idsktune(1M) | Not provided | At the root of the unzipped zip distribution |
| insync(1) | *install-path*/ds6/bin/insync | *install-path*/ds6/bin/insync |
| ns-accountstatus(1M) | *install-path*/ds6/bin/ns-accountstatus | *install-path*/ds6/bin/ns-accountstatus |
| ns-activate(1M) | *install-path*/ds6/bin/ns-activate | *install-path*/ds6/bin/ns-activate |
| ns-inactivate(1M) | *install-path*/ds6/bin/ns-inactivate | *install-path*/ds6/bin/ns-inactivate |
| repldisc(1) | *install-path*/ds6/bin/repldisc | *install-path*/ds6/bin/repldisc |
| schema_push(1M) | *install-path*/ds6/bin/schema_push | *install-path*/ds6/bin/schema_push |
| smcwebserver | Solaris, Linux -<br>/usr/sbin/smcwebserver | This command pertains only to DSCC when it is installed using native packages distribution. |
| | Windows -<br>*install-path*\share\webconsole\bin\smcwebserver | |

**TABLE P–3** Command Locations *(Continued)*

| Command | Java ES, Native Package Distribution | Zip Distribution |
|---|---|---|
| `wcadmin` | Solaris, Linux -<br>`/usr/sbin/wcadmin` | This command pertains only to DSCC when it is installed using native packages distribution. |
| | Windows -<br>*install-path*`\share\webconsole\bin\wcadmin` | |

# Typographic Conventions

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

**TABLE P–4** Typographic Conventions

| Typeface | Meaning | Example |
|---|---|---|
| `AaBbCc123` | The names of commands, files, and directories, and onscreen computer output | Edit your `.login` file.<br>Use `ls -a` to list all files.<br>`machine_name% you have mail.` |
| **`AaBbCc123`** | What you type, contrasted with onscreen computer output | `machine_name%` **`su`**<br>`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*.<br>A *cache* is a copy that is stored locally.<br>Do *not* save the file. |

# Shell Prompts in Command Examples

The following table shows default system prompts and superuser prompts.

**TABLE P–5** Shell Prompts

| Shell | Prompt |
|---|---|
| C shell on UNIX and Linux systems | `machine_name%` |
| C shell superuser on UNIX and Linux systems | `machine_name#` |

**TABLE P–5** Shell Prompts *(Continued)*

| Shell | Prompt |
|---|---|
| Bourne shell and Korn shell on UNIX and Linux systems | `$` |
| Bourne shell and Korn shell superuser on UNIX and Linux systems | `#` |
| Microsoft Windows command line | `C:\` |

# Symbol Conventions

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

**TABLE P–6** 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.

---

# Searching Sun Product Documentation

Besides searching for Sun product documentation from the docs.sun.com web site, you can use a search engine of your choice by typing the following syntax in the search field:

*search-term* `site:docs.sun.com`

For example, to search for `Directory Server`, type the following:

```
"Directory Server" site:docs.sun.com
```

To include other Sun web sites in your search, such as java.sun.com, www.sun.com, and developers.sun.com, use `sun.com` in place of `docs.sun.com` in the search field.

# Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. To share your comments, go to `http://docs.sun.com` and click Send Comments. In the online form, provide the full document title and part number. The part number is a 7-digit or 9-digit number that can be found on the book's title page or in the document's URL. For example, the part number of this book is 820-2760.

PART I

# Overview of Deployment Planning for Directory Server Enterprise Edition

This part provides an introduction to Directory Server Enterprise Edition and explains the steps involved in planning a deployment (solution life cycle). It contains the following chapters:

- Chapter 1, "Introduction to Deployment Planning for Directory Server Enterprise Edition," covers the deployment planning process.
- Chapter 2, "Business Analysis for Directory Server Enterprise Edition," covers business requirements.

For more information about the deployment planning process, see the Sun Java Enterprise System Deployment Planning Guide.

◆ ◆ ◆ CHAPTER 1

# Introduction to Deployment Planning for Directory Server Enterprise Edition

This chapter provides an overview of Sun Java System Directory Server Enterprise Edition, and describes at a high level the deployment planning process.

This chapter covers the following topics:



## About Directory Server Enterprise Edition

Directory Server Enterprise Edition provides secure, highly available, scalable directory services for storing and managing identity data. Directory Server Enterprise Edition is the foundation of an enterprise identity infrastructure. It enables mission-critical enterprise applications and large-scale extranet applications to access consistent and reliable identity data.

Directory Server Enterprise Edition provides a central repository for storing and managing identity profiles, access privileges, application and network resource information. Directory Server Enterprise Edition integrates smoothly into multi-platform environments. It also provides secure, on-demand synchronization of passwords, users, and groups with Microsoft Active Directory.

Prior to Directory Server Enterprise Edition, Sun provided these functions in four separate product offerings including Directory Server, Directory Proxy Server, Directory Server Resource Kit and Identity Synchronization for Windows. These and other products are now components of one comprehensive, integrated solution.

## Quality of Service Requirements for a Robust Directory Service

The more users and applications in an enterprise, the more critical is the need for a robust directory service. Directory Server Enterprise Edition addresses the challenges faced by a rapidly changing and expanding enterprise by providing the following quality of service requirements:

- **Availability.** A measure of how often the system's resources and services are accessible to end users, often expressed as the *uptime* of the system.
- **Scalability.** The ability to add capacity, and users, to a deployed system over time. Scalability typically involves adding resources to the system but should not require changes to the deployment architecture.
- **Security.** A complex combination of factors that describe the integrity of a system and its users. Security includes authentication and authorization of users, security of data, and secure access to the deployed system.
- **Interoperability.** The ease with which the system operates in conjunction with other systems.
- **Serviceability.** The ease with which a deployed system can be maintained. Maintenance tasks include monitoring the system, repairing problems that arise, and upgrading hardware and software components.

This chapter briefly describes how the components of Directory Server Enterprise Edition fill the quality of service requirements. The requirements are discussed in detail in the remainder of this guide.

# Directory Server Enterprise Edition Components and Their Capabilities

Directory Server Enterprise Edition includes these separate components:


- "Directory Server" on page 27
- "Directory Proxy Server" on page 29
- "Identity Synchronization for Windows" on page 29
- "Directory Editor" on page 30
- "Directory Server Resource Kit" on page 30


Each of these components addresses one or more of the quality of service requirements described previously. This section describes the components and illustrates how they fit together to provide a robust directory service.

# Directory Server

Directory Server provides a scalable, high-performance data store for identity information. Directory Server supports the Lightweight Directory Access Protocol (LDAP) v3 and the Directory Service Markup Language (DSML) v2 natively for standards-based access. With LDAP and DSML over HTTP or SOAP (Simple Object Access Protocol), clients anywhere on a network are able to securely search and update directory data objects. Clients are also able to receive changes made by other applications and to authenticate users or applications even through firewalls.

## Directory Server and Security

Directory Server provides several security features to achieve compliance with information security policies. These features ensure that only users with proper authorization have access to information.

- **Macro-level, dynamic access control instructions (ACIs).** Provide a means for defining access down to the level of an LDAP attribute. Access control policies can be defined once, and then reused across the directory tree. Macro ACIs can be used to optimize the number of ACIs in the directory, thereby reducing the complexity of the security framework.
- **Role-based access.** Enables you to provide access that is based on information in a user's entry. Roles are defined and administered like groups, but roles provide more efficient grouping mechanisms for applications. Roles can be used in ACIs to control access to data. They can also be used by Class of Service (CoS), a capability of Directory Server to create virtual attributes that can apply to many entries at the same time. These virtual attributes reduce storage requirements on entries. They also allow a single change to update an unlimited number of related entries.
- **Get Effective Rights control.** Provides a means for determining what access a user has to a set of information. Administrators who maintain access policies for the directory service can tighten security by auditing the permissions of directory users and applications. This capability can also be used to build applications with adaptive interfaces that are based on the user's rights.
- **Encryption mechanisms.** Protect data on the disk and during transfer through communications channels. Directory Server also supports fractional replication and data hiding based on access. These mechanisms can be used to comply with European Union and other international privacy regulations.
- **Multiple password policies.** Can be defined on a per-user basis or targeted to certain groups. These policies help to ensure that users change passwords on a regular basis and that unauthorized access to an account is blocked.

## Directory Server and Availability

Directory Server natively supports a variety of access protocols and offers a highly flexible, scalable replication environment that helps to ensure availability in distributed environments.

Directory Server replication prevents a single point of failure for applications that are using these protocols to access identity data. Directory Server supports a theoretically unlimited number of masters and read-only consumers in a replicated environment across both local and wide area networks. Special features of the replication protocol allow for optimizations when replicating data over high-latency networks. For more information, see "Using Replication and Redundancy for High Availability" on page 189.

On Solaris platforms, Directory Server supports clustering, a pre-packaged high availability hardware and software solution. For more information, see "Using Clustering for High Availability" on page 198.

## Directory Server and Scalability

Directory Server provides for both vertical and horizontal growth without major deployment redesign. This level of scalability becomes increasingly critical as deployment grows.

Depending on the hardware, Directory Server can provide sustained search performance of 20,000 entries per second on a single machine and horizontal scalability to several thousand searches per second. For information about how to deploy Directory Server for read scalability, see Chapter 10, "Designing a Scaled Deployment."

The requirement to store and update information constantly increases with the expansion of use across the organization. Update performance of Directory Server is close to relational database-write performance. For information about how to deploy Directory Server for write scalability, see Chapter 10, "Designing a Scaled Deployment."

Directory Server provides linear CPU scalability to up to 28 CPUs for "read from cache" operations. It allows access to maximum memory capacity and delivers high performance that accommodates large directories on a single system for maximum hardware benefit.

## Directory Server and Serviceability

Directory Server provides a comprehensive set of management tools for administering individual servers as well as the entire directory service.

A centralized, web-based administration console can be used to configure and manage multiple Directory Servers. The interface includes all the tools required for effective, day-to-day server administration and service from configuration to monitoring. In addition, the `dsadm` and `dsconf` command-line utilities can be used dynamically while the servers are running. These management features mean that most management operations can be performed while the directory is online, thus maximizing availability.

Management flexibility simplifies the deployment of the directory service into many different environments. The command-line utilities make remote management as easy as if the service were in a local data center.

# Directory Proxy Server

Directory Proxy Server is an LDAP application-layer protocol gateway. It is designed to deliver enhanced directory access control, schema compatibility, and high availability.

## Directory Proxy Server and Availability

With features such as configurable load balancing, failover, and failback, Directory Proxy Server ensures that systems have access to required data.

Directory Proxy Server works with Directory Server to ensure reliability and to protect against denial-of-service attacks. Directory Proxy Server automatically routes requests appropriately and provides secure firewall-like services for Directory Server.

To prevent a single point of failure for mission-critical applications, Directory Proxy Server detects outages and routes traffic around affected areas, effectively load balancing requests across systems. When the affected areas are restored to operation, Directory Proxy Server detects the restored servers automatically.

For more information, see "Using Directory Proxy Server as Part of a Redundant Solution" on page 190.

## Directory Proxy Server and Security

Directory Proxy Server accommodates large numbers of users who are accessing the directory and minimizes the security risks associated with providing this level of access. Security features enable administrators to determine where a request is coming from, whether the request is allowed, and what type of authentication is required. In the event of a search request, Directory Proxy Server can also ensure that the request meets minimum requirements.

Directory Proxy Server uses groups to define how to identify an LDAP client and what restrictions to enforce on clients that match a particular group. Groups can be defined using a variety of criteria.

To protect private directory information from unauthorized access, Directory Proxy Server can configure a fine-grained access control policy on LDAP directories. Such a policy can include controlling who can perform different types of operations on different parts of directories. Directory Proxy Server can be configured to prevent certain kinds of operations typically performed by web trawlers and robots in search of information.

# Identity Synchronization for Windows

Identity Synchronization for Windows provides basic synchronization of identity data between Directory Server Enterprise Edition and Microsoft Active Directory.

Identity Synchronization for Windows fulfills the requirement of interoperability. Synchronization of key identity data such as passwords eliminates the need for users to modify passwords several times to accommodate different application authentication mechanisms.

Use of a non intrusive implementation for synchronizing key identity data eliminates the time-consuming and maintenance-intensive need to install a client component on Active Directory servers.

Identity Synchronization for Windows enables users to change passwords and other identity data in either the Windows environment or the web-based application environment. In this way, Identity Synchronization for Windows maintains synchronization between Active Directory and Directory Server. Disabled accounts can also be synchronized between Active Directory and Directory Server. This synchronization ensures conformance of access policies to applications and data between the Windows desktops and web-based applications.

## Directory Editor

Directory Editor is a Java web application that provides efficient, cost-effective management of directory data.

Directory Editor fulfills the requirement of serviceability by enabling users to manage identity data within the directory service. Administrators can create a forms-based web interface with which users can perform everyday tasks. Directory Editor supports extensive customization, branding, and embedding for the interface. Customization is done using a form-based interface for configuration rather than writing code. To ensure data security and privacy, built-in authorization controls limit visibility of menus and actions. In this way, users see only what they are authorized to see within the application.

## Directory Server Resource Kit

The Directory Server Resource Kit provides tools and application programming interfaces (APIs) for deploying, accessing, tuning, and maintaining Directory Server Enterprise Edition. These utilities help to implement and maintain more robust LDAP-based solutions.

Performance testing and capacity planning tools help administrators to measure performance and to perform capacity planning on installations of Directory Server Enterprise Edition. Debugging and maintenance tools help with troubleshooting as well as daily maintenance of Directory Server Enterprise Edition. Deployment utilities and tools facilitate the rollout of new installations of Directory Server Enterprise Edition and migration to new releases. LDAP productivity tools include sample LDAP applications that were developed using Directory Server Enterprise Edition.

In addition, Sun has developed SLAMD, a powerful load-generation testing application that includes all the tests needed to thoroughly performance-test Directory Server Enterprise Edition applications. SLAMD is available free of charge at `http://www.slamd.com`.

# Directory Server Enterprise Edition Components in a Deployment

The combination of Directory Server Enterprise Edition components that you deploy depends on the requirements of your organization. The following figure shows a typical deployment scenario using the components described previously.

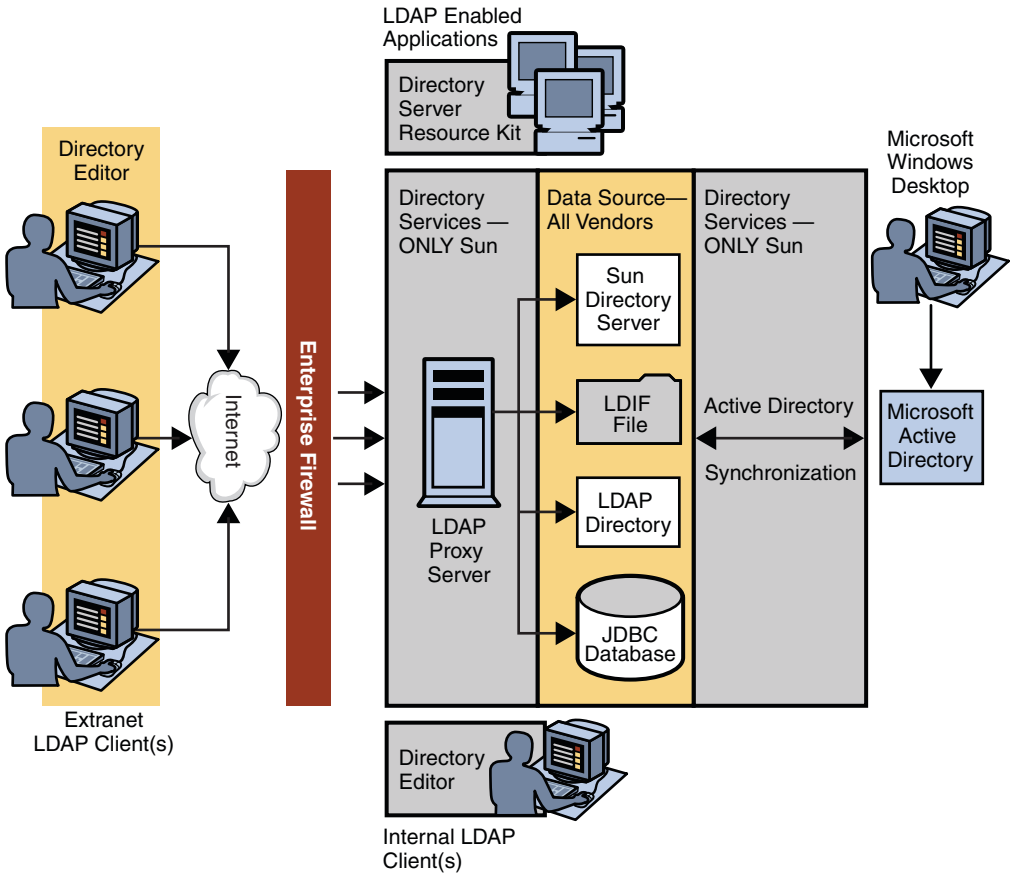



**FIGURE 1–1** Directory Server Enterprise Edition Components

# About Deployment Planning

Deployment planning is a critical step in the successful implementation of a Directory Server Enterprise Edition solution. Each enterprise has its own set of goals, requirements, and priorities to consider. Successful planning starts with analyzing the goals of an enterprise and determining the business requirements to meet those goals. The business requirements must then be converted into technical requirements. The technical requirements can be used as a basis for designing and implementing a system that meets the goals of the enterprise.

Successful deployment planning is the result of careful preparation, analysis, and design. Errors and missteps that occur anywhere during the planning process can result in a system that can be problematic in many ways. Significant problems can arise from a poorly planned system. For example, the system could under perform or be difficult to maintain, expensive to operate, or unable to scale.

The principles of deployment planning are discussed in depth in the Sun Java Enterprise System Deployment Planning Guide. This guide refers to the *solution life cycle*, which addresses deployment planning in clearly defined steps.

# Solution Life Cycle

The solution life cycle shown in the following figure depicts the steps in the planning, design, and implementation of an enterprise software solution based on Java Enterprise System. The life cycle is a useful tool for keeping a deployment project on track. The solution life cycle is described in detail in the *Sun Java Enterprise System Deployment Planning Guide*.

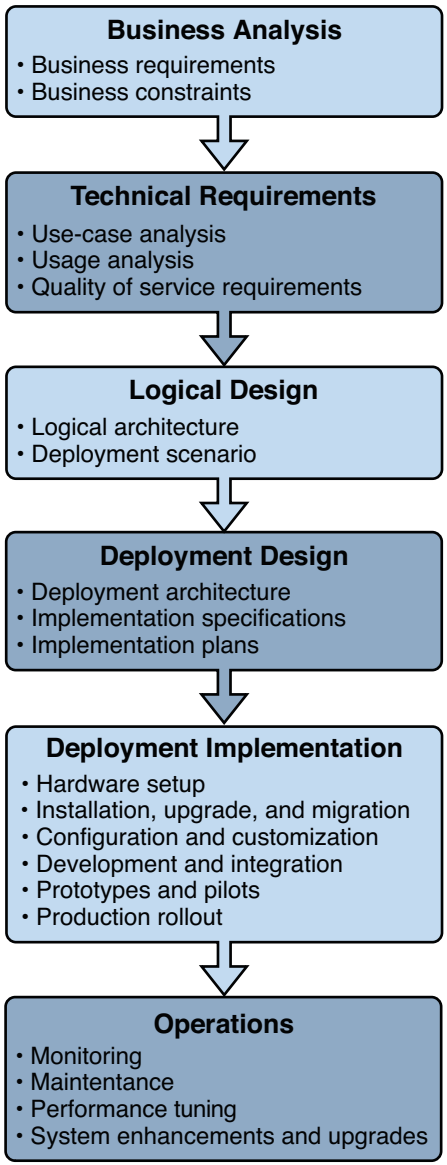



**FIGURE 1–2** Solution Life Cycle

◆ ◆ ◆ CHAPTER 2

# Business Analysis for Directory Server Enterprise Edition

During the business analysis phase of the solution life cycle, you define business goals by analyzing a business problem. You then identify the business requirements and business constraints to meet those goals.

This chapter contains the following sections:



## About Business Analysis

Business analysis starts with stating business goals. You then analyze the business problems that you must solve and identify the business requirements that must be met to achieve the business goals. Consider any business constraints that limit your ability to achieve the goals. The analysis of business requirements and constraints results in a set of business requirements documents.

You use the resulting set of business requirements documents as a basis for deriving technical requirements in the technical requirements phase. Throughout the solution life cycle, you measure the success of your planning and of your solution according to the analysis performed in the business analysis phase.

## Defining Directory Server Enterprise Edition Business Requirements

No simple formula exists to identify business requirements. Business requirements are determined based on collaboration with the stakeholders requiring an identity management solution, your own knowledge about the business domain, and applied creative thinking. The Sun Java Enterprise System Deployment Planning Guide describes the business analysis process

in detail. It includes factors to consider when defining business requirements and constraints. This section outlines the business requirements that drive the need for a robust directory service.

Your enterprise requires a robust directory service in the following situations:

- You must make critical business information and applications available to a constantly growing and changing user base.
- These users include not only internal employees but external users such as customers, vendors, and other business partners.

A directory service addresses these needs by providing a highly available, scalable, manageable, integratable, and secure foundation for an effective identity management infrastructure. The service delivers a set of capabilities to provide a centralized data store for users' identity data and for supporting data for web services architectures.

By delivering an effective identity management infrastructure, the directory service addresses the key enterprise requirements associated with serving users and the applications that help users perform their jobs.

These requirements include the following:

- Opening up access to large and constantly changing groups of users
- Increasing security to ensure that information is properly used and shared, and that sensitive information is protected
- Providing consistently reliable access and a high quality of service to users and applications
- Delivering information and services to users efficiently, no matter how business needs change or user requirements grow

A high-performing directory service that is highly available, reliable, and secure addresses the primary business drivers : security, quality of service, and cost-efficiency.

PART II

# Technical Requirements

Technical requirements analysis begins with the business requirements documents that are created during the business analysis phase of the solution life cycle. Using the business analysis, you perform a usage analysis. This analysis helps you to determine expected load conditions and to create use cases that model typical user interaction with the system. The analysis also helps when creating a set of quality of service requirements. These requirements define how a deployed solution must perform in areas such as response time, availability, and security.

This part describes the technical requirements that must be defined for a Directory Server Enterprise Edition deployment. It is divided into the following chapters:

- Chapter 3, "Usage Analysis for Directory Server Enterprise Edition," covers usage analysis requirements.
- Chapter 4, "Defining Data Characteristics," describes how data requirements are defined.
- Chapter 5, "Defining Service Level Agreements," covers quality of service requirements.
- Chapter 6, "Tuning System Characteristics and Hardware Sizing," describes Directory Server Enterprise Edition system requirements.
- Chapter 7, "Identifying Security Requirements," covers security requirements.
- Chapter 8, "Identifying Administration and Monitoring Requirements," describes the administration decisions that must be made at design-time.

◆ ◆ ◆ CHAPTER 3

# Usage Analysis for Directory Server Enterprise Edition

Usage analysis involves identifying the users of your system and determining the usage patterns for those users. In doing so, a usage analysis enables you to determine expected load conditions on your directory service.

## Usage Analysis Factors

Your reasons for offering Sun Java System Directory Server Enterprise Edition as an identity management solution have a direct effect on how you deploy the server.

During usage analysis, interview users whenever possible. Research existing data on usage patterns, and interview builders and administrators of previous systems. A usage analysis should provide you with the data that enables you to determine the service requirements that are described in Chapter 5, "Defining Service Level Agreements."

The information that should come out of a usage analysis includes the following:

- **Number and type of client applications.** Identify how many client applications your deployment must support, and categorize those applications, if necessary.
- **Administrative users.** Identify users who access the directory to monitor, update, and support its deployment. Determine any specific administrative usage patterns that might affect technical requirements, for example, administration of the deployment from outside the firewall.
- **Usage patterns.** Identify how various types of applications access the system, and provide targets for expected usage.

  Answer the following questions, for example:

  - Are there times when usage spikes?
  - What are usual business hours?
  - Are client applications distributed globally?
  - What is the expected duration of application connectivity?

- **Client application growth.** Determine if the number of client applications is fixed or expected to grow. If you anticipate additional applications, try to create reasonable projections of the growth.
- **Application transactions.** Identify the types of transactions that must be supported.

  These transactions can be categorized into use cases, for example:

  - What tasks are performed by the applications?
  - When applications bind to the directory, do they remain bound, or do they typically perform a few tasks and unbind?
- **Studies and statistical data.** Use preexisting studies and other sources to determine patterns of application behavior. Often, enterprises or industry organizations have research studies from which you can extract useful information about users and client applications. Log files for existing applications might contain statistical data that is useful for making estimates for a system.

For more information about usage analysis, see the *Sun Java Enterprise System Deployment Planning Guide*.

◆ ◆ ◆ CHAPTER 4

# Defining Data Characteristics

The type of data in your directory determines how you structure the directory, who can access the data, and how access is granted. Data types can include, among others, user names, email addresses, telephone numbers, and information about groups to which users belong.

This chapter explains how to locate, categorize, structure, and organize data. It also explains how to map data to the Directory Server schema. This chapter covers the following topics:



## Determining Data Sources and Ownership

The first step in categorizing existing data is to identify where that data comes from and who owns it.

### Identifying Data Sources

To identify the data to be included in your directory, locate and analyze existing data sources.

- Identify organizations that provide information.

  Locate all the organizations that manage information essential to your enterprise. Typically, these organizations include your information services, human resources, payroll, and accounting departments.
- Identify tools and processes that are information sources.

Common sources for information include the following:

- Networking operating systems, such as Windows, Novell Netware, and UNIX® NIS
- Email systems
- Security systems
- PBX or telephone switching systems
- Human resources applications

- Determine how centralizing each piece of data affects the management of data.

  Centralized data management might require new tools and new processes. Issues can arise when centralization requires increasing staff in some organizations and decreasing staff in others.

## Determining Data Ownership

*Data ownership* refers to the person or organization that is responsible for ensuring that data is up-to-date. During the data design phase, decide who can write data to the directory. Common strategies for determining data ownership include the following:

- Allow read-only access to the directory for everyone except a small group of directory content managers.
- Allow individual users to manage strategic subsets of information.

  These subsets of information might include their passwords, descriptive information about themselves, and their role within the organization.
- Allow a person's manager to write to some strategic subset of that person's information, such as contact information or job title.
- Allow an organization's administrator to create and manage entries for that organization.

  Organization administrators in effect become your directory content managers.
- Create roles that give groups of people read or write access privileges.

  For example, you might create roles for human resources, finance, or accounting. Allow each of these roles to have read access, write access, or both to the data needed by the group. This data might include salary information, government identification number, and home phone numbers and address.

  For more information about roles and grouping entries, see "Grouping Directory Data and Managing Attributes" on page 48, Chapter 10, "Directory Server Groups, Roles, and CoS," in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide* and Chapter 8, "Directory Server Groups and Roles," in *Sun Java System Directory Server Enterprise Edition 6.3 Reference*.

As you determine who can write to the data, you might find that multiple individuals require write access to the same information. For example, an information systems or directory management group should have write access to employee passwords. You might also want all

employees to have write access to their own passwords. While you generally must give multiple people write access to the same information, try to keep this group small and easy to identify. Small groups help to ensure your data's integrity.

For information about setting access control for your directory, see Chapter 7, "Directory Server Access Control," in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide* and "How Directory Server Provides Access Control" in *Sun Java System Directory Server Enterprise Edition 6.3 Reference*.

## Distinguishing Between User and Configuration Data

To distinguish between data used to configure Directory Server and other Java Enterprise System servers and the actual user data stored in the directory, do the following:

- Provide different backup strategies for user and configuration data.
- Provide different high availability standards for user and configuration data.
- Shut down, restore, and power up configuration servers quickly.
- Keep configuration servers up while performing maintenance on other Directory Server instances.

# Identifying Data From Disparate Data Sources

When determining data sources, ensure that you include data from other data sources, including legacy data sources. This data might not be stored in the directory. However, Directory Server might need to have some knowledge of, or control over, the data.

Directory Proxy Server provides a *virtual directory* feature that aggregates information, in real-time, from multiple data repositories. These repositories include LDAP directories, data that complies with the JDBC specification, and LDIF flat files.

The virtual directory supports complex filters that handle attributes from different data sources. It also supports modifications that combine attributes from different data sources.

During the data analysis phase, you might find that the same data is required by several applications, but in a different format. Instead of duplicating this information, it is preferable to have the applications transform it for their requirements.

# Structuring Data With the Directory Information Tree

The directory information tree (DIT) provides a way to structure directory data so that the data can be referred to by client applications. The DIT interacts closely with other design decisions, including how you distribute, replicate, or control access to directory data.

## DIT Terminology

A well-designed DIT provides the following:

- Simplified directory data maintenance
- Flexibility in creating replication policies and access controls
- Support for the applications that use the directory
- Simplified directory navigation for users

The DIT structure follows the hierarchical LDAP model. The DIT organizes data, for example, by group, by people, or by geographical location. It also determines how data is partitioned across multiple servers.

DIT design has an impact on replication configuration and on how you use Directory Proxy Server to distribute data. If you want to replicate or distribute certain portions of a DIT, consider replication and the requirements of Directory Proxy Server at design time. Also, decide at design time whether you require access controls on branch points.

A DIT is defined in terms of suffixes, subsuffixes, and chained suffixes. A *suffix* is a branch or subtree whose entire contents are treated as a unit for administrative tasks. Indexing is defined for an entire suffix, and an entire suffix can be initialized in a single operation. A suffix is also usually the unit of replication. Data that you want to access and manage in the same way should be located in the same suffix. A suffix can be located at the root of the directory tree, where it is called a *root suffix*.

Because data can only be partitioned at the suffix level, an appropriate directory tree structure is required to spread data across multiple servers.

The following figure shows a directory with two root suffixes. Each suffix represents a separate corporate entity.

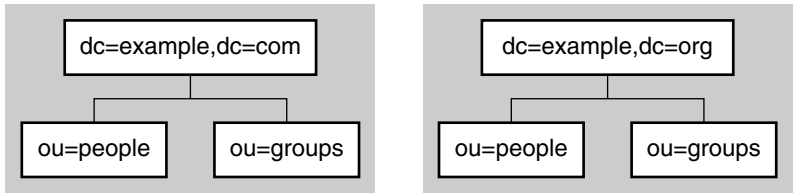



**FIGURE 4–1** Two Root Suffixes in a Single Directory Server

A suffix might also be a branch of another suffix, in which case it is called a *subsuffix*. The parent suffix does not include the contents of the subsuffix for administrative operations. The subsuffix is managed independently of its parent. Because LDAP operation results contain no information about suffixes, directory clients are unaware of whether entries are part of root suffixes or subsuffixes.

The following figure shows a directory with a single root suffix and multiple subsuffixes for a large corporate entity.

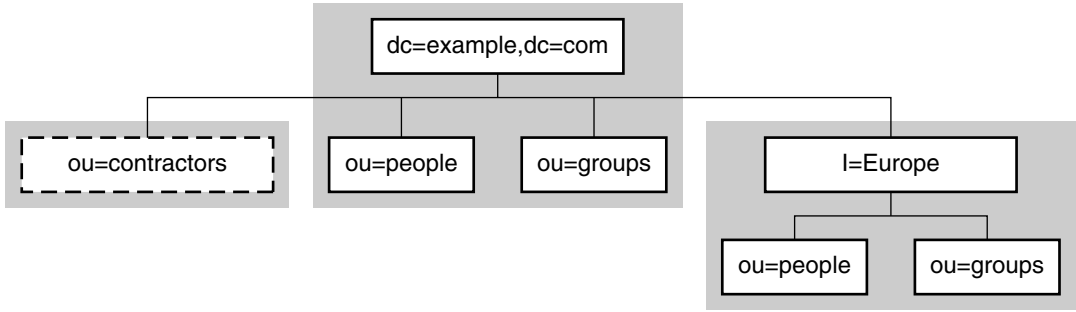



**FIGURE 4–2** One Root Suffix With Multiple Subsuffixes

A suffix corresponds to an individual database within the server. However, databases and their files are managed internally by the server and database terminology is not used.

Chained suffixes create a virtual DIT by referencing suffixes on other servers. With chained suffixes, Directory Server performs the operation on the remote suffix. The directory then returns the result as if the operation had been performed locally. The location of the data is transparent. The client is unaware that the suffix is chained and that the data is retrieved from a remote server. A root suffix on one server can have subsuffixes that are chained to another server. In this scenario, the client is aware of a single tree structure.

In the special case of cascading chaining, the chained suffix might reference another chained suffix on the remote server, and so on. Each server forwards the operation and eventually returns the result to the server that handles the client's request.

# Designing the DIT

DIT design involves choosing a suffix to contain your data, determining the hierarchical relationship between data entries, and naming the entries in the DIT hierarchy. The following sections describe the design process in more detail.

## Choosing a Suffix

The suffix is the name of the entry at the root of the DIT. If you have two or more DITs that do not have a natural common root, you can use multiple suffixes. The default Directory Server installation contains multiple suffixes. One suffix is used to store user data. The other suffixes are for data that is needed by internal directory operations, such as configuration information and directory schema.

All directory entries must be located below a common base entry, the suffix. Each suffix name must be as follows:

- Globally unique
- Static, so that the name rarely changes
- Short, so that entries beneath the suffix are easier to read online
- Easy for a person to type and remember

It is generally considered best practice to map your enterprise domain name to a Distinguished Name (DN). For example, an enterprise with the domain name `example.com` would use a DN of `dc=example,dc=com`.

## Creating the DIT Structure and Naming Entries

The structure of a DIT can be flat or hierarchical. Although a flat tree is easier to manage, a degree of hierarchy might be required for data partitioning, replication management, and access control.

### Branch Points and Naming Considerations

A *branch point* is a point at which you define a new subdivision within the DIT. When deciding on branch points, avoid potential problematic name changes. The likelihood of a name changing is proportional to the number of components in the name that can potentially change. The more hierarchical the DIT, the more components in the names, and the more likely the names are to change.

Use the following guidelines when defining and naming branch points:

- Branch your tree to represent only the largest organizational subdivisions in your enterprise.

  Limit branch points to divisions, such as Corporate Information Services, Customer Support, Sales, and Professional Services. Make sure that your divisions are stable. Do not perform this kind of branching if your enterprise reorganizes frequently.
- Use functional or generic names rather than actual organizational names.

  Names change and you do not want to have to change your DIT every time your enterprise renames its divisions. Instead, use generic names that represent the function of the organization. For example, use `Engineering` instead of `Widget Research and Development`.
- If you have multiple organizations that perform similar functions, create a single branch point for that function instead of branching based on divisional lines.

  For example, even if you have multiple marketing organizations that are responsible for a specific product line, create a single Marketing subtree. All marketing entries then belong to that tree.
- Try to use only the traditional branch point attributes that are shown in the following table.

  Traditional attributes increase the likelihood of retaining compatibility with third-party LDAP client applications. In addition, traditional attributes are known to the default directory schema, which simplifies the construction of entries for the branch distinguished name (DN).
- Branch according to the type of data stored in the directory.

  For example, you might create a separate branch for people, groups, service, and devices.

**TABLE 4–1** Traditional DN Branch Point Attributes

| Attribute Name | Definition |
|---|---|
| `c` | A country name. |
| `o` | An organization name. This attribute is typically used to represent a large divisional branching. The branching might include a corporate division, academic discipline, subsidiary, or other major branching within the enterprise. You should also use this attribute to represent a domain name. |
| `ou` | An organizational unit. This attribute is typically used to represent a smaller divisional branching of your enterprise than an organization. Organizational units are generally subordinate to the preceding organization. |
| `st` | A state or province name. |
| `l` | A locality, such as a city, country, office, or facility name. |
| `dc` | A domain component. |

Be consistent when choosing attributes for branch points. Some LDAP client applications might fail if the DN format is inconsistent across your DIT. If `l (localityName)` is subordinate to `o (organizationName)` in one part of your DIT, ensure that `l` is subordinate to `o` in all other parts of your directory.

### Replication Considerations

When designing a DIT, consider which entries will be replicated to other servers. If you want to replicate a specific group of entries to the same set of servers, those entries should fall below a specific subtree. To describe the set of entries to be replicated, specify the DN at the top of the subtree. For more information about replicating entries, see Chapter 4, "Directory Server Replication," in *Sun Java System Directory Server Enterprise Edition 6.3 Reference*.

### Access Control Considerations

A DIT hierarchy can enable certain types of access control. As with replication, it is easier to group similar entries and to administer the entries from a single branch.

A hierarchical DIT also enables distributed administration. For example, you can use the DIT to give an administrator from the marketing department access to marketing entries, and an administrator from the sales department access to sales entries.

You can also set access controls based on directory content, rather than the DIT. Use the ACI filtered target mechanism to define a single access control rule. This rule states that a directory entry has access to all entries that contain a particular attribute value. For example, you can set an ACI filter that gives the sales administrator access to all entries that contain the attribute `ou=Sales`.

However, ACI filters can be difficult to manage. You must decide which method of access control is best suited to your directory: organizational branching in the DIT hierarchy, ACI filters, or a combination of the two.

## Grouping Directory Data and Managing Attributes

The directory information tree organizes entries hierarchically. This hierarchy is a type of grouping mechanism. The hierarchy is not well suited for associations between dispersed entries, for organizations that change frequently, or for data that is repeated in many entries. Directory Server groups and roles offer more flexible associations between entries. The class of service (CoS) mechanism enables you to manage attributes so that the attributes are shared between entries. This sharing is done in a way that is invisible to applications.

These entry grouping and attribute management mechanisms are described in detail in Chapter 8, "Directory Server Groups and Roles," in *Sun Java System Directory Server Enterprise Edition 6.3 Reference* and in Chapter 9, "Directory Server Class of Service," in *Sun Java System Directory Server Enterprise Edition 6.3 Reference*.

This section provides an overview of the grouping mechanisms that is sufficient to design an administrative strategy. It does not explain how the mechanisms work or how to set them up.

The section is divided into the following topics:



# Static, Dynamic, and Nested Groups

Directory Server distinguishes among the static, dynamic, and nested groups.

Although groups may identify members anywhere in the directory, the group definitions themselves should be located under an appropriately named node such as `ou=Groups`. This makes them easy to find, for example, when defining access control instructions (ACIs) that grant or restrict access when the bind credentials are members of a group.

## Static Groups

Static groups explicitly name their member entries. For example, a group of directory administrators would name the specific people who formed part of that group, as shown in the following illustration.

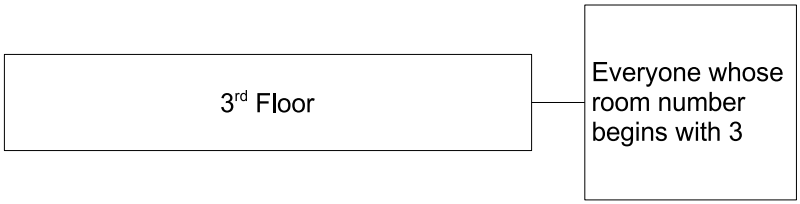

The following LDIF extract shows how the members of this static group would be defined.

```
dn: cn=Directory Administrators, ou=Groups, dc=example,dc=com
...
member: uid=kvaughan, ou=People, dc=example,dc=com
member: uid=rdaugherty, ou=People, dc=example,dc=com
member: uid=hmiller, ou=People, dc=example,dc=com
```

## Dynamic Groups

Dynamic groups specify a filter and all entries that match the filter are members of the group. These groups are dynamic because membership is defined each time the filter is evaluated.

Imagine, for example, that all management employees and their assistants were situated on the 3rd floor of your building, and that the room number of each employee commenced with the number of the floor. If you wanted to create a group containing just the employees on the third floor, you could use the room number to define just these employees, as shown in the following illustration.



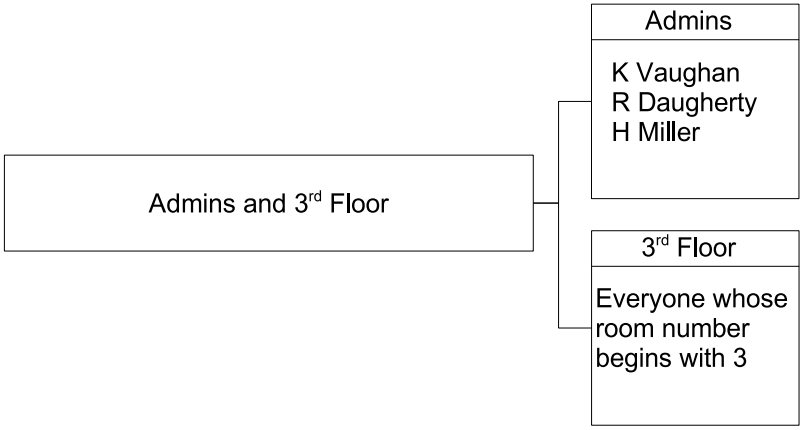

The following LDIF extract shows how the members of this dynamic group would be defined.

```
dn: cn=3rd Floor, ou=Groups, dc=example,dc=com
...
memberURL: ldap:///dc=example,dc=com??sub?(roomnumber=3*)
```

## Nested Groups

Nested groups use the DN of another group as the `uniqueMember` attribute of a static or dynamic group to place groups inside other groups. Directory Server also supports mixed groups, that is groups that reference individual entries, static groups, and dynamic groups.

Imagine for example that you wanted a group containing all directory administrators, and all management employees and their assistants. You could use a combination of the two groups defined earlier to create one nested group, as shown in the following illustration.




The following LDIF extract shows how the members of this nested group would be defined.

```
dn: cn=Admins and 3rd Floor, ou=Groups, dc=example,dc=com
...
member: cn=Directory Administrators, ou=Groups, dc=example,dc=com
member: cn=3rd Floor, ou=Groups, dc=example,dc=com
```

Nested groups are not the most efficient grouping mechanism. Dynamic nested groups incur an even greater performance cost. To avoid these performance problems, consider using roles instead.

## Managed, Filtered, and Nested Roles

Roles are an entry grouping mechanism. Roles enable you to determine role membership as soon as an entry is retrieved from the directory. Each role has *members*, or entries that possess the role. As with groups, you can specify role members explicitly or dynamically.

Directory Server supports the following three types of roles:

- **Managed roles.** Explicitly assign a role to member entries.
- **Filtered roles.** Automatically make entries members if the entries match a specified LDAP filter. In this way, the role depends on the attributes contained in each entry.
- **Nested roles.** Enable you to create roles that contain other roles.

## Deciding Between Groups and Roles

The functionality of the groups and roles mechanisms overlap somewhat. Both mechanisms have advantages and disadvantages. Generally, the roles mechanism is designed to provide frequently required functionality more efficiently. Because the choice of a grouping mechanism influences server complexity and determines how clients process membership information, you must plan your grouping mechanism carefully. To decide which mechanism is more suitable, you need to understand the typical membership queries and management operations that are performed.

## Advantages of the Groups Mechanism

Groups have the following advantages:

- Static groups are the only standards-based grouping mechanism. Static groups are therefore interoperable with most client applications and LDAP servers.
- Static groups are preferable to roles for enumerating members.

  If you *only* need to enumerate members of a given set, static groups are less costly. Enumerating members of a static group by retrieving the `member` attribute is easier than recovering all entries that share a role. In Directory Server, significant performance improvements have been made for large multi-valued attributes. Equality matching and modify operations on these attributes are greatly improved, specifically in relation to static groups. Membership testing for group entries has also been improved. These improvements remove some of the previous restrictions on static groups, specifically the restriction on group size.

  Directory Server also provides group membership directly in user entries, with the `isMemberOf` operational attribute. This feature applies to static groups only but includes nested groups. For more information, see "Managing Groups" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.
- Static groups are preferable to roles for management operations such as assigning and removing members.

  Static groups are the simplest mechanism for assigning a user to a set or removing a user from a set. Special access rights are not required to add the user to the group.

  The right to create the group entry automatically gives you the right to assign members to that group. This is not the case for managed and filtered roles. In these roles, the administrator must also have the right to write the `nsroledn` attribute to the user entry. The same access right restrictions also apply indirectly to nested roles. The ability to create a nested role implies the ability to pull together other roles that have already been defined.
- Dynamic groups are preferable to roles for use in filter-based ACIs.

  If you *only* need to find all members based on a filter, such as for designating bind rules in ACIs, use dynamic groups. Although filtered roles are similar to dynamic groups, filtered roles trigger the roles mechanism and generate the virtual `nsRole` attribute. If your client does not need the `nsRole` value, use dynamic groups to avoid the overhead of this computation.
- Groups are preferable to roles for adding or removing sets into or from existing sets.

  If you want to add a set to an existing set, or remove a set from an existing set, the groups mechanism is simplest. The groups mechanism presents no nesting restrictions. The roles mechanism only allows nested roles to receive other roles.
- Groups are preferable to roles if flexibility of scope for grouping entries is critical.

Groups are flexible in terms of scope because the scope for possible members is the entire directory, regardless of where the group definition entries are located. Although roles can also extend their scope beyond a given subtree, they can only do so by adding the scope-extending attribute `nsRoleScopeDN` to a nested role.

## Advantages of the Roles Mechanism

Roles have the following advantages:

- Roles are preferable to dynamic groups if you want to enumerate members of a set *and* find all sets of which a given entry is a member. Static groups also provide this functionality with the `isMemberOf` attribute.

  Roles push membership information out to the user entry where this information can be cached to make subsequent membership tests more efficient. The server performs all computations, and the client only needs to read the values of the `nsRole` attribute. In addition, all types of roles appear in this attribute, allowing the client to process all roles uniformly. Roles can perform both operations more efficiently and with simpler clients than is possible with dynamic groups.
- Roles are preferable to groups if you want to integrate your grouping mechanism with existing Directory Server functionality such as CoS, Password Policy, Account Inactivation, and ACIs.

  If you want to use the membership of a set "naturally" in the server, roles are a better option. This implies that you use the membership computations that the server does automatically. Roles can be used in resource-oriented ACIs, as a basis for CoS, as part of more complex search filters, and with Password Policy, Account Inactivation, and so forth. Groups do not allow this kind of integration.

## Restricting Permissions on Roles

Be aware of the following issues when using roles:

- The `nsRole` attribute can only be assigned by the roles mechanism. While this attribute cannot be assigned or modified by any directory user, it is potentially *readable* by any directory user. Define access controls to keep this attribute from being read by unauthorized users.
- The `nsRoleDN` attribute defines managed role membership. You need to decide whether users can add or remove themselves from the role. To keep from modifying their own roles, you must define an ACI to that effect.
- Filtered roles determine membership through filters that are based on the existence or the values of attributes in user entries. Assign the user permissions of these attributes carefully to control who can define membership in the filtered role.

# Managing Attributes With Class of Service

The Class of Service (CoS) mechanism allows attributes to be shared between entries. Like the role mechanism, CoS generates virtual attributes on the entries as the entries are retrieved. CoS does not define membership, but it does allow related entries to share data for coherency and space considerations. CoS values are calculated dynamically when the values are requested. CoS functionality and the various types of CoS are described in detail in the *Sun Java System Directory Server Enterprise Edition 6.3 Reference*.

The following sections examine the ways in which you can use the CoS functionality as intended, while avoiding performance pitfalls:


- "Using CoS When Many Entries Share the Same Value" on page 54
- "Using CoS When Entries Have Natural Relationships" on page 55
- "Avoiding Excessive CoS Definitions" on page 58


---

**Note –** CoS generation always impacts performance. Client applications that search for more attributes than they actually need can compound the problem.

If you can influence how client applications are written, remind developers that client applications perform much better when looking up only those attribute values that they actually need.

---

## Using CoS When Many Entries Share the Same Value

CoS provides substantial benefits for relatively low cost when you need the same attribute value to appear on numerous entries in a subtree.

Imagine, for example, a directory for MyCompany, Inc. in which every user entry under `ou=People` has a `companyName` attribute. Contractors have real values for `companyName` attributes on their entries, but all regular employees have a single CoS-generated value, `MyCompany, Inc.`, for `companyName`. The following figure demonstrates this example with pointer CoS. Notice that CoS generates `companyName` values for all permanent employees without overriding real, not CoS-generated, `companyName` values stored for contractor employees. The company name is generated only for those entries for which `companyName` is an allowed attribute.

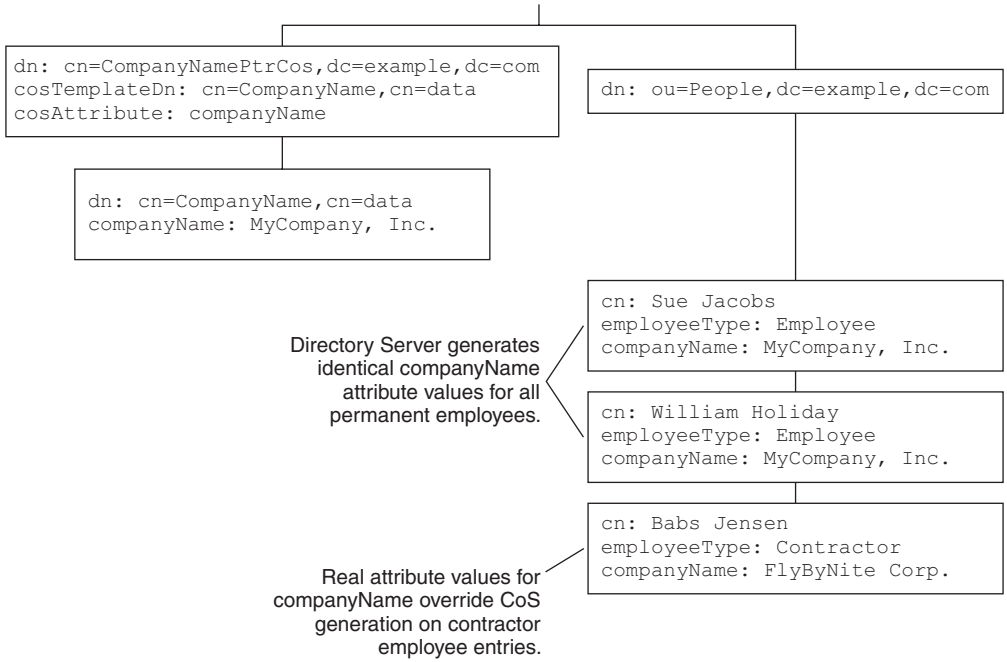



**FIGURE 4–3** Generating `companyName` With Pointer CoS

In cases where many entries share the same value, pointer CoS works particularly well. The ease of maintaining `companyName` for permanent employees offsets the additional processing cost of generating attribute values. Deep directory information trees (DITs) tend to bring together entries that share common characteristics. Pointer CoS can be used in deep DITs to generate common attribute values by placing CoS definitions at appropriate branches in the tree.

## Using CoS When Entries Have Natural Relationships

CoS also provides substantial data administration benefits when directory data has natural relationships.

Consider an enterprise directory in which every employee has a manager. Every employee shares a mail stop and fax number with the nearest administrative assistant. Figure 4–4 demonstrates the use of indirect CoS to retrieve the department number from the manager entry. In Figure 4–5, the mail stop and fax number are retrieved from the administrative assistant entry.

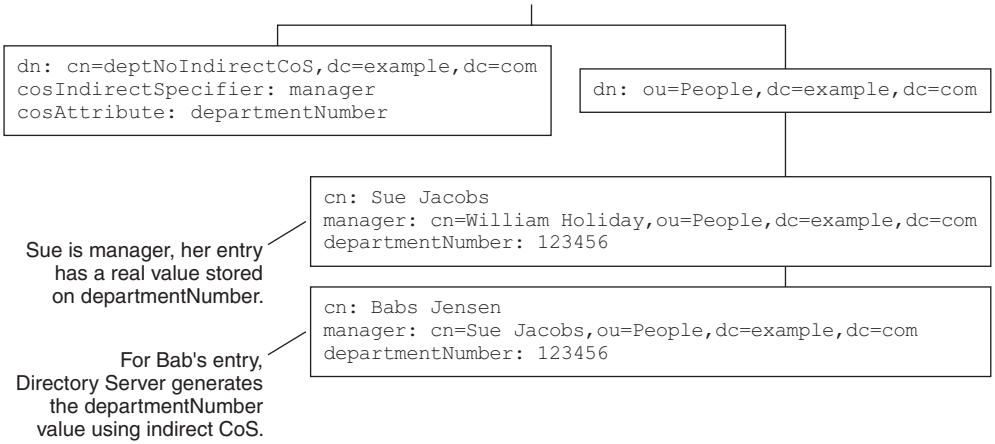



**FIGURE 4–4** Generating `DepartmentNumber` With Indirect CoS

In this implementation, the manager's entry has a real value for `departmentNumber`, and this real value overrides any generated value. Directory Server does not generate attribute values from CoS-generated attribute values. Thus, in the Figure 4–4 example, the department number attribute value needs to be managed only on the manager's entry. Likewise, for the example shown in Figure 4–5, mail stop and fax number attributes need to be managed only on the administrative assistant's entry.

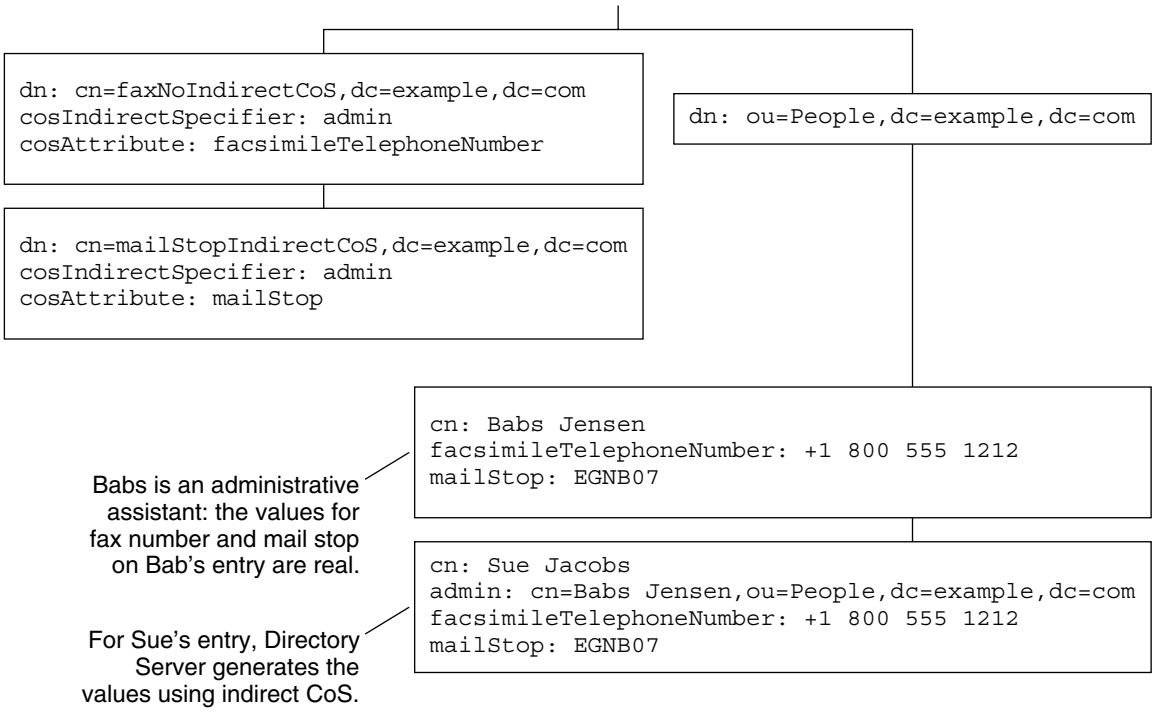



**FIGURE 4–5** Generating Mail Stop and Fax Number With Indirect CoS

A single CoS definition entry can be used to exploit relationships such as these for many different entries in the directory.

Another natural relationship is service level. Consider an Internet service provider that offers customers standard, silver, gold, and platinum packages. A customer's disk quota, number of mailboxes, and rights to prepaid support levels depend on the service level purchased. The following figure demonstrates how a classic CoS scheme enables this functionality.

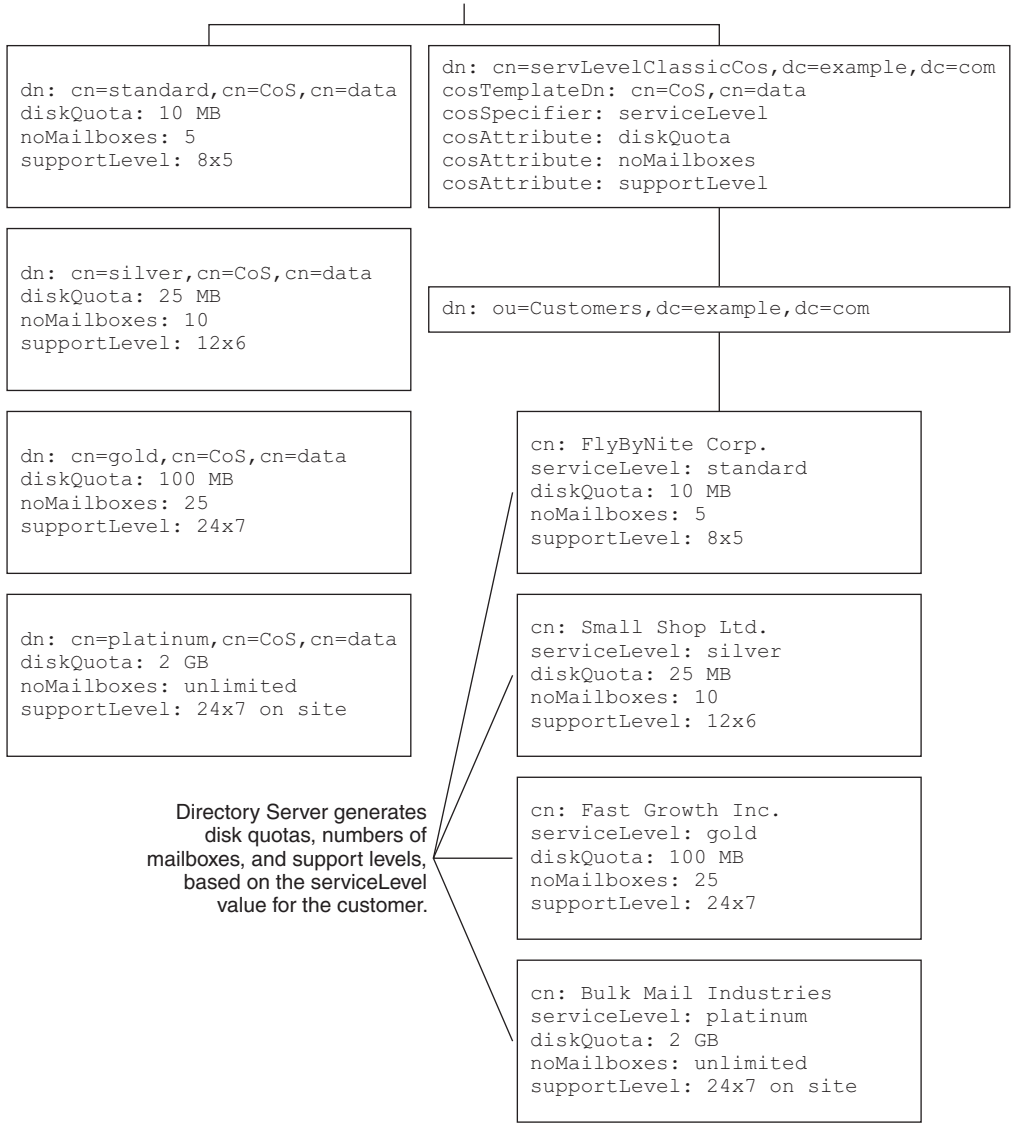



**FIGURE 4–6** Generating Servic-Level Data With Classic CoS

One CoS definition might be associated with multiple CoS template entries.

## Avoiding Excessive CoS Definitions

Directory Server optimizes CoS when one classic CoS definition entry is associated with multiple CoS template entries. Directory Server does not optimize CoS if many CoS definitions

potentially apply. Instead, Directory Server checks each CoS definition to determine whether the definition applies. This behavior leads to performance problems if you have thousands of CoS definitions.

This situation can arise in a modified version of the example shown in Figure 4–6. Consider an Internet service provider that offers customers delegated administration of their customers' service level. Each customer provides definition entries for standard, silver, gold, and platinum service levels. Ramping up to 1000 customers means creating 1000 classic CoS definitions. Directory Server performance would be affected as it runs through the list of 1000 CoS definitions to determine which apply. If you must use CoS in this sort of situation, consider indirect CoS. In indirect CoS, customers' entries identify the entries that define their class of service allotments.

When you start approaching the limit of having different CoS schemes for every target entry or two, you are better off updating the real values. You then achieve better performance by reading real, not CoS-generated values.

# Designing a Directory Schema

The directory schema describes the types of data that can be stored in a directory. During schema design, each data element is mapped to an LDAP attribute. Related elements are gathered into LDAP object classes. A well-designed schema helps maintain data integrity by imposing constraints on the size, range, and format of data values. You decide what types of entries your directory contains and the attributes that are available to each entry.

The predefined schema that is included with Directory Server contains the Internet Engineering Task Force (IETF) standard LDAP schema. The schema contains additional application-specific schema to support the features of the server. It also contains Directory Server-specific schema extensions. While this schema meets most directory requirements, you might need to extend the schema with new object classes and attributes that are specific to your directory.

## Schema Design Process

Schema design involves doing the following:

- Mapping your data to the default schema.

  To map existing data to the default schema, identify the type of object that each data element describes then select a similar object class from the default schema. Use the common object classes, such as groups, people, and organizations. Select a similar attribute from the matching object class that best matches the data element.
- Identifying unmatched data.
- Extending the default schema to define new elements to meet your remaining needs.

If data elements exist that do not match the object classes and attributes defined by the default directory schema, you can customize the schema. You can also extend the schema to impose additional constraints on the existing schema. For more information, see "About Custom Schema" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

- Planning for schema maintenance.

Where possible, use the existing schema elements that are defined in the default Directory Server schema. Standard schema elements help to ensure compatibility with directory-enabled applications. Because the schema is based on the LDAP standard, it has been reviewed and agreed to by a large number of directory users.

## Maintaining Data Consistency

Consistent data assists LDAP client applications in locating directory entries. For each type of information that is stored in the directory, select the required object classes and attributes to support that information. Always use the same object classes and attributes. If you use schema objects inconsistently, it is difficult to locate information.

You can maintain schema consistency in the following ways:

- Use schema checking to ensure that attributes and object classes conform to the schema rules.

  For more information about schema checking, see Chapter 12, "Directory Server Schema," in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

- Select and apply a consistent data format.

  The LDAP schema allows you to place any data on any attribute value. However, you should store data consistently in the DIT by selecting a format appropriate for your LDAP client applications and directory users. With the LDAP protocol and Directory Server, you must represent data using the data formats specified in RFC 4517.

# Other Directory Data Resources

For more information about the standard LDAP schema, and about designing a DIT, see the following sites:

- RFC 4510: Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map
  `http://www.ietf.org/rfc/rfc4510.txt`
- RFC 4511: Lightweight Directory Access Protocol (LDAP): The Protocol
  `http://www.ietf.org/rfc/rfc4511.txt`
- RFC 4512: Lightweight Directory Access Protocol (LDAP): Directory Information Models
  `http://www.ietf.org/rfc/rfc4512.txt`

- RFC 4513: Lightweight Directory Access Protocol (LDAP): Authentication Methods and Security Mechanisms
  http://www.ietf.org/rfc/rfc4513.txt
- RFC 4514: Lightweight Directory Access Protocol (LDAP): String Representation of Distinguished Names
  http://www.ietf.org/rfc/rfc4514.txt
- RFC 4515: Lightweight Directory Access Protocol (LDAP): String Representation of Search Filters
  http://www.ietf.org/rfc/rfc4515.txt
- RFC 4516: Lightweight Directory Access Protocol (LDAP): Uniform Resource Locator
  http://www.ietf.org/rfc/rfc4516.txt
- RFC 4517: Lightweight Directory Access Protocol (LDAP): Syntaxes and Matching Rules
  http://www.ietf.org/rfc/rfc4517.txt
- RFC 4518: Lightweight Directory Access Protocol (LDAP): Internationalized String Preparation
  http://www.ietf.org/rfc/rfc4518.txt
- RFC 4519: Lightweight Directory Access Protocol (LDAP): Schema for User Applications
  http://www.ietf.org/rfc/rfc4519.txt
- *Understanding and Deploying LDAP Directory Services.* T. Howes, M. Smith, G. Good. Macmillan Technical Publishing, 1999

For a complete list of the RFCs and standards supported by Directory Server Enterprise Edition, see Appendix A, "Standards and RFCs Supported by Directory Server Enterprise Edition," in *Sun Java System Directory Server Enterprise Edition 6.3 Evaluation Guide*.

◆ ◆ ◆ CHAPTER 5

# Defining Service Level Agreements

Service level agreements are technical specifications that determine how the system must perform under certain conditions. This chapter describes the service requirements that are specific to Directory Server Enterprise Edition. The chapter includes questions that you need to ask during the planning phase to ensure that your deployment meets these requirements.

This chapter covers the following topics:



## Identifying System Qualities

To identify system qualities, specify the minimum requirements that your directory service must provide. The following system qualities typically form a basis for quality of service requirements:

- **Performance.** The measurement of response time and throughput with respect to user load conditions.
- **Availability.** A measure of how often a system's resources and services are accessible to end users, often expressed as the uptime of a system.
- **Scalability.** The ability to add capacity and users to a deployed system over time. Scalability typically involves adding resources to the system without changing the deployment architecture.

- **Security.**A complex combination of factors that describe the integrity of a system and its users. Security includes authentication and authorization of users, security of data, and secure access to a deployed system.
- **Latent capacity.** The ability of a system to handle unusual peak loads without additional resources. Latent capacity is a factor in availability, performance, and scalability.
- **Serviceability.** The ease by which a deployed system can be maintained, including monitoring the system, fixing problems that arise, and upgrading hardware and software components.

# Defining Performance Requirements

Performance requirements should be based on typical models of directory usage. In all directory deployments, Directory Server supports one or more client applications, and the requirements of these applications must be assessed. Estimating how much information your directory contains, and how often that information is accessed, involves identifying these applications and determining how they use Directory Server.

## Identifying Client Applications

The applications that access your directory and the data needs of these applications have a significant impact on performance requirements. When identifying client applications, consider the following:

- What types of client applications are accessing Directory Server?
- How many users access each of these applications?
- What kind of operations do these applications perform?
- What are the usage patterns for these operations?

Common applications that might use your directory include the following:

- **Browser applications, such as white pages.** Applications of this type generally access information such as email addresses, telephone numbers, and employee names.
- **Messaging applications, especially email servers.** All email servers require email addresses, user names, and routing information. Others require more advanced information such as the place on disk where a user's mailbox is stored, vacation notification information, and protocol information.
- **Directory-enabled human resources applications.** These applications require more personal information such as government identification numbers, home addresses, home telephone numbers, and salary details.
- **Security, web portal, or personalization applications.** Applications of this type access profile information.

When you have identified the information used by each application, you might see that some types of data are used by more than one application. Performing this kind of exercise during the planning stage can help you to avoid data redundancy.

## Determining the Number and Size of Directory Entries

The number and size of entries that are stored in the directory depend largely on your data requirements, as described in Chapter 4, "Defining Data Characteristics."

Consider the following when calculating the number and size of entries:

- Does the deployment require repeated bulk import initialization?
- If so, how often are imports performed?
- How many entries are imported at a time?
- What types of entries are imported?
- Must initialization be performed online with the server running?

## Determining the Number of Reads

In estimating read traffic, consider the following:

- How many searches per second are expected?
- What types of searches are expected?

  For example, unique ID searches, wildcard searches, exact match searches.
- What is the estimated peak search rate?
- What is the estimated average search rate?
- How many unindexed searches are expected?

  An unindexed search means that the database is searched directly, instead of the index file. Unindexed searches occur either when the All IDs Threshold is reached within the index file used for the search, when no index file exists or when the index file is not configured in the way required by the search.

  Unindexed searches are generally more time consuming than indexed searches.
- Are searches concentrated in a particular data center or geographic region?

  If one data receives proportionally more search traffic than other data centers, it might be worth placing additional, replicated servers in this data center to balance the load.
- Are searches concentrated at a particular time of day?
- How many searches are anticipated from within the firewall?
- How many searches are anticipated from outside the firewall?

If read performance is crucial to your enterprise, see Chapter 10, "Designing a Scaled Deployment," for suggestions on deploying a directory service that is scaled for reads.

## Determining the Number of Writes

In estimating write traffic, consider the following:

- How many updates per second are expected?
- What types of updates are expected?
- What is the estimated peak update rate?
- What is the estimated average update rate?
- Are updates concentrated in a particular data center or geographic region?

  If one data receives proportionally more update traffic than other data centers, it might be worth placing additional writable servers in this data center to distribute the update load.
- Are updates concentrated at a particular time of day?

If write performance is crucial to your enterprise, see Chapter 10, "Designing a Scaled Deployment," for suggestions on deploying a directory service that is scaled for writes.

## Estimating the Acceptable Response Time

For each client application, determine the maximum response time that is acceptable. The acceptable response time might differ for various geographical locations, and for different kinds of operations.

## Estimating the Acceptable Replication Latency

Estimate the level of synchronicity that is required between master replicas and consumer replicas. The Directory Server replication model is loosely consistent, that is, updates are accepted on a master without requiring communication with the other replicas in a topology. At any given time, the contents of each replica might be different. Over time, the replicas converge until each replica has an identical copy of the data. As part of performance planning, determine the maximum acceptable time that replicas have to converge.

Directory Server 6.x includes a new *prioritized replication* feature. This feature enables you to specify that changes to certain attributes must be replicated as soon as possible. Prioritized replication might affect your decisions about acceptable replication latency. For more information, see "Prioritized Replication" in *Sun Java System Directory Server Enterprise Edition 6.3 Reference*.

# Defining Availability Requirements

Availability implies an agreed minimum *up time* and level of performance for your directory service. Failure, in this context, is defined as anything that prevents the directory service from providing this minimum level of service.

In assessing availability requirements, consider the following:

- Is your directory service accessed only at particular times of the day?
- Do you have different availability requirements for read and write operations?
- Does the service span multiple geographical sites, and if so, do these sites have different access time requirements?
- Can your system be shut down for maintenance?

  If so, what is the maximum acceptable downtime?
- Can the system be shut down during migration?
- What is the cost of downtime to your organization?

For suggestions on deploying a highly available directory service, see Chapter 12, "Designing a Highly Available Deployment."

# Defining Scalability Requirements

As your directory evolves, the service levels that must be supported might change. To raise the level of service after a system has been deployed can be difficult. Thus, the initial design must take future requirements into account.

When defining scalability requirements, consider the following:

- Is there an anticipated increase in entry volume?
- How many new users are expected within the next few years?
- What is the expected growth rate, over the next few years, in terms of data, users, and client applications?
- Are any new business processes expected?

Increase CPU estimates to make sure that your deployment does not have to be scaled prematurely. Look at the anticipated milestones for scaling and projected load increase over time to make sure that you allow enough latent capacity to reach the milestones.

# Defining Security Requirements

Security requirements warrant separate discussion. These requirements are described in detail in Chapter 7, "Identifying Security Requirements."

# Defining Latent Capacity Requirements

In determining latent capacity requirements, estimate the peak load conditions for your directory service. Consider the following:

- If all client applications were running, what would be the maximum number of concurrent connections to Directory Server?
- What would be the load on the remaining servers in your deployment if one or more servers were to fail?

# Defining Serviceability Requirements

Serviceability requirements are discussed in detail in Chapter 8, "Identifying Administration and Monitoring Requirements."

◆ ◆ ◆ CHAPTER 6

# Tuning System Characteristics and Hardware Sizing

A Directory Server Enterprise Edition deployment requires that certain system characteristics be defined at the outset. This chapter describes the system information that you need to address in the planning phase of your deployment.

This chapter covers the following topics:



## Host System Characteristics

When identifying the host systems that will be used in your deployment, consider the following:

- Will the system be dedicated to a single server?
- Will the system be running other applications, and if so, what will the other applications be?
- What percentage of the system's resources will these applications require?

When the host systems have been identified, select a host name for each host in the topology. Make sure that each host system has a static IP address.

Restrict physical access to the host system. Although Directory Server Enterprise Edition includes many security features, directory security is compromised if physical access to the host system is not controlled.

If the Directory Server instances do not provide a naming service for the network, or if the deployment involves remote administration, a naming service and the domain name for the host must be properly configured.

# Port Numbers

At design time, select port numbers for each Directory Server and Directory Proxy Server instance. If possible, do not change port numbers after your directory service is deployed in a production environment.

Separate port numbers must be allocated for various services and components.



## Directory Server and Directory Proxy Server LDAP and LDAPS Port Numbers

Specify the port number for accepting LDAP connections. The standard port for LDAP communication is 389, although other ports can be used. For example, if you must be able to start the server as a regular user, use an unprivileged port, by default 1389. Port numbers less than 1024 require privileged access. If you use a port number that is less than 1024, certain LDAP commands must be run as `root`.

Specify the port number for accepting SSL-based connections. The standard port for SSL-based LDAP (LDAPS) communication is 636, although other ports can be used, such as the default 1636 when running as a regular user. For example, an unprivileged port might be required so that the server can be started as a regular user.

If you specify a non-privileged port and a server instance is installed on a system to which other users have access, you might expose the port to a hijack risk by another application. In other words, another application can bind to the same address/port pair. The rogue application might then be able to process requests that are intended for the server. The application could also be used to capture passwords used in the authentication process, to alter client requests or server responses, or to produce a denial of service attack.

Both Directory Server and Directory Proxy Server allow you to restrict the list of IP addresses on which the server listens. Directory Server has configuration attributes `nsslapd-listenhost` and `nsslapd-securelistenhost`. Directory Proxy Server has `listen-address` properties on `ldap-listener` and `ldaps-listener` configuration objects. When you specify the list of interfaces on which to listen, other programs are prevented from using the same port numbers as your server.

## Directory Server DSML Port Numbers

In addition to processing requests in LDAP, Directory Server also responds to requests sent in the Directory Service Markup Language v2 (DSML). DSML is another way for a client to encode directory operations. Directory Server processes DSML as any other request, with the same access control and security features.

If your topology includes DSML access, identify the following:

- A standard HTTP port for receiving DSML requests. The default port is 80.
- If SSL is activated, an encrypted (HTTPS) port for receiving encrypted DSML requests. The default port is 443.
- A relative URL that, when appended to the host and port, determines the complete URL that clients must use to send DSML requests

For information about configuring DSML, see "To Enable the DSML-over-HTTP Service" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

## Directory Service Control Center and Common Agent Container Port Numbers

Directory Service Control Center, DSCC, is a web application for Sun Java Web Console that enables you to administer Directory Server and Directory Proxy Server instances through a web browser. For a server to be recognized by DSCC, the server must be registered with DSCC. Unregistered servers can still be managed using command-line utilities.

DSCC communicates with DSCC agents located on the systems where servers are installed. The DSCC agents run inside a common agent container, which routes network traffic to them and provides them a framework in which to run.

If you plan to use DSCC to administer servers in your topology, identify the following port numbers.

- The encrypted HTTPS port for accessing DSCC through Sun Java Web Console on the system where DSCC is installed. The default port is 6789.
- The management traffic port for DSCC to access its agents local to the server through the common agent container, default: 11162, on the system where the server instances are installed.
- The port numbers for the DSCC Registry instance, if you plan to replicate the configuration information. See `dsccsetup`(1M) for details.

Even if all components are installed on the same system, DSCC still communicates with its agents through these network ports.

## Identity Synchronization for Windows Port Numbers

If your deployment includes identity synchronization with Microsoft Active Directory, an available port is required for the Message Queue instance. This port must be available on each Directory Server instance that participates in the synchronization. The default non-secure port for Message Queue is 80, and the default secure port is 443.

You must also make additional installation decisions and configuration decisions when planning your deployment. For details on installing and configuring Identity Synchronization for Windows, see Part II, "Installing Identity Synchronization for Windows," in *Sun Java System Directory Server Enterprise Edition 6.3 Installation Guide*.

# Hardware Sizing For Directory Service Control Center

DSCC runs as a web application inside Sun Java Web Console, which runs inside a web application container. DSCC also runs its own local instance of Directory Server to store configuration data.

The minimum requirement to run DSCC is 256 megabytes of memory and 100 megabytes of free disk space. However, for optimum performance run DSCC on a system with at least one gigabyte of memory devoted to DSCC and a couple gigabytes of free disk space.

# Hardware Sizing For Directory Proxy Server

Directory Proxy Server runs as a multithreaded Java program, and is built to scale across multiple processors. In general, the more processing power available the better, though you might find that in practice adding memory, faster disks, or faster network connections can enhance performance more than additional processors.

## Configuring Virtual Memory

Directory Proxy Server uses memory mainly to hold information that is being processed. Complex aggregations for processing some virtual directory requests against multiple data sources may temporarily use extra memory. If one of your data sources is an LDIF file, Directory Proxy Server constructs a representation of that data source in memory. However, unless you use large LDIF data sources, not a recommended deployment practice, a couple gigabytes of memory devoted to Directory Proxy Server should suffice. You might want to increase the Java virtual machine heap size when starting Directory Proxy Server if enough memory is available. For example, to set the Java virtual machine heap size to 1000 megabytes, use the following command.

```
$ dpadm set-flags instance-path jvm-args="-Xmx1000M -Xms1000M -XX:NewRatio=1"
```

This command uses the `-XX:NewRatio` option, which is specific to the Sun Java virtual machine. The default heap size is 250 megabytes.

## Configuring Worker Threads and Backend Connections

Directory Proxy Server allows you to configure how many threads the server maintains to process requests. You configure this using the server property `number-of-worker-threads`, described in `number-of-worker-threads`(5dpconf). As a rule of thumb, try setting this number to 50 threads plus 20 threads for each data source used. To gauge whether the number is sufficient, monitor the status of the Directory Proxy Server work queue on `cn=Work Queue,cn=System Resource,cn=`*instance-path*`,cn=Application System,cn=DPS6.0,cn=Installed Product,cn=monitor`. If you find that the `operationalStatus` for the work queue is `STRESSED`, this can mean thread-starved connection handlers are unable to handle new client requests. Increasing `number-of-worker-threads` may help if more system resources are available for Directory Proxy Server.

The number of worker threads should also be appropriate for the number of backend connections. If there are *too many worker threads* for the number of backend connections, incoming connections are accepted but cannot be transmitted to the backend connections. Such a situation is generally problematic for client applications.

To determine whether this situation has arisen, check the log files for error messages of the following type: "`Unable to get backend connections`". Alternatively, look at the `cn=monitor` entry for load balancing. If the `totalBindConnectionsRefused` attribute in that entry is not null, the proxy was unable to process certain operations because there were not enough backend connections. To solve this issue, increase the maximum number of backend connections. You can configure the number of backend connections for each data source by using the `num-bind-limit`, `num-read-limit` and `num-write-limit` properties of the data source. If you have already reached the limit for backend connections, reduce the number of worker threads.

If there are *not enough worker threads* for the number of backend connections, so much work can pile up in the server's queue that no new connections can be handled. Client connections can then be refused at the TCP/IP level, with no LDAP error returned. To determine if this situation has arisen, look at the statistics in the `cn=monitor` entry for the work queue. In particular, `readConnectionsRefused` and `writeConnectionsRefused` should remain low. Also, the value of the `maxNormalPriorityPeak` attribute should remain low.

## Disk Space for Directory Proxy Server

By default Directory Proxy Server requires up to one gigabyte of local disk space for `access` logging, and another gigabyte of local disk space for `errors` logging. Given the quantity of `access` log messages Directory Proxy Server writes when handling client application requests,

logging can be a performance bottleneck. Typically, however, you must leave logging on in a production environment. For best performance therefore put Directory Proxy Server logs on a fast, dedicated disk subsystem. See "Configuring Directory Proxy Server Logs" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide* for instructions on adjusting log settings.

## Network Connections for Directory Proxy Server

Directory Proxy Server is a network-intensive application. For each client application request, Directory Proxy Server may send multiple operations to different data sources. Make sure the network connections between Directory Proxy Server and your data sources are fast, with plenty of bandwidth and low latency. Also make sure the connections between Directory Proxy Server and client applications can handle the amount of traffic you expect.

# Hardware Sizing For Directory Server

Getting the right hardware for a medium to large Directory Server deployment involves some testing with data similar to the data you expect to serve in production, and access patterns similar to those you expect from client applications. When optimizing for particular systems, make sure you understand how system buses, peripheral buses, I/O devices, and supported file systems work. This knowledge helps you take advantage of I/O subsystem features when tuning these features to support Directory Server. Sun Services (`http://www.sun.com/servicessolutions/`) can help you make the right deployment decisions, including sizing the hardware to your requirements.

This section looks at how to approach hardware sizing for Directory Server. It covers what to consider when deciding how many processors, how much memory, how much disk space, and what type of network connections to dedicate to Directory Server in your deployment.

This section covers the following topics:

**Note –** Unless indicated otherwise, the *server properties* described in the following sections can be set with the `dsconf` command. For more information about using `dsconf`, see `dsconf`(1M).

# The Tuning Process

To tune performance implies modification of the default configuration to reflect specific deployment requirements. The following list of process phases covers the key things to think about when tuning Directory Server.

Define goals
: Define specific, measurable objectives for tuning, based on deployment requirements.

  Consider the following questions.

  - Which applications use Directory Server?
  - Can you dedicate the entire system to Directory Server?

    Does the system run other applications?

    If so, which other applications run on the system?
  - How many *entries* are handled by the deployment?

    How large are the entries?
  - How many *searches* per second must Directory Server support?

    What types of searches are expected?
  - How many *updates* per second must Directory Server support?

    What types of updates are expected?
  - What sort of peak update and search rates are expected?

    What average rates are expected?
  - Does the deployment call for repeated bulk import initialization on this system?

    If so, how often do you expect to import data? How many entries are imported?

    What types of entries?

    Must initialization be performed online with the server running?

  The list here is not exhaustive. Ensure that your list of goals is exhaustive.

Select methods
: Determine how you plan to implement optimizations. Also, determine how you plan to measure and analyze optimizations.

Consider the following questions.

- Can you change the hardware configuration of the system?
- Are you limited to using hardware that you already have, tuning only the underlying operating system, and Directory Server?
- How can you simulate other applications?
- How should you generate representative data samples for testing?
- How should you measure results?
- How should you analyze results?

Perform tests
: Carry out the tests that you planned. For large, complex deployments, this phase can take considerable time.

Verify results
: Check whether the potential optimizations tested reach the goals defined at the outset of the process.

  If the optimizations reach the goals, document the results.

  If the optimizations do not reach the goals, profile and monitor Directory Server.

Profile and monitor
: Profile and monitor the behavior of Directory Server after applying the potential modifications.

  Collect measurements of all relative behavior.

Plot and analyze
: Plot and analyze the behavior that you observed while profiling and monitoring. Attempt to find evidence and to discover patterns that suggest further tests.

  You might need to go back to the profiling and monitoring phase to collect more data.

Tweak and tune
: Apply further potential optimizations suggested by your analysis of measurements.

  Return to the phase of performing tests.

Document results
: When the optimizations applied reach the goals defined at the outset of the process, document the optimizations well so the optimizations can be easily reproduced.

# Making Sample Directory Data

How much disk and memory space you devote to Directory Server depends on your directory data. If you already have representative data in LDIF, use that data when sizing hardware for your deployment. Representative data here means sample data that corresponds to the data you

expect to use in deployment, but *not actual data you use in deployment*. Real data comes with real privacy concerns, can be multiple orders of magnitude larger than the specifications need to generate representative data, and may not help you exercise all the cases you want to test. Representative data includes entries whose average size is close to the size you expect to see in deployment, whose attributes have values similar to those you expect to see in deployment, and whose numbers are present in proportions similar to those you expect to see in deployment.

Take anticipated growth into account when you are deciding on representative data. It is advisable to include an overhead on current data for capacity planning.

If you do not have representative data readily available, you can use the `makeldif(1)` command to generate sample LDIF, which you can then import into Directory Server. Chapter 4, "Defining Data Characteristics," can help you figure out what representative data would be for your deployment. The `makeldif` command is one of the Directory Server Resource Kit tools.

For deployments expected to serve millions of entries in production, ideally you would load millions of entries for testing. Yet loading millions of entries may not be practical for a first estimate. Start by creating a few sets of representative data, for example 10,000 entries, 100,000 entries, and 1,000,000 entries, import those, and extrapolate from the results you observe to estimate the hardware required for further testing. When you are estimating hardware requirements, make provision for data that will be replicated to multiple servers.

Notice when you import directory data from LDIF into Directory Server the resulting database files (including indexes) are larger than the LDIF representation. The database files, by default, are located under the *instance-path*`/db/` directory.

# What to Configure and Why

Directory Server default configuration settings are defined for typical small deployments and to make it easy to install and evaluate the product. This section examines some key configuration settings to adjust for medium to large deployments. In medium to large deployments you can often improve performance significantly by adapting configuration settings to your particular deployment.

## Directory Server Database Page Size

When Directory Server reads or writes data, it works with fixed blocks of data, called pages. By increasing the page size you increase the size of the block that is read or written in one disk operation.

The page size is related to the size of entries and is a critical element of performance. If you know that the average size of your entries is greater than `db-page-size/4–24` (`24` is the per page binary tree internal structure), you must increase the database page size. The database page size should also match the file system disk block size.

## Directory Server Cache Sizes

Directory Server is designed to respond quickly to client application requests. In order to avoid waiting for directory data to be read from disk, Directory Server caches data in memory. You can configure how much memory is devoted to cache for database files, for directory entries, and for importing directory data from LDIF.

Ideally the hardware on which you run Directory Server allows you to devote enough space to cache all directory data in physical memory. The data should fit comfortably, such that the system has enough physical memory for operation, and the file system has plenty of physical memory for its caching and operation. Once the data are cached, Directory Server has to read data from and write data to disk only when a directory entry changes.

Directory Server supports 64–bit memory addressing, and so can handle total cache sizes as large as a 64–bit processor can address. For small to medium deployments it is often possible to provide enough memory that all directory data can be held in cache. For large deployments, however, caching everything may not be practical or cost effective.

For large deployments, caching everything in memory can cause side effects. Tools such as the `pmap` command, that traverse the process memory map to gather data, can freeze the server process for a noticeable time. Core files can become so large that writing them to disk during a crash can take several minutes. Startup times can be slow if the server is shut down abruptly and then restarted. Directory Server can also pause and stop responding temporarily when it reaches a checkpoint and has to flush dirty cached pages to disk. When the cache is very large, the pauses can become so long that monitoring software assumes Directory Server is down.

I/O buffers at the operating system level can provide better performance. Very large buffers can compensate for smaller database caches.

For a detailed discussion of cache and cache settings, read Chapter 5, "Directory Server Data Caching," in *Sun Java System Directory Server Enterprise Edition 6.3 Reference*. For more information on tuning cache sizes, read The Basics of Directory Server Cache Sizing (`http://blogs.sun.com/DirectoryManager/resource/ds_cache_sizing.pdf`).

## Directory Server Indexes

Directory Server indexes directory entry attribute values to speed searches for those values. You can configure attributes to be indexed in various ways. For example, indexes can help Directory Server determine quickly whether an attribute has a value, whether it has a value equal to a given value, and whether it has a value containing a given substring.

Indexes can add to search performance, but they can also impact write performance. When an attribute is indexed, Directory Server has to update the index as values of the attribute change.

Directory Server saves index data to files. The more indexes you configure, the more disk space required. Directory Server indexes and data files are found, by default, under the *instance-path*`/db/` directory.

For a detailed discussion of indexing and index settings, read Chapter 6, "Directory Server Indexing," in *Sun Java System Directory Server Enterprise Edition 6.3 Reference*.

## Directory Server Administration Files

Some Directory Server administration files can potentially become very large. These files include the LDIF files containing directory data, backups, core files, and log files.

Depending on your deployment, you may use LDIF both to import Directory Server data, and to serve as auxiliary backup. A standard text format, LDIF allows you to export binary data as well as strings. LDIF can occupy significant disk space in large deployments. For example, a directory containing 10 million entries having an average size of 2 kilobytes, would in LDIF representation occupy 20 gigabytes on disk. You might maintain multiple LDIF files of that size if you use the format for auxiliary backup.

Binary backup files also occupy space on disk, at least until you move them somewhere else for safekeeping. Backup files produced with Directory Server utilities consist of binary copies of the directory database files. Alternatively for large deployments you can put Directory Server in frozen mode and take a snapshot of the file system. Either way, you must have disk space available for the backup.

By default Directory Server writes log messages to *instance-path*`/logs/access` and *instance-path*`/logs/errors`. By default Directory Server requires one gigabyte of local disk space for `access` logging, and another 200 megabytes of local disk space for `errors` logging.

For a detailed discussion of Directory Server logging, read Chapter 7, "Directory Server Logging," in *Sun Java System Directory Server Enterprise Edition 6.3 Reference*.

## Directory Server Replication

Directory Server lets you replicate directory data for availability and load balancing between the servers in your deployment. Directory Server allows you to have multiple read-write (master) replicas deployed together.

Internally, the server makes this possible by keeping track of changes to directory data. When the same data are modified on more than one read-write replica Directory Server can resolve the changes correctly on all replicas. The data to track these changes, must be retained until they are no longer needed for replication. Changes are retained for a period of time specified by the *purge delay* whose default value is seven days. If your directory data undergoes much modification, especially of large multi-valued attributes, this data can grow quite large.

Because the level of growth is dependent on several factors, there is no catch-all formula to calculate potential data growth. The best approach is to test typical modifications and measure the growth. The following factors have an effect on data growth as a result of entry modification:

- The type of entries and the types of attributes that are modified.

  Multi-valued attributes cause larger growth. If the attribute values are small, the growth is more visible.

- The workload applied to the entry.

  Adding and deleting entries causes larger growth. Adding an attribute value causes larger growth than replacing an attribute value.

- The number of entries that are modified, and the number of attributes that are modified in each entry.

- The size of the database page.

  After numerous modifications, certain entries can become larger than the database page size.

Note that the replication meta-data remains in the entry until the purge delay has passed *and* the entry is modified again.

For a detailed discussion of Directory Server replication, read Chapter 4, "Directory Server Replication," in *Sun Java System Directory Server Enterprise Edition 6.3 Reference*.

## Directory Server Threads and File Descriptors

Directory Server runs as a multithreaded process, and is designed to scale on multiprocessor systems. You can configure the number of threads Directory Server creates at startup to process operations. By default Directory Server creates 30 threads. The value is set using the `dsconf`(1M) command to adjust the server property `thread-count`.

The trick is to keep the threads as busy as possible without incurring undo overhead from having to handle many threads. As long as all directory data fits in cache, better performance is often seen when `thread-count` is set to twice the number of processors plus the expected number of simultaneous update operations. If only a fraction of a large directory data set fits in cache, Directory Server threads may often have to wait for data being read from disk. In that case you may find performance improves with a much higher thread count, up to 16 times the number of available processors.

Directory Server uses file descriptors to hold data related to open client application connections. By default Directory Server uses a maximum of 1024 file descriptors. The value is set using the `dsconf` command to adjust the server property `file-descriptor-count`. If you see a message in the `errors` log stating `too many fds open`, you may observe better performance by increasing `file-descriptor-count`, presuming your system allows Directory Server to open additional file descriptors.

The `file-descriptor-count` property does not apply on Windows.

## Directory Server Growth

Once in deployment Directory Server use is likely to grow. Planning for growth is key for a successful deployment, in which you continue to provide a consistently high level of service. Plan for larger, more powerful systems than you need today, basing your requirements in part on the growth you expect tomorrow.

Sometimes directory services must grow rapidly, even suddenly. This is the case for example when a directory service sized for one organization is merged with that of another organization. By preparing for growth in advance and by explicitly identifying your expectations, you are better equipped to deal with rapid and sudden growth, because you know in advance whether the expected increase outstrips the capacity you planned.

## Top Tuning Tips

Basic recommendations follow. These recommendations apply in most situations. Although the recommendations presented here are in general valid, avoid the temptation to apply the recommendations without understanding the impact on the deployment at hand. This section is intended as a checklist, not a cheat sheet.

1. Adjust cache sizes.

   Ideally, the server has enough available physical memory to hold all caches used by Directory Server. Furthermore, an appropriate amount of extra physical memory is available to account for future growth. When plenty of physical memory is available, set the entry cache size large enough to hold all entries in the directory. Use the `entry-cache-size` suffix property. Set the database cache size large enough to hold all indexes with the `db-cache-size` property. Use the `dn-cache-size` or `dn-cache-count` properties to control the size of the DN cache.

2. Optimize indexing.

   a. Remove unnecessary indexes. Add additional indexes to support expected requests.

      From time to time, you can add additional indexes that support requests from new applications. You can add, remove, or modify indexes while Directory Server is running. Use for example the `dsconf create-index` and `dsconf delete-index` commands.

      Be careful not to remove system indexes. For a list of system indexes, see "System Indexes and Default Indexes" in *Sun Java System Directory Server Enterprise Edition 6.3 Reference*.

      Directory Server gradually indexes data after you make changes to the indexes. You can also force Directory Server to rebuild indexes with the `dsconf reindex` command.

   b. Allow only indexed searches.

      Unindexed searches can have a strong negative impact on server performance. Unindexed searches can also consume significant server resources.

      Consider forcing the server to reject unindexed searches by setting the `require-index-enabled` suffix property to `on`.

c. Adjust the maximum number of values per index key with the `all-ids-threshold` property.

3. Tune the underlying operating system according to recommendations made by the `idsktune` command. For more information, see `idsktune`(1M).
4. Adjust operational limits.

   Adjustable operational limits prevent Directory Server from devoting inordinate resources to any single operation. Consider assigning unique bind DNs to client applications requiring increased capabilities, then setting resource limits specifically for these unique bind DNs.
5. Distribute disk activity.

   Especially for deployments that support large numbers of updates, Directory Server can be extremely disk I/O intensive. If possible, consider spreading the load across multiple disks with separate controllers.
6. Disable unnecessary logging.

   Disk access is slower than memory access. Heavy logging can therefore have a negative impact on performance. Reduce disk load by leaving audit logging off when not required, such as on a read-only server instance. Leave error logging at a minimal level when not using the error log to troubleshoot problems. You can also reduce the impact of logging by putting log files on a dedicated disk, or on a lesser used disk, such as the disk used for the replication changelog.
7. When replicating large numbers of updates, consider adjusting the appropriate replication agreement properties.

   The properties are `transport-compression`, `transport-group-size`, and `transport-window-size`.
8. On Solaris systems, move the database home directory to a `tmpfs` file system.

   The database home directory, specified by the `db-env-path` property, indicates where Directory Server locates database cache backing files. Data files continue to reside by default under *instance-path*`/db`.

   With the database cache backing files on a `tmpfs` file system, the system does not repeatedly flush the database cache backing files to disk. You therefore avoid a performance bottleneck for updates. In some cases, you also avoid the performance bottleneck for searches. The database cache memory is mapped to the Directory Server process space. The system essentially shares cache memory and memory used to hold the backing files in the `tmpfs` file system. You therefore gain performance at essentially no cost in terms of memory space needed.

   The primary cost associated with this optimization is that database cache must be rebuilt after a restart of the host machine. This cost is probably not a cost that you can avoid, however, if you expect a restart to happen only after a software or hardware failure. After such a failure, the database cache must be rebuilt anyway.

9. Enable transaction batches if you can afford to lose updates during a software or hardware failure.

   You enable transaction batches by setting the server property `db-batched-transaction-count`.

   Each update to the transaction log is followed by a sync operation to ensure that update data is not lost. By enabling transaction batches, updates are grouped together before being written to the transaction log. Sync operations only take place when the whole batch is written to the transaction log. Transaction batches can therefore significantly increase update performance. The improvement comes with a trade off. The trade off is during a crash, you lose update data not yet written to the transaction log.

   ---

   **Note –** With transaction batches enabled, you lose up to `db-batched-transaction-count - 1` updates during a software or hardware failure. The loss happens because Directory Server waits for the batch to fill, or for 1 second, whichever is sooner, before flushing content to the transaction log and thus to disk.

   Do *not* use this optimization if you cannot afford to lose updates.

   ---

10. Configure the referential integrity plug-in to delay integrity checks.

    The referential integrity plug-in ensures that when entries are modified, or deleted from the directory, all references to those entries are updated. By default, the processing is performed synchronously, before the response for the delete operation is returned to the client. You can configure the plug-in to have the updates performed asynchronously. Use the `ref-integrity-check-delay` server property.

# Simulating Client Application Load

To measure Directory Server performance, you prepare the server, then subject it to the kind of client application traffic you expect in production. The better you reproduce the kind of access patterns client applications that happen in production, the better job you can do sizing the hardware and configuring Directory Server appropriately.

Directory Server Resource Kit provides the `authrate`(1), `modrate`(1), and `searchrate`(1) commands you can use for basic tests. These commands let you measure the rate of binds, modifications, and searches your directory service can support.

You can also simulate, measure, and graph complex, realistic client access using SLAMD. The SLAMD Distributed Load Generation Engine (SLAMD) is a Java application that is designed to stress test and analyze the performance of network-based applications. It was originally developed by Sun Microsystems, Inc. to benchmark and analyze the performance of LDAP Directory Servers. SLAMD is available as an open source application under the Sun Public

License, an OSI-approved open source license. To obtain information about SLAMD, go to `http://www.slamd.com/`. SLAMD is also available as a java.net project. See `https://slamd.dev.java.net/`.

## Directory Server and Processors

As a multithreaded process built to work on systems with multiple processors, Directory Server performance scales linearly in most cases as you devote more processors to it. When running Directory Server on a system with many processors, consider using the `dsconf` command to adjust the server property `thread-count`, which is the number of threads Directory Server starts to process server operations.

In specific directory deployments, however, adding more processors might not significantly impact performance. When handling demanding performance requirements for searching, indexing, and replication, consider load balancing and directory proxy technologies as part of the solution.

## Directory Server and Memory

The following factors significantly affect the amount of memory needed:

- Directory Server database cache, entry cache, and import cache settings
- Peak replication load
- Peak client application load
- Server settings for `file-descriptor-count` and `thread-count`
- Overhead for the operating system, other applications running on the system, and system administration activity

To estimate the memory size required to run Directory Server, estimate the memory needed for a specific Directory Server configuration on a system loaded as in production, including application load generated for example using the Directory Server Resource Kit commands or SLAMD.

Before you measure Directory Server process size, give the server some time after startup to fill entry caches as during normal or peak operation. If you have space to put everything in cache memory, you can speed this warm up period for Directory Server by reading every entry in the directory to fill entry caches. If you do not have space to put everything in cache memory, simulate client access for some time until the cache fills as it would with a pattern of normal or peak operation.

With the server in an equilibrium state, you can use utilities such as `pmap` on Solaris or Linux, or the Windows Task Manager to measure memory used by the Directory Server process,

`ns-slapd` on UNIX systems, `slapd.exe` on Windows systems. For more information, see the `pmap(1)` man page. Measure process size both during normal operation and peak operation before deciding how much memory to use.

Make sure to add to your estimates the amount of memory needed for system administration, and for the system itself. Operating system memory requirements can vary widely depending on the system configuration. Therefore, estimating the memory needed to run the underlying operating system must be done empirically. After tuning the system, monitor memory use to your estimate. You can use utilities such as the Solaris `vmstat` and `sar` commands, or the Task Manager on Windows to measure memory use.

At a minimum, provide enough memory so that running Directory Server does not cause constant page swapping, which negatively affects performance. Utilities such as `MemTool`, unsupported and available separately for Solaris systems, can be useful in monitoring how memory is used by and allocated to running applications.

If the system cannot accommodate additional memory, yet you continue to observe constant page swapping, reduce the size of the database and entry caches. Although you can throttle memory use with the `heap-high-threshold-size` and `heap-low-threshold-size` server settings, consider the heap threshold mechanism as a last resort. Performance suffers when Directory Server must delay other operations to free heap memory.

On Red Hat Linux systems, you can adjust the `/proc/sys/vm/swappiness` parameter to tune how aggressively the kernel swaps out memory. High swappiness means that the kernel will swap out a large amount and low swappiness means that the kernel will try not to use swap space at all. Decreasing the swappiness setting may therefore result in improved Directory performance as the kernel holds more of the server process in memory longer before swapping it out. If the system is dedicated to a single Directory Server instance, set the swappiness to zero. If the system runs several heavy processes or multiple concurrent instances of Directory Server, consider testing the Directory performance with various swappiness settings.

## Directory Server and Local Disk Space

Disk use and I/O capabilities can have great impact on performance. The disk subsystem can become an I/O bottleneck, especially for a deployment that supports large numbers of modifications. This section recommends ways to estimate overall disk capacity for a Directory Server instance.

**Note –** Do *not* install Directory Server or any data it accesses on network disks.

Directory Server software does not support the use of network-attached storage through NFS, AFS, or SMB. All configuration, database, and index files must reside on local storage at all times, even after installation. Log files can be stored on network disks.

The following factors significantly affect the amount of local disk space needed:

- Number of directory entries
- Average sizes of entries
- Server database page size setting when directory data is imported

  To adjust the database page size, set the `nsslapd-db-page-size` attribute. For more information, see "Directory Server Database Page Size" on page 77.
- Number of indexes maintained on directory data
- Size of stored LDIF, backups, logs, and core files

When you have set up indexes, adjusted the database page size, and imported directory data, you can estimate the disk capacity required for the instance by reading the size of the *instance-path*`/` contents, and adding the size of expected LDIF, backups, logs, and core files. Also estimate how much the sizes you measure are expected to grow, particularly during peak operation. Make sure you leave a couple of gigabytes of extra space for the `errors` log in case you need to increase the log level and size for debugging purposes.

Getting an estimation of the disk required for directory data can be done in some cases by extrapolation. If it is not practical to load Directory Server with as much data as you expect in production, extrapolate from smaller sets of sample data as suggested in "Making Sample Directory Data" on page 76. When the amount of directory data you use is smaller than in production, you must extrapolate for other measurements, too.

The following factors determine how fast the local disk must be:

- Level of updates sustained, including the volume of replication traffic
- Whether directory data are mainly in cache or on disk
- Log levels used for access and error logging, and whether the audit log is enabled
- Whether directory data, logs, and the transaction log (for updates) can be placed on separate disk subsystems
- Whether backups are performed with Directory Server online or offline

Disks used should not be saturated under normal operating circumstances. You can use tools such as the Solaris `iostat` command to isolate potential I/O bottlenecks.

To increase disk throughput distribute files across disk subsystems. Consider providing dedicated disk subsystems for transaction logs (`dsconf set-server-prop db-log-path:`*/transaction/log/path*), databases (`dsconf create-suffix --db-path` */suffix/database/path suffix-name*), and log files (`dsconf set-log-prop path:`*/log/file/path*). In addition consider putting database cache files on a memory-based file system such as a Solaris `tmpfs` file system, where files are swapped to disk only if available memory is exhausted (for example, `dsconf set-server-prop db-env-path:/tmp`). If you put database cache files on a memory-based file system, make sure the system does not run out of space to keep that entire file system in memory.

To further increase throughput use multiple disks in RAID configuration. Large, non volatile I/O buffers and high-performance disk subsystems such as those offered in Sun StorEdge™ products can greatly enhance Directory Server performance and uptime. On Solaris 10 systems, using ZFS can also improve performance.

# Directory Server and Network Connectivity

Directory Server is a network-intensive application. You can estimate theoretical maximum throughput using the following formula. Notice that this formula does not account for replication traffic.

```
max. throughput = max. entries returned/second x average entry size
```

Imagine that a Directory Server must respond to a peak of 5000 searches per second and that the server returns one entry per search. The entries have an average size of 2000 bytes. The theoretical maximum throughput would be 10 megabytes, or 80 megabits, not counting replication. 80 megabits are likely to be more than a single 100-megabit Ethernet adapter can provide. To improve network availability for a Directory Server instance, equip the system with a faster connection, or with multiple network interfaces. Directory Server can listen on multiple network interfaces within the same process.

---

**Note –** The preceding example assumes that the client application requests *all* attributes when reading or searching the directory. Generally, you should design client applications so that they request only the *required* attributes.

---

If you intend to cluster Directory Servers on the same network for load balancing purposes, make sure the network infrastructure can support the additional load generated for replication. If you plan multi-master replication over a wide area network, test your configuration to make sure the connection provides sufficient throughput with minimum latency and near-zero packet loss. High latency and packet loss both slow replication. In addition, avoid a topology where replication traffic goes through a load balancer.

# Limiting Directory Server Resources Available to Clients

The default configuration of Directory Server can allow client applications to use more Directory Server resources than are required.

The following uses of resources can hurt directory performance:

- Opening many connections then leaving them idle or unused
- Launching costly and unnecessary unindexed searches
- Storing enormous and unplanned for binary attribute values

In some deployment situations, you should not modify the default configuration. For deployments where you cannot tune Directory Server, use Directory Proxy Server to limit resources, and to protect against denial of service attacks.

In some deployment situations, one instance of Directory Server must support client applications, such as messaging servers, and directory clients such as user mail applications. In such situations, consider using bind DN based resource limits to raise individual limits for directory intensive applications. The limits for an individual account can be adjusted by setting the attributes `nsSizeLimit`, `nsTimeLimit`, `nsLookThroughLimit`, and `nsIdleTimeout` on the individual entry. For information about how to control resource limits for individual accounts, see "Setting Resource Limits For Each Client Account" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

Table 6–1 describes the parameters that set the global values for resource limits. The limits in Table 6–1 do not apply to the Directory Manager user, therefore, ensure client applications do not connect as the Directory Manager user.

**TABLE 6–1** Tuning Recommendations For Resources Devoted to Client Applications

| Tuning Parameter | Description |
|---|---|
| Server property<br>`idle-timeout` | Sets the time in seconds after which Directory Server closes an idle client connection. Here *idle* means that the connection remains open, yet no operations are requested. By default, no time limit is set.<br>You set this server property with the `dsconf set-server-prop` command.<br>Some applications, such as messaging servers, may open a pool of connections that remain idle when traffic is low, but that should not be closed. Ideally, you might dedicate a replica to support the application in this case. If that is not possible, consider bind DN based individual limits.<br>In any case, set this value high enough not to close connections that other applications expect to remain open, but set it low enough that connections cannot be left idle abusively. Consider setting it to 7200 seconds, which is 2 hours, for example. |
| Attribute<br>`nsslapd-ioblocktimeout` on `dn: cn=config` | Sets the time in milliseconds after which Directory Server closes a stalled client connection. Here *stalled* means that the server is blocked either sending output to the client or reading input from the client.<br>You set this attribute with the `ldapmodify` command.<br>For Directory Server instances particularly exposed to denial of service attacks, consider lowering this value from the default of 1,800,000 milliseconds, which is 30 minutes. |
| Server property<br>`look-through-limit` | Sets the maximum number of candidate entries checked for matches during a search.<br>You set this server property with the `dsconf set-server-prop` command.<br>Some applications, such as messaging servers, may need to search the entire directory. Ideally, you might dedicate a replica to support the application in this case. If that is not possible, consider bind DN based, individual limits.<br>In any case, consider lowering this value from the default of 5000 entries, but not below the threshold value of `search-size-limit`. |

**TABLE 6–1** Tuning Recommendations For Resources Devoted to Client Applications *(Continued)*

| Tuning Parameter | Description |
|---|---|
| Attribute<br>`nsslapd-maxbersize` on `dn: cn=config` | Sets the maximum size in bytes for an incoming ASN.1 message encoded according to Basic Encoding Rules, BER. Directory Server rejects requests to add entries larger than this limit.<br>You set this attribute with the `ldapmodify` command.<br>If you are confident you can accurately anticipate maximum entry size for your directory data, consider changing this value from the default of 2097152, which is 2 MB, to the size of the largest expected directory entry.<br>The next largest size limit for an update is the size of the transaction log file, `nsslapd-db-logfile-size`, which by default is 10 MB. |
| Server property<br>`max-threads-per-connection-count` | Sets the maximum number of threads per client connection.<br>You set this server property with the `dsconf set-server-prop` command.<br>Some applications, such as messaging servers, may open a pool of connections and may issue many requests on each connection. Ideally, you might dedicate a replica to support the application in this case. If that is not possible, consider bind DN based, individual limits.<br>If you anticipate that some applications may perform many requests per connection, consider increasing this value from the default of 5, but do not increase it to more than 10. Typically do not specify more than 10 threads per connection. |
| Server property<br>`search-size-limit` | Sets the maximum number of entries Directory Server returns in response to a search request.<br>You set this server property with the `dsconf set-server-prop` command.<br>Some applications, such as messaging servers, may need to search the entire directory. Ideally, you might dedicate a replica to support the application in this case. If that is not possible, consider bind DN based, individual limits.<br>In any case, consider lowering this value from the default of 2000 entries. |

**TABLE 6–1** Tuning Recommendations For Resources Devoted to Client Applications *(Continued)*

| Tuning Parameter | Description |
|---|---|
| Server property<br>`search-time-limit` | Sets the maximum number of seconds Directory Server allows for handling a search request.<br>You set this server property with the `dsconf set-server-prop` command.<br>Some applications, such as messaging servers, may need to perform very large searches. Ideally, you might dedicate a replica to support the application in this case. If that is not possible, consider bind DN based, individual limits.<br>In any case, set this value as low as you can and still meet deployment requirements. The default value of 3600 seconds, which is 1 hour, is larger than necessary for many deployments. Consider using 600 seconds, which is 10 minutes, as a starting point for optimization tests. |

# Limiting System Resources Used By Directory Server

Table 6–2 describes the parameters that can be used to tune how a Directory Server instance uses system and network resources.

**TABLE 6–2** Tuning Recommendations For System Resources

| Tuning Parameter | Description |
|---|---|
| Attribute<br>`nsslapd-listenhost` on `dn: cn=config` | Sets the hostname for the IP interface on which Directory Server listens. This attribute is multivalued.<br>You set this attribute with the `ldapmodify` command.<br>Default behavior is to listen on all interfaces. The default behavior is adapted for high volume deployments using redundant network interfaces for availability and throughput.<br>Consider setting this value when deploying on a multihomed system, or when listening only for IPv4 or IPv6 traffic on a system supporting each protocol through a separate interface. Consider setting `nsslapd-securelistenhost` when using SSL. |

**TABLE 6–2** Tuning Recommendations For System Resources *(Continued)*

| Tuning Parameter | Description |
|---|---|
| Server property<br>`file-descriptor-count` | Sets the maximum number of file descriptors Directory Server attempts to use.<br>You set this server property with the `dsconf set-server-prop` command.<br>The default value is the maximum number of file descriptors allowed for a process on the system at the time when the Directory Server instance is created. The maximum value corresponds to the maximum number of file descriptors allowed for a process on the system. Refer to your operating system documentation for details.<br>Directory Server uses file descriptors to handle client connections, and to maintain files internally. If the error log indicates Directory Server sometimes stops listening for new connections because not enough file descriptors are available, increasing the value of this attribute may increase the number of client connections Directory Server can handle simultaneously.<br>If you have increased the number of file descriptors available on the system, set the value of this attribute accordingly. The value of this property should be less than or equal to the maximum number of file descriptors available on the system. |
| Attribute<br>`nsslapd-nagle` on `dn: cn=config` | Sets whether to delay sending of TCP packets at the socket level.<br>You set this attribute with the `ldapmodify` command.<br>Consider setting this to `on` if you need to reduce network traffic. |

**TABLE 6–2** Tuning Recommendations For System Resources *(Continued)*

| Tuning Parameter | Description |
|---|---|
| Attribute<br>`nsslapd-reservedescriptors` on `dn: cn=config` | Sets the number of file descriptors Directory Server maintains to manage indexing, replication and other internal processing. Such file descriptors *become unavailable to handle client connections*.<br><br>You set this attribute with the `ldapmodify` command. Consider increasing the value of this attribute from the default of 64 if all of the following are true.<br>■ Directory Server replicates to more than 10 consumers or Directory Server maintains more than 30 index files.<br>■ Directory Server handles a large number of client connections.<br>■ Messages in the error log suggest Directory Server is running out of file descriptors for operations *not* related to client connections.<br><br>Notice that as the number of reserved file descriptors increases, the number of file descriptors available to handle client connections decreases. If you increase the value of this attribute, consider increasing the number of file descriptors available on the system, and increasing the value of `file-descriptor-count`.<br><br>If you decide to change this attribute, for a first estimate of the number of file descriptors to reserve, try setting the value of `nsslapd-reservedescriptors` according to the following formula.<br><br>`20 +`<br>`4 * (number of databases) +`<br>`(total number of indexes) +`<br>`(value of nsoperationconnectionslimit) *`<br>`(number of chaining backends) +`<br>`ReplDescriptors +`<br>`PTADescriptors +`<br>`SSLDescriptors`<br><br>Here *ReplDescriptors* is number of supplier replica plus 8 if replication is used. *PTADescriptors* is 3 if the Pass Through Authentication, PTA, plug-in is enabled, and 0 otherwise. *SSLDescriptors* is 5 if SSL is used, and 0 otherwise.<br><br>The number of databases is the same as the number of suffixes for the instance, unless the instance is configured to use more than one database per suffix. Verify estimates through empirical testing. |

**TABLE 6–2** Tuning Recommendations For System Resources *(Continued)*

| Tuning Parameter | Description |
|---|---|
| Attribute<br>`nsslapd-securelistenhost` on `dn: cn=config` | Sets the hostname for the IP interface on which Directory Server listens for SSL connections. This attribute is multivalued.<br>You set this attribute with the `ldapmodify` command.<br>Default behavior is to listen on all interfaces. Consider this attribute in the same way as `nsslapd-listenhost`. |
| Server property<br>`max-thread-count` | Sets the number of threads Directory Server uses.<br>You set this server property with the `dsconf set-server-prop` command. Consider adjusting the value of this property if any of the following are true.<br>■ Client applications perform many simultaneous, time-consuming operations such as updates or complex searches.<br>■ Directory Server supports many simultaneous client connections.<br>Multiprocessor systems can sustain larger thread pools than single processor systems. As a first estimate when optimizing the value of this attribute, use two times the number of processors or 20 plus the number of simultaneous updates.<br>Consider also adjusting the maximum number of threads per client connection, `max-threads-per-connection-count`. The maximum number of these threads handling client connections cannot exceed the maximum number of file descriptors available on the system. In some cases, it may prove useful to *reduce*, rather than increase, the value of this attribute.<br>Verify estimates through empirical testing. Results depend not only on the particular deployment situation but also on the underlying system. |

# Basic Directory Server Sizing Example: Disk and Memory Requirements

This section provides an example that shows initial steps in sizing Directory Server disk and memory requirements for deployment. The system used for this example was selected by chance and because it had sufficient processing power and memory to complete the sizing tasks quickly. It does not necessarily represent a recommended system for production use. You can it however to gain insight into how much memory and disk space might be required for production systems.

## System Characteristics

The following system information was observed using the Solaris Management Console (`smc`).

- 2 AMD64 CPUs (2.2 gigahertz)
- Solaris 10 Operating System
- 4 gigabytes physical memory
- 40 gigabytes swap
- Physical memory in use before Directory Server installation: 700 megabytes
- Physical memory free before Directory Server installation: 3400 megabytes
- CPU usage: 1%
- Local disk: one partition formatted UFS with logging

For this example, the system was dedicated to Directory Server. No other user was logged in, and only the default system processes were running.

## Preparing a Directory Server Instance

After unpacking the zip distribution, install Directory Server software on local disk space.

```
$ ./dsee_deploy install -c DS -i /local
```

For convenience set environment variables as shown.

```
$ export PATH=/local/ds6/bin:/local/dsrk6/bin:/local/dsee6/bin:${PATH}
$ export DIRSERV_PORT=1389
$ export LDAP_ADMIN_PWF=~/.pwd
```

After installing the software and setting environment variables, create a Directory Server instance using default ports for LDAP and LDAPS, respectively.

```
$ dsadm create -p 1389 -P 1636 /local/ds
Choose the Directory Manager password:
Confirm the Directory Manager password:
$ du -hs /local/ds
610K   /local/ds
```

Until you create a suffix, the Directory Server instance uses less than one megabyte of disk space.

```
$ dsadm start /local/ds
Server started: pid=8046
$ dsconf create-suffix dc=example,dc=com
Certificate "CN=hostname, CN=1636, CN=Directory Server,
 O=Sun Microsystems" presented by the server is not trusted.
Type "Y" to accept, "y" to accept just once, "n" to refuse, "d" for more
 details: Y
$ du -hs /local/ds
53M   /local/ds
```

For this example, make no additional changes to the default Directory Server configuration except those shown explicitly.

## Populating the Suffix With 10,000 Sample Directory Entries

Using the `makeldif` command with the example files provided as part of Directory Server Resource Kit, you can create sample LDIF files from one kilobyte to one megabyte in size. See "To Install Directory Server Enterprise Edition 6.3 From Zip Distribution" in *Sun Java System Directory Server Enterprise Edition 6.3 Installation Guide* for an example showing how to use the `makeldif` command.

The entries in these files are smaller than you would expect in a real deployment.

```
$ du -h /var/tmp/*
 57M   /var/tmp/100k.ldif
 5.7M   /var/tmp/10k.ldif
 573M   /var/tmp/1M.ldif
```

An example entry from these files is shown in the following LDIF.

```
dn: uid=Aartjan.Aalders,ou=People,dc=example,dc=com
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
givenName: Aartjan
sn: Aalders
cn: Aartjan Aalders
initials: AA
uid: Aartjan.Aalders
mail: Aartjan.Aalders@example.com
userPassword: trj49xeq
telephoneNumber: 935-748-6699
homePhone: 347-586-0252
pager: 906-399-8417
mobile: 452-898-9034
employeeNumber: 1000004
street: 64197 Broadway Street
l: Lawton
st: IN
postalCode: 57924
postalAddress: Aartjan Aalders$64197 Broadway Street$Lawton, IN  57924
description: This is the description for Aartjan Aalders.
```

Begin sizing by importing the content of `10k.ldif`, which occupies 5.7 megabytes on disk.

```
$ dsadm stop /local/ds
Server stopped
```

```
$ dsadm import -i /local/ds /var/tmp/10k.ldif dc=example,dc=com
...
```

With default indexing the content of `10k.ldif` increases the size of the instance files by 72 megabytes - 53 megabytes, or 19 megabytes.

```
$ du -hs /local/ds
 72M   /local/ds
```

If you index five more attributes, size increases by about seven megabytes.

```
$ dsconf create-index dc=example,dc=com employeeNumber street st \
 postalCode description
$ dsconf reindex dc=example,dc=com
...
## example: Finished indexing.

Task completed (slapd exit code: 0).
$ du -hs /local/ds
 79M   /local/ds
```

Observing memory size with the default cache settings, and nothing loaded from the suffix into entry cache yet, the server process occupies approximately 170 megabytes of memory with a heap size of about 56 megabytes.

```
$ dsadm start /local/ds
Server started: pid=8482
$ pmap -x 8482
...
         Address     Kbytes        RSS       Anon     Locked Mode   Mapped File
0000000000437000      61348      55632      55380          - rw---    [ heap ]
...
---------------- ---------- ---------- ---------- ----------
        total Kb     178444     172604      76532          -
```

As you then prime the cache and examine output from the `pmap` command again, the heap grows by about 10 megabytes, and so does the total size of the process.

```
$ ldapsearch -D cn=Directory\ Manager -w - -p 1389 -b dc=example,dc=com \
 objectclass=\* > /dev/null
Enter bind password:
$ pmap -x 8482
...
         Address     Kbytes        RSS       Anon     Locked Mode   Mapped File
...
0000000000437000      70564      65268      65024          - rw---    [ heap ]
...
---------------- ---------- ---------- ---------- ----------
        total Kb     187692     182272      86224          -
```

The numbers are comparable for default indexing.

```
$ dsconf delete-index dc=example,dc=com employeeNumber street st \
 postalCode description
$ dsconf reindex dc=example,dc=com
...
## example: Finished indexing.

Task completed (slapd exit code: 0).
$ dsadm stop /local/ds
 Server stopped
$ dsadm start /local/ds
 Server started: pid=8541
$ ldapsearch -D cn=Directory\ Manager -w - -p 1389 -b dc=example,dc=com \
 objectclass=\* > /dev/null
Enter bind password:
$ pmap -x 8541
...
         Address     Kbytes        RSS       Anon     Locked Mode   Mapped File
...
0000000000437000      70564      65248      65004          - rw---    [ heap ]
...
---------------- ---------- ---------- ---------- ----------
        total Kb     187680     182240      86192          -
```

For only 10,000 entries, do not change the default cache sizes.

```
$ dsconf get-server-prop | grep cache
db-cache-size                     :  32M
import-cache-size                 :  64M
$ dsconf get-suffix-prop dc=example,dc=com | grep entry-cache-size
entry-cache-size                  :  10M
```

The small default entry cache was no doubt filled completely after priming, even with only 10,000 entries. To see the size for a full entry cache, set a large entry cache size, import the data again, and prime the cache.

```
$ dsconf set-suffix-prop dc=example,dc=com entry-cache-size:2G
$ dsadm stop /local/ds
Server stopped
$ dsadm import -i /local/ds /var/tmp/10k.ldif dc=example,dc=com
...
$ dsadm start /local/ds
Server started: pid=8806
$ ldapsearch -D cn=Directory\ Manager -w - -p 1389 -b dc=example,dc=com \
 objectclass=\* > /dev/null
Enter bind password:
$ pmap -x 8806
```

```
...
         Address     Kbytes        RSS       Anon     Locked Mode   Mapped File
...
0000000000437000     116644     109996     109780          - rw---    [ heap ]
```

Here 10,000 entries occupy approximately 55 megabytes of entry cache memory (110 - 55).

## Populating the Suffix With 100,000 Sample Directory Entries

As you move to 100,000 entries, you have more directory data to fit into database and entry caches. Initially, import 100,000 entries and examine the size required on disk for this volume of directory data.

```
$ dsadm import -i /local/ds /var/tmp/100k.ldif dc=example,dc=com
...
$ du -hs /local/ds
 196M   /local/ds
```

Directory data contained in the database for our example suffix, `dc=example,dc=com`, now occupy about 142 megabytes.

```
$ du -hs /local/ds/db/example/
 142M   /local/ds/db/example
```

You can increase the size of the database cache to hold this content. If you expect the volume of directory data to grow over time, you can set the database cache larger than currently necessary. You can also set the entry cache size larger than necessary. Entry cache grows as the server responds to client requests, unlike the database cache, which is allocated at startup.

```
$ dsconf set-server-prop db-cache-size:200M
$ dsconf set-suffix-prop dc=example,dc=com entry-cache-size:2G

$ dsadm stop /local/ds
 Server stopped
$ dsadm start /local/ds
 Server started: pid=8640
$ pmap -x 8640
...
         Address     Kbytes        RSS       Anon     Locked Mode   Mapped File
...
0000000000437000      61348      55404      55148          - rw---    [ heap ]
...
---------------- ---------- ---------- ---------- ----------
        total Kb     491984     485736     174620          -
```

This shows the server instance has a relatively small heap at startup, but that the database cache memory has been allocated. Process size is nearing half a gigabyte.

```
$ ldapsearch -D cn=Directory\ Manager -w - -p 1389 -b dc=example,dc=com \
 objectclass=\* > /dev/null
Enter bind password:
$ pmap -x 8640
...
         Address     Kbytes        RSS       Anon     Locked Mode   Mapped File
...
0000000000437000     610212     604064     603840          - rw---    [ heap ]
...
---------------- ---------- ---------- ---------- ----------
        total Kb    1040880    1034428     723360          -
```

Heap size now reflects the entry cache being filled. It has increased by roughly 550 megabytes for 100,000 small directory entries, whose LDIF occupied 57 megabytes on disk.

With five extra indexes, the process size is about the same. The database cache size has not changed.

```
$ dsconf create-index dc=example,dc=com employeeNumber street st \
 postalCode description
$ dsadm stop /local/ds
 Server stopped
$ dsadm import -i /local/ds /var/tmp/100k.ldif dc=example,dc=com
...
$ dsadm start /local/ds
 Server started: pid=8762
$ ldapsearch -D cn=Directory\ Manager -w - -p 1389 -b dc=example,dc=com \
 objectclass=\* > /dev/null
Enter bind password:
$ pmap -x 8762
...
         Address     Kbytes        RSS       Anon     Locked Mode   Mapped File
...
0000000000437000     610212     603832     603612          - rw---    [ heap ]
...
---------------- ---------- ---------- ---------- ----------
        total Kb    1040876    1034192     723128          -
```

The database is somewhat larger, however. The additional indexes increased the size of the database from 142 megabytes to 163 megabytes.

```
$ du -hs /local/ds/db/example/
 163M   /local/ds/db/example
```

## Populating the Suffix With 1,000,000 Sample Directory Entries

As you move from 100,000 entries to 1,000,000 entries, you no longer have enough space on a system with 4 gigabytes of physical memory to include all entries in the entry cache. You can begin by importing the data and examining the size it occupies on disk.

```
$ dsadm import -i /local/ds /var/tmp/1M.ldif dc=example,dc=com
...
$ du -hs /local/ds/db/example/
 1.3G   /local/ds/db/example
```

Assuming you expect approximately 25% growth in directory data size during the lifetime of the instance, set the database cache size to 1700 megabytes.

```
$ dsadm start /local/ds
Server started: pid=9060
$ dsconf set-server-prop db-cache-size:1700M
$ dsadm stop /local/ds
Server stopped
$ dsadm start /local/ds
Server started: pid=9118
$ pmap -x 9118
...
         Address     Kbytes        RSS       Anon     Locked Mode   Mapped File
...
0000000000437000      65508      55700      55452          - rw---    [ heap ]
...
---------------- ---------- ---------- ---------- ----------
        total Kb    1882448    1034180      76616          -
```

Given a database cache this large and only 4 gigabytes of physical memory, you cannot fit more than a fraction of entries into the entry cache for the suffix. Here, set entry cache size to one gigabyte, and then prime the cache to see the change in the process heap size.

```
$ dsconf set-suffix-prop dc=example,dc=com entry-cache-size:1G
$ ldapsearch -D cn=Directory\ Manager -w - -p 1389 -b dc=example,dc=com \
 objectclass=\* > /dev/null
Enter bind password:
$ pmap -x 9118
...
         Address     Kbytes        RSS       Anon     Locked Mode   Mapped File
...
0000000000437000    1016868    1009852    1009612          - rw---    [ heap ]
...
---------------- ---------- ---------- ---------- ----------
        total Kb    2883268    2477064    1080076          -
```

Total process size is over 2.8 gigabytes.

```
$ prstat -p 9118
   PID USERNAME  SIZE   RSS STATE  PRI NICE      TIME  CPU PROCESS/NLWP
  9118 myuser   2816M 2374M sleep   59    0   0:03:26 0.5% ns-slapd/42
```

Extrapolating from earlier entry cache sizes, you can expect to use 5.5 or 6 gigabytes for entry cache alone if you had enough physical memory.

Examining the directory database size with five additional indexes, you find adding indexes has increased the size of the database by about 200 megabytes.

```
$ dsconf create-index dc=example,dc=com employeeNumber street st \
 postalCode description
$ dsadm stop /local/ds
Server stopped
$ dsadm import -i /local/ds /var/tmp/1M.ldif dc=example,dc=com
...
$ du -hs /local/ds/db/example
 1.5G   /local/ds/db/example
```

### Summary of Observations

Table 6–3 records what was observed in this example. It includes neither server process size, nor default database cache file size.

---

**Note –** Your observations made through empirical testing for your deployment are likely to differ significantly from those shown here.

---

**TABLE 6–3** Sizing Summary

| Number of Entries | LDIF File Size | Disk with Default Indexes | Disk with Five Additional Indexes | Database Cache | Entry Cache |
|---|---|---|---|---|---|
| 0[1] | n/a | n/a | n/a | n/a | n/a |
| 10,000 | 5.7 megabytes | 19 megabytes | 26 megabytes | 32 megabytes | 55 megabytes |
| 100,000 | 57 megabytes | 142 megabytes | 163 megabytes | 200 megabytes | 550 megabytes |
| 1,000,000 | 573 megabytes | 1300 megabytes | 1500 megabytes | 1700 megabytes (default indexing) | n/a |

[1] The suffix has been created, but is empty.

In an actual deployment, you may have significantly larger entries and more indexed attributes. Do your own empirical testing and tuning before ordering hardware.

# Operating System Tuning For Directory Server

Default system settings do not necessarily result in top directory service performance. This section addresses how to tune the operating system for top performance.

# Operating System Version and Patch Support

See *Sun Java System Directory Server Enterprise Edition 6.3 Release Notes* for an updated list of supported operating systems.

You want to maintain overall system security. You also want to ensure proper Directory Server operation. You therefore install the latest recommended system patches, service packs, or fixes. See *Sun Java System Directory Server Enterprise Edition 6.3 Release Notes* for an updated list of the latest patches to apply for your platform.

# Basic Security Checks

The recommendations in this section do not eliminate all risk. Instead, the recommendations are intended as a short checklist to help you limit typical security risks.

- **Isolate and firewall the system.** If at all possible, isolate the system where Directory Server runs from the public Internet with a network firewall.
- **Do not allow dual boot.** Do not run other operating systems on the system that runs a production Directory Server. Other systems can permit access to files, which you should not allow.
- **Use strong passwords.** Use a root password at least eight characters long. The password should include punctuation or other non-alphabetic characters.

  You can use the Strong Password Check server plug-in to refuse weak passwords. The `dsconf` server property `pwd-strong-check-enabled` can be used to turn the plug-in on.

  If you choose to use longer operating system passwords, you might have to configure the way passwords are handled by the system. See your operating system documentation for instructions.
- Use a safe user and group ID for the server. For security reasons, do not run Directory Server with super user privileges.

  You can, for example, use the UNIX commands `groupadd` and `useradd` to create a user and group without login privileges. You can then run the server as this user and group.

  For example, to add a group that is named servers, do the following.

  ```
  # groupadd servers
  ```

  To add a user named `server1` as a member of the group `servers`, use the following command.

```
# useradd -g servers -s /bin/false -c "server1"
```

A particular deployment can call for sharing Directory Server files with other servers, such as a messaging server. In such a deployment, consider running the servers with the same user, group ID.

- **Use the core facility.** To facilitate debugging, you can allow processes running with this user, group ID to dump core. Use a utility such as the Solaris command `coreadm`. For example, you can enable Directory Server to generate core files by allowing `setuid` processes to do so, and updating the `coreadm` configuration:

  ```
  # coreadm -e proc-setid
  # coreadm -u
  ```

  When scripting server startup, you can add the following line to your startup script. The line allows Directory Server to generate core files of the form `core.ns-slapd.`*pid*, where *pid* is the process ID.

  ```
  coreadm -p core.%f.%p $$
  ```

- **Disable unnecessary services.** For top performance with less risk, dedicate the system to Directory Server. As explained elsewhere in this guide, do not run Directory Service Control Center on the same system. When you run additional services, especially network services, you negatively affect server performance and scalability. You can also thereby increase security risks.

  Disable as many network services as possible. Directory Server does not require file sharing and other services. Disable services such as IP Routing, Mail, NetBIOS, NFS, RAS, Web Publishing, and Windows Network Client services. Consider disabling `telnet`, and `ftp`.

  As with many network services, `telnet` and `ftp` pose security risks. These two services are particularly dangerous, because the commands transmit user passwords in clear text over the network. Work around the need for `telnet` and `ftp` by using clients such as Secure Shell, `ssh`, and Secure FTP, `sftp`, instead. See your operating system documentation for details on disabling network services.

  If the Directory Server instance does not provide the naming service for the network, consider enabling a naming service for the system. Directory Server then uses the naming service for example when resolving ACIs.

# Accurate System Clock Time

Ensure the system clock is reasonably in sync with other systems. Good clock synchronization facilitates replication. Good synchronization also facilitates correlation of date and time stamps in log files between systems. Consider using a Network Time Protocol, NTP, client to set the correct system time.

## Restart When System Reboots

You can enable a server instance to restart at system boot time by using the `dsadm` command. For example, use the `dsadm enable-service` command on Solaris 10 and Windows systems. On other systems, use the `dsadm autostart` command. If you did not install from native packages, refer to your operating system documentation for help ensuring the server starts at system boot time.

When possible, stop Directory Server with the `dsadm` command, or from DSCC. If the Directory Server is stopped abruptly during system shutdown, there is no way to guarantee that all data has been written to disk correctly. When Directory Server restarts, it must therefore verify the database integrity. This process can take some time.

Furthermore, consider using a logging option with your file system. File system logging generally both improves write performance, and also decreases the time required to perform a file system check. The system must check the file system when the file system is not cleanly unmounted during a crash. Also, consider using RAID for storage.

## System-Specific Tuning With The `idsktune` Command

The `idsktune`(1M) utility that is provided with the product can help you diagnose basic system configuration shortcomings. The utility offers recommendations for tuning the system to support high performance directory services. The utility does not actually implement any of the recommendations. The recommendations should be implemented by a qualified system administrator.

When you run the utility as `root`, `idsktune` gathers information about the system. The utility displays notices, warnings, and errors with recommended corrective actions. The `idsktune` command checks the following.

- Operating system and kernel versions are supported for this release.
- Available memory, and available disk space meet minimum requirements for typical use.
- System resource limits meet minimum requirements for typical use.
- Required patches are installed.

---

**Tip –** Fix at minimum all `ERROR` conditions before installing Directory Server software on a system intended for production use.

---

Individual deployment requirements can exceed minimum requirements. You can provide more resources than the resources identified as minimum system requirements by the `idsktune` utility.

Consider local network conditions and other applications before implementing specific recommendations. Refer to the operating system documentation for additional tips on tuning network settings.

## File Descriptor Settings

Directory Server uses file descriptors when handling concurrent client connections. A low maximum number of file descriptors that are available for the server process can thus limit the number of concurrent connections. Recommendations that concern the number of file descriptors therefore relate to the number of concurrent connections Directory Server can handle.

On Solaris systems, the number of file descriptors available is configured through the `rlim_fd_max` parameter. Refer to the operating system documentation for further instructions on modifying the number of available file descriptors.

## Transmission Control Protocol (TCP) Settings

Specific network settings depend on the platform. On some systems, you can enhance Directory Server performance by modifying TCP settings.

---

**Note –** First deploy your directory service, then consider tuning these parameters, if necessary.

---

This section discusses the reasoning behind `idsktune` recommendations that concern TCP settings, and provides a method for tuning these settings on Solaris 10 systems.

### Inactive Connections

Some systems allow you to configure the interval between transmission of `keepalive` packets. This setting can determine how long a TCP connection is maintained while inactive and potentially disconnected. When set too high, the `keepalive` interval can cause the system to use unnecessary resources to keep connections for clients that have become disconnected. For most deployments, set this parameter to a value of 600 seconds. This value, which is 600,000 milliseconds, or 10 minutes, allows more concurrent connections to Directory Server.

When set too low, however, the `keepalive` interval can cause the system to drop connections during transient network outages.

On Solaris systems, this time interval is configured through the `tcp_keepalive_interval` parameter.

### Outgoing Connections

Some systems allow you to configure how long a system waits for an outgoing connection to be established. When set too high, establishing outgoing connections to destination servers such as replicas not responding quickly can cause long delays. For Intranet deployments on fast, reliable networks, you can set this parameter to a value of 10 seconds to improve performance. Do not, however, use such a low value on networks with slow, unreliable, or WAN connections, however.

On Solaris systems, this time interval is configured through the `tcp_ip_abort_cinterval` parameter.

### Retransmission Timeout

Some systems allow you to configure the initial time interval between retransmission of packets. This setting affects the wait before retransmission of an unacknowledged packet. When set too high, clients can be kept waiting on lost packets. For Intranet deployments on fast, reliable networks, you can set this parameter to a value of 500 milliseconds to improve performance. Do not, however, use such a low value on networks with round trip times of more than 250 milliseconds.

On Solaris systems, this time interval is configured through the `tcp_rexmit_interval_initial` parameter.

### Sequence Numbers

Some systems allow you to configure how the system handles initial sequence number generation. For extranet and Internet deployments, set this parameter so initial sequence number generation is based on RFC 1948 to prevent sequence number attacks. In such environments, other TCP tuning settings mentioned here are not useful.

On Solaris systems, this behavior is configured through the `tcp_strong_iss` parameter.

## Tuning TCP Settings on Solaris 10 Systems

On Solaris 10 systems, the simplest way to tune TCP settings is to create a simple SMF service as follows:

- Create an SMF profile for Directory Server tuning.
- Edit the following xml file according to your environment and save the file as `/var/svc/manifest/site/ndd-nettune.xml`.

```
<?xml version="1.0"?>
<!DOCTYPE service_bundle SYSTEM "/usr/share/lib/xml/dtd/ service_bundle.dtd.1">
<!--
    ident    "@(#)ndd-nettune.xml    1.0    04/09/21 SMI"
-->

<service_bundle type='manifest' name='SUNWcsr:ndd'>

<service
    name='network/ndd-nettune'
    type='service'
    version='1'>

            <create_default_instance enabled='true' />

            <single_instance />

        <dependency
        name='fs-minimal'
        type='service'
        grouping='require_all'
        restart_on='none'>
            <service_fmri value='svc:/system/filesystem/minimal' />
        </dependency>

        <dependency
        name='loopback-network'
        grouping='require_any'
        restart_on='none'
        type='service'>
            <service_fmri value='svc:/network/loopback' />
        </dependency>

        <dependency
        name='physical-network'
        grouping='optional_all'
        restart_on='none'
        type='service'>
```

```
        <service_fmri value='svc:/network/physical' />
    </dependency>

    <exec_method
    type='method'
    name='start'
    exec='/lib/svc/method/ndd-nettune'
    timeout_seconds='3' />
    </exec_method>

    <exec_method
           type='method'
           name='stop'
           exec=':true'
           timeout_seconds='3' >
    </exec_method>

    <property_group name='startd' type='framework'>
        <propval name='duration' type='astring'  value='transient' />
    </property_group>

    <stability value='Unstable' />

    <template>
        <common_name>
            <loctext xml:lang='C'>
                ndd network tuning
            </loctext>
        </common_name>
        <documentation>
            <manpage title='ndd' section='1M'
            manpath='/usr/share/man' />
        </documentation>
    </template>

</service>

</service_bundle>
```

- Before you import the `ndd-nettune.xml` configuration, verify that the syntax is correct. You can do this by running the following command:

  ```
  $ svccfg validate /var/svc/manifest/site/ndd-nettune.xml
  ```

- Import the configuration by running the following command:

  ```
  $ svccfg import /var/svc/manifest/site/ndd-nettune.xml
  ```

  For more information see the `svccfg(1M)` man page.

- Copy the following shell script into `/lib/svc/method/ndd-nettune`.

```
#!/sbin/sh
#
# ident   "@(#)ndd-nettune.xml    1.0     01/08/06 SMI"

. /lib/svc/share/smf_include.sh
. /lib/svc/share/net_include.sh

# Make sure that the libraries essential to this stage of booting  can be found.
LD_LIBRARY_PATH=/lib; export LD_LIBRARY_PATH
echo "Performing Directory Server Tuning..." >> /tmp/smf.out
/usr/sbin/ndd -set /dev/tcp tcp_conn_req_max_q 1024
/usr/sbin/ndd -set /dev/tcp tcp_keepalive_interval 600000
/usr/sbin/ndd -set /dev/tcp tcp_ip_abort_cinterval 10000
/usr/sbin/ndd -set /dev/tcp tcp_ip_abort_interval 60000


# Reset the library path now that we are past the critical stage
unset LD_LIBRARY_PATH
```

- Run `svcadm` to enable `nettune` (for more information, see the `svcadm(1M)` man page).
- Run `svcs -x` (for more information see the `svcs(1)` man page).

# Physical Capabilities of Directory Server

Following are the physical capabilities of Directory Server that specify its scalability:

- Process size. Depending on the operating system, the 32-bit versions of Directory Server supports 2GB – 4GB process size. The process size on 64-bit versions of Directory Server is defined by the amount of physical memory available on the machine. It is tested with 128GB process size.
- Number of LDAP entries. The total number of LDAP entries that can be created on a single server instance is 2^32 -1, that is, 4G entries.
- Size of each entry. The size of a single record in LDAP server is 4GB as per the DB itself. The size of an entry also depends on maximum size of the LDAP request (`maxbersize`). Its maximum value is 2 GB.
- Number of LDAP connections. The number of LDAP connections depends on the number of file descriptors that a process can open. Note that too many open connections tend to degrade performances.
- Size of LDAP Server (Berkery DB). The size of an LDAP server is defined by the size of your filesystem.

◆ ◆ ◆ CHAPTER 7

# Identifying Security Requirements

How you secure data in Directory Server Enterprise Edition has an impact on all other areas of design. This chapter describes how to analyze your security needs and explains how to design your directory service to meet those needs.

This chapter covers the following topics:

# Security Threats

The most typical threats to directory security include the following:

- **Eavesdropping.** Information remains intact, but its privacy is compromised. For example, someone could learn your credit card number, record a sensitive conversation, or intercept classified information.
- **Unauthorized access.** This threat includes unauthorized access to data through data-fetching operations. Unauthorized users might also gain access to reusable client authentication information by monitoring the access of others. The Directory Server Enterprise Edition authentication methods, password policies, and access control mechanisms provide effective ways of preventing unauthorized access.
- **Tampering.** Information in transit is changed or replaced and then sent on to the recipient. For example, someone could alter an order for goods or change a person's resume.

  This threat includes unauthorized modification of data or configuration information. If your directory cannot detect tampering, an attacker might alter a client's request to the server. The attacker might also cancel the request or change the server's response to the client. The Secure Socket Layer (SSL) protocol and similar technologies can solve this problem by signing information at either end of the connection.
- **Impersonation.** Information passes to a person who poses as the intended recipient.

  Impersonation can take two forms, spoofing and misrepresentation.

  - **Spoofing.** A person or computer impersonates someone else. For example, a person can pretend to have the mail address `jdoe@example.com`, or a computer can identify itself as a site called `www.example.com` when it is not.
  - **Misrepresentation.** A person or organization misrepresents itself. For example, suppose the site `www.example.com` pretends to be a furniture store when it is really just a site that takes credit-card payments but never sends any goods.
- **Denial of service.** An attacker uses the system resources to prevent these resources from being used by legitimate users.

  In a denial of service attack, the attacker's goal is to prevent the directory from providing service to its clients. Directory Server Enterprise Edition provides a way of preventing denial of service attacks by setting limits on the resources that are allocated to a particular bind DN.

# Overview of Security Methods

A security policy must be able to prevent sensitive information from being modified or retrieved by unauthorized users, but easy enough to administer.

Directory Server Enterprise Edition provides the following security methods:

- **Authentication.** Provides a means for one party to verify another's identity. For example, a client gives a password to Directory Server during an LDAP bind operation. As part of the authentication process, *password policies* define the criteria that a password must satisfy to be considered valid, for example, age, length, and syntax. *Account inactivation* disables a user account, group of accounts, or an entire domain so that all authentication attempts are automatically rejected.
- **Encryption.** Protects the privacy of information. When data is encrypted, the data is scrambled in a way that only the recipient can decode. The *Secure Sockets Layer* (SSL) maintains data integrity by encrypting information in transit. If encryption and message digests are applied to the information being sent, the recipient can determine that the information was not tampered with during transit. *Attribute encryption* maintains data integrity by encrypting stored information.
- **Access control.** Tailors the access rights that are granted to different directory users, and provides a means of specifying required credentials or bind attributes.
- **Auditing.** Enables you to determine if the security of your directory has been compromised. For example, you can audit the log files maintained by your directory.

These security tools can be used in combination in your security design. You can also use other features of the directory, such as replication and data distribution, to support your security design.

# Determining Authentication Methods

Directory Server Enterprise Edition supports the following authentication mechanisms:


- "Anonymous Access" on page 114
- "Simple Password Authentication" on page 114
- "Simple Password Authentication Over a Secure Connection" on page 115
- "Certificate-Based Client Authentication" on page 115
- "SASL-Based Client Authentication" on page 116


The same authentication mechanism applies to all users, whether the users are people or LDAP-aware applications.

Apart from the authentication mechanisms described above, this section also includes the following information about authentication:


- "Preventing Authentication by Account Inactivation" on page 116
- "Preventing Authentication by Using Global Account Lockout" on page 116
- "External Authentication Mappings and Services" on page 117

## Anonymous Access

Anonymous access is the simplest form of directory access. Anonymous access makes data available to any directory user, regardless of whether the user has authenticated.

Anonymous access does not allow you to track who is performing searches or what kind searches are being performed, only that someone is performing searches. When you allow anonymous access, anyone who connects to your directory can access the data. If you allow anonymous access to data, and attempt to block a user or group from that data, the user can access the data by binding to the directory anonymously.

You can restrict the privileges of anonymous access. Usually, directory administrators allow anonymous access only for read, search, and compare privileges. You can also limit access to a subset of attributes that contain general information such as names, telephone numbers, and email addresses. Do not allow anonymous access to sensitive data, such as government identification numbers, home telephone numbers and addresses, and salary information.

Anonymous access to the root DSE (base DN "") is required. Access to the root DSE enables applications to discover the capabilities of the server, the supported security mechanisms, and the supported suffixes.

## Simple Password Authentication

If anonymous access is not set up, a client must authenticate to Directory Server to access the directory contents. With simple password authentication, a client authenticates to the server by providing a simple, reusable password.

The client authenticates to Directory Server through a bind operation in which the client provides a distinguished name and credentials. The server locates the entry that corresponds to the client DN, then checks whether the client's password matches the value stored with the entry. If the password matches, the server authenticates the client. If the password does not match, the authentication operation fails and the client receives an error message.

---

**Note –** The drawback of simple password authentication is that the password is transmitted in clear text, which can compromise security. If a rogue user is listening, that user can impersonate an authorized user.

---

Simple password authentication offers an easy way of authenticating users. However, you need to restrict the use of simple password authentication to your organization's intranet. This kind of authentication does not offer the level of security that is required for transmissions between business partners over an extranet or for transmissions with customers on the Internet.

## Simple Password Authentication Over a Secure Connection

A secure connection uses encryption to make data unreadable to third parties while the data is sent over the network between Directory Server and its clients. Clients can establish secure connections in either of the following ways:

- Binding to the secure port by using the Secure Socket Layer (SSL)
- Binding to an insecure port with anonymous access, then sending the Start TLS control to begin using Transport Layer Security (TLS)

In either case, the server must have a security certificate, and the client must be configured to trust this certificate. The server sends its certificate to the client to perform *server authentication*, using public-key cryptography. This results in the client knowing that it is connected to the intended server and that the server is not being tampered with.

Then, for privacy, the client and server encrypt all data transmitted through the connection. The client sends the bind DN and password on the encrypted connection to authenticate the user. All further operations are performed with the identity of the user. The operations might also be performed with a proxy identity if the bind DN has proxy rights to other user identities. In all cases, the results of operations are encrypted when these results are returned to the client.

## Certificate-Based Client Authentication

When establishing encrypted connections over SSL or TLS, you can also configure the server to require *client authentication*. The client must send its credentials to the server to confirm the identity of the user. The user's certificate, not the DN, is used to determine the bind DN. Client authentication protects against user impersonation and is the most secure type of connection.

Certificate-based client authentication operates at the SSL, TLS layer only. To use a certificate-based authentication ID with LDAP, you must use SASL EXTERNAL authentication after establishing the SSL connection.

You can configure certificate-based client authentication by using the `dsconf get-server-prop` command. See `dsconf`(1M) for more information.

## SASL-Based Client Authentication

Client authentication during an SSL or TLS connection can also use the Simple Authentication and Security Layer (SASL), a generic security interface, to establish the identity of the client. Directory Server Enterprise Edition supports the following mechanisms through SASL:

- **DIGEST-MD5.** This mechanism authenticates clients by comparing a hashed value sent by the client with a hash of the user's password. However, because the mechanism must read user passwords, all users wanting to be authenticated through DIGEST-MD5 must have `{CLEAR}` passwords in the directory.
- **GSSAPI.** The General Security Services API (GSSAPI) is available only on the Solaris Operating System. It allows Directory Server to interact with the Kerberos V5 security system to identify a user. The client application must present its credentials to the Kerberos system, which in turn validates the user's identity to Directory Server.
- **EXTERNAL.** This mechanism authenticates a user in LDAP based on the credentials specified by an external security layer, such as SSL or TLS.

For more information, see "Using SASL DIGEST-MD5 in Clients" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide* and "Using Kerberos SASL GSSAPI in Clients" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

## Preventing Authentication by Account Inactivation

You can temporarily prevent authentication by inactivating a user account or a set of accounts. When the account is inactivated, the user cannot bind to Directory Server, and authentication operations fail. For more information, see "Manually Locking Accounts" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

## Preventing Authentication by Using Global Account Lockout

In this version of Directory Server, authentication failures with a password are monitored and replicated. This enables rapid, global account lockout after a specified number of authentication attempts with an invalid password. Global account lockout is supported in any of the following cases:

- Client applications bind to a single server in the topology only
- The topology does not include any read-only consumers
- Directory Proxy Server is used to control all bind traffic

Imagine a situation where all bind attempts are not directed to the same server, and the client application performs bind attempts on multiple servers faster than lockout data can be replicated. In the worst-case scenario, the client would be allowed the specified number of

attempts on each server where the client attempted to bind. This situation would be unlikely if the client application were driven by input from a human user. However, an automated client built to attack a topology could exploit this deployment choice.

Prioritized replication can be used to minimize the impact of asynchronous replication latency on intrusion detection. However, you might require account lockout immediately after the specified number of failed bind attempts. In this situation, you must use Directory Proxy Server to route all bind attempts on a particular entry to the same master replica. For information about how to configure Directory Proxy Server to do this, see "Operational Affinity Algorithm for Global Account Lockout" in *Sun Java System Directory Server Enterprise Edition 6.3 Reference*.

To retain a strictly local lockout policy in a replicated topology, you must maintain compatibility with the 5.2 password policy. In this situation, the policy in effect must not be the default password policy. Local lockout can also be achieved by restricting binds to read-only consumers.

For detailed information about how global account lockout works, see "Global Account Lockout" in *Sun Java System Directory Server Enterprise Edition 6.3 Reference*.

## External Authentication Mappings and Services

Directory Server provides user account host mapping, which associates a network user account with a Directory Server user account. This feature simplifies the management of both user accounts. Host mapping is required for users who are externally authenticated.

# Proxy Authorization

Proxy authorization is a special form of access control. Proxy authorization or proxy authentication is when an application is forced to use a specific username/password combination to gain access to the server.

With proxy authorization, an administrator can request access to Directory Server by assuming the identity of a regular user. The administrator binds to the directory with his own credentials and is granted the rights of the regular user. This assumed identity is called the *proxy user*. The DN of that user is called the *proxy DN*. The proxy user is evaluated as a regular user. Access is denied if the proxy user entry is locked or inactivated or if the password has expired.

An advantage of the proxy mechanism is that you can enable an LDAP application to use a single bind to service multiple users who are accessing Directory Server. Instead of each user having to bind and authenticate, the client application binds to Directory Server and uses proxy rights.

For more information, see Chapter 7, "Directory Server Access Control," in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

# Designing Password Policies

A password policy is a set of rules that govern how passwords are administered in a system. Directory Server supports multiple password policies, as well as a default password policy.

Several elements of the password policy are configurable, enabling you to design a policy that suits the security requirements of your organization. Configuration of the password policy is described in Chapter 8, "Directory Server Password Policy," in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*. The individual attributes available for configuring password policies are described in the `pwpolicy`(5dssd) man page.

This section is divided into the following topics:

- Password policy options
- Password policies in a replicated environment
- Password policy migration

## Password Policy Options

The following password policy options are provided:

- A default password policy is applied. The parameters of this default policy can be changed.
- An additional, specialized password policy can be applied to a particular user.
- An additional, specialized password policy can be applied to a set of users by using the CoS and Roles functionality. Password policies cannot be applied to static groups.

## Password Policies in a Replicated Environment

Configuration information for the default password policy is not replicated. Instead, it is part of the server instance configuration. If you modify the default password policy, the same modifications must be made on each server in the topology. If you need a password policy that *is* replicated, you must define a specialized password policy under a part of the directory tree that is replicated.

All password information that is stored in the user entry is replicated. This information includes the current password, password history, password expiration dates and so forth.

Consider the following impact of password policies in a replicated environment:

- A user with an impending password expiration receives a warning from every replica to which the user binds before changing his password.
- When a user changes his password, the new password might take a while to be updated on all replicas. A situation could arise where a user changes his password and then immediately rebinds to one of the consumer replicas with the new password. In this case, the bind could fail until the replica receives the updated password. This situation can be alleviated using prioritized replication to force password changes to be replicated first.

## Password Policy Migration

The Directory Server Enterprise Edition password policy configuration settings differ from the password policy configuration settings provided with the 5.x version of Directory Server. If your topology includes servers that run different versions of Directory Server, see "New Password Policy" in *Sun Java System Directory Server Enterprise Edition 6.3 Migration Guide* for information about how to migrate password policy settings.

# Password Synchronization With Windows

Identity Synchronization for Windows synchronizes user account information, including passwords, between Directory Server and Windows. Both Windows Active Directory and Windows NT are supported. Identity Synchronization for Windows helps build a scalable and security-enriched password synchronization solution for small, medium, and large enterprises.

This solution provides the following:

- Synchronization of account creation, modification, inactivation, and deletion between Active Directory, Windows NT, and Directory Server
- Integration with disparate and proprietary directory sources to synchronize native password changes

For more information about using Identity Synchronization for Windows in your deployment, see the *Sun Java System Identity Synchronization for Windows 6.0 Deployment Planning Guide*.

# Determining Encryption Methods

Encryption helps to protect data in transit, as well as stored data. This section describes the following methods of data encryption:



## Securing Connections With SSL

Security design involves more than an authentication scheme for identifying users and an access control scheme for protecting information. You must also protect the integrity of information between servers and client applications while it is being sent over the network.

To provide secure communications over the network, you can use both the LDAP and DSML protocols over the Secure Sockets Layer (SSL). When SSL is configured and activated, clients connect to a dedicated secure port where all communications are encrypted after the SSL connection is established. Directory Server and Directory Proxy Server also support the Start Transport Layer Security (Start TLS) control. Start TLS allows the client to initiate an encrypted connection over the standard LDAP port.

For an overview of SSL and TLS in Directory Server, see Chapter 2, "Directory Server Security," in *Sun Java System Directory Server Enterprise Edition 6.3 Reference*.

## Encrypting Stored Attributes

Attribute encryption concerns the protection of stored data. This section describes the attribute encryption functionality, and covers the following topics:



### What Is Attribute Encryption?

Directory Server Enterprise Edition provides various features to protect data at the access level, including password authentication, certificate-based authentication, SSL, and proxy authorization. However, the data stored in database files, backup files, and LDIF files must also be protected. The attribute encryption feature prevents users from accessing sensitive data while the data is stored.

Attribute encryption enables certain attributes to be stored in encrypted form. Attribute encryption is configured at the database level. Thus, after an attribute is encrypted, the attribute is encrypted in every entry in the database. Because attribute encryption occurs at the attribute level (not the entry level), the only way to encrypt an entire entry is to encrypt all of its attributes.

Attribute encryption also enables you to export data to another database in an encrypted format. The purpose of attribute encryption is to protect sensitive data only when the data is being stored or exported. Therefore, the encryption is always reversible. Encrypted attributes are decrypted when returned through search requests.

The following figure shows a user entry being added to the database, where attribute encryption has been configured to encrypt the `salary` attribute.

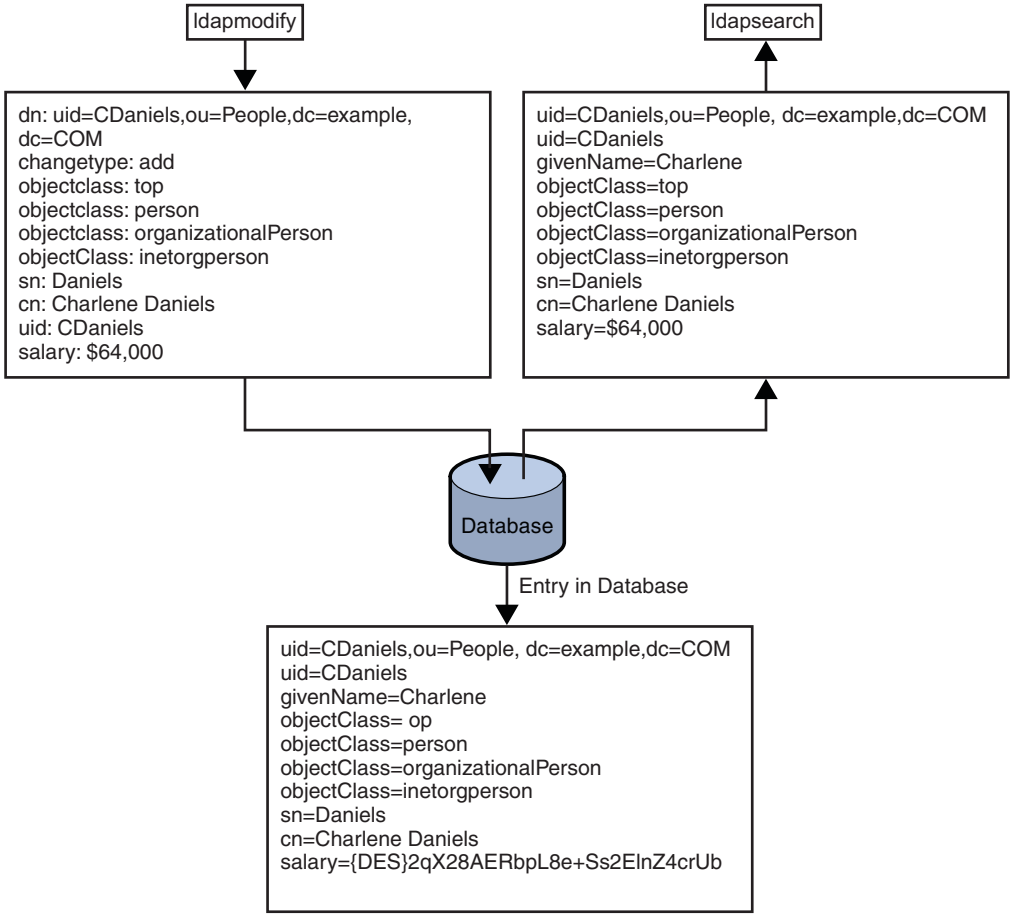



**FIGURE 7–1** Attribute Encryption Logic

## Attribute Encryption Implementation

The attribute encryption feature supports a wide range of encryption algorithms. Portability across different platforms is ensured. As an additional security measure, attribute encryption uses the private key of the server's SSL certificate to generate its own key. This key is then used to perform the encryption and decryption operations. To be able to encrypt attributes, a server

must be running over SSL. The SSL certificate and its private key are stored securely in the database and protected by a password. This key database password is required to authenticate to the server. The server assumes that whoever has access to this key database password is authorized to export decrypted data.

Note that attribute encryption only protects stored attributes. If you are replicating these attributes, replication must be configured over SSL to protect the attributes during transport.

If you use attribute encryption, you cannot use the binary copy feature to initialize one server from another server.

### Attribute Encryption and Performance

While attribute encryption offers increased data security, this feature does impact performance. Use attribute encryption only to encrypt particularly sensitive attributes.

Sensitive data can be accessed directly through index files. Thus, you must encrypt the index keys corresponding to the encrypted attributes, to ensure that the attributes are fully protected. Indexing already has a performance impact, without the added cost of encrypting index keys. Therefore, configure attribute encryption *before* data is imported or added to the database for the first time. This procedure ensures that encrypted attributes are indexed as such from the outset.

## Designing Access Control With ACIs

Access control enables you to specify that certain clients have access to particular information, while other clients do not. You implement access control using one or more access control lists (ACLs). ACLs consist of a series of access control instructions (ACIs) that either allow or deny permissions to specified entries and their attributes. Permissions include the ability to read, write, search, proxy, add, delete, compare, import and export.

By using an ACL, you can set permissions for the following:

- The entire directory
- A particular subtree of the directory
- Specific entries in the directory
- A specific set of entry attributes
- Any entry that matches a given LDAP search filter

In addition, you can set permissions for a specific user, for all users that belong to a group, or for all users of the directory. You can also define access for a network location, such as an IP address or a DNS name.

This section provides an overview of the Directory Server access control mechanism. For detailed information about configuring access control and ACIs, see Chapter 7, “Directory

Server Access Control," in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*. For information about the architecture of the access control mechanism, see "How Directory Server Provides Access Control" in *Sun Java System Directory Server Enterprise Edition 6.3 Reference*.

## Default ACIs

The default behavior of Directory Server is to deny access unless there is a specific ACI that grants access. therefore, if no ACIs are defined, all access to the server is denied.

When you install Directory Server or when you add a new suffix, several default ACIs are defined automatically in the root DSE. These ACIs can be modified to suit your security requirements.

For details on the default ACIs and how to modify them, see "How Directory Server Provides Access Control" in *Sun Java System Directory Server Enterprise Edition 6.3 Reference*.

## ACI Scope

Directory Server 6.x includes two major changes to ACI scope.

- **Ability to specify the scope of an ACI.** In Directory Server 5.x, you could not specify the scope of an ACI. ACIs automatically applied to the entry that contained the ACI and all of its subtree. Therefore, it was necessary to use *deny* ACIs in several cases. Deny ACIs can be difficult to manage, particularly when a deny ACI and an allow ACI apply to a single entry.

  In Directory Server 6.x, you can specify the scope of an ACI, that is, you can use *allow* ACIs to control access. Although, deny-based access control might sometimes be unavoidable or simpler to configure, the use of deny ACIs is generally discouraged. For information about how to specify the scope of an ACI, see Chapter 7, "Directory Server Access Control," in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.
- **Root ACIs now apply to the root entry and its entire subtree.** In Directory Server 5.x, ACIs located in the root DSE applied to the root entry only and not its children. ACIs placed in any other entry applied to the entry that contained the ACI and all of its subtree. In Directory Server Enterprise Edition ACIs placed in the root entry are treated like ACIs placed anywhere else.

  The new root ACIs have an obvious security advantage. Administrators no longer have to bind as the Directory Manager to perform certain operations. Administrators can now be forced to bind by using strong authentication such as SSL. When configuring ACIs that are intended to apply *only* to the root entry, the scope of the ACI must now specifically be set to `base`.

The change in ACI scope has implications for migration. If you are migrating to Directory Server 6.x from a 5.x version of Directory Server, see "Changes to ACIs" in *Sun Java System Directory Server Enterprise Edition 6.3 Migration Guide*.

# Obtaining Effective Rights Information

The access control model provided by Directory Server can grant access to users through many different mechanisms. However, this flexibility can make your security policy fairly complex. Several parameters can define the security context of a user, including IP address, machine name, time of day, and authentication method.

To simplify the security policy, Directory Server enables you to request and list the effective access rights that a given user has to specified directory entries and attributes. For more information, see "Viewing Effective Rights" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

# Tips on Using ACIs

The following tips can simplify your directory security model and improve directory performance:

- Minimize the number of ACIs in your directory, and use macro ACIs where possible.

  Although Directory Server can evaluate over 50,000 ACIs, managing a large number of ACI statements can be difficult. Excessive ACIs can also have a negative impact on memory consumption.

- Balance allow and deny permissions.

  The default rule is to deny access to any user who has not been specifically granted access. However, you can reduce the number of ACIs by using one ACI that allows access close to the root of the tree and using a small number of deny ACIs close to the leaf entries. This approach can prevent excessive allow ACIs close to the leaf entries.

- Identify the smallest set of attributes on any given ACI.

  If you allow or deny access to a subset of attributes on an object, determine whether the smallest list is the set of attributes that are allowed or the set of attributes that are denied. Then express your ACI so that you are managing the smallest list.

  For example, the people object class contains dozens of attributes. To allow a user to update just a few attributes, write your ACI so that it allows write access for just those few attributes. To allow a user to update all but one or two attributes, create the ACI so that it denies write access for those one or two attributes.

- Use LDAP search filters cautiously.

  Search filters do not directly name the object for which you are managing access. Search filters can therefore result in unexpected results especially as your directory becomes more complex. If you use search filters in ACIs, run an `ldapsearch` operation with the same filter. This action will ensure that you know what the results of the changes mean to your directory.

- Do not duplicate ACIs in different parts of your directory tree.

Look for overlapping ACIs. Imagine that you have an ACI at your directory root point that allows a group write access to the `commonName` and `givenName` attributes. Imagine also that you have another ACI that allows the same group write access to just the `commonName` attribute. In this scenario, consider reworking your ACIs so that only one attribute grants write access for the group.

As your directory grows more complicated, accidental overlapping of ACIs becomes increasingly common. If you avoid ACI overlap, security management becomes easier and the total number of ACIs in your directory is reduced.

- Name your ACIs.

  While naming ACIs is optional, giving each ACI a short, meaningful name makes managing your security model easier.

- Use standard attributes in user entries to determine access rights.

  As far as possible, use information that is already part of standard user entries to define access rights. If you must create special attributes, consider creating the attributes as part of a role or Class of Service (CoS) definition. For more information about roles and CoS, see Chapter 8, "Directory Server Groups and Roles," in *Sun Java System Directory Server Enterprise Edition 6.3 Reference* and Chapter 9, "Directory Server Class of Service," in *Sun Java System Directory Server Enterprise Edition 6.3 Reference*.

- Group ACIs as closely together as possible.

  Limit ACI placement to your directory root point and to major directory branch points. If you organize ACIs into groups, the total list of ACIs is easier to manage and the total number of ACIs can be kept to a minimum.

- Avoid using double negatives, such as deny write if the bind DN is not equal to `cn=Joe`.

  Although this syntax is acceptable to the server, the syntax can be confusing for an administrator.

# Designing Access Control With Connection Rules

Connection rules enable you to prevent selected clients from establishing connections to Directory Server. The purpose of connection rules is to prevent a denial-of-service attack caused by malicious or poorly designed clients that connect to Directory Server and flood the server with requests.

Connection rules are established at the TCP level by defining *TCP wrappers*. For more information about TCP wrappers, see "Client-Host Access Control Through TCP Wrapping" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

# Designing Access Control With Directory Proxy Server

Directory Proxy Server connection handlers provide a method of access control that enables you to classify incoming client connections. In this way, you can restrict the operations that can be performed based on how the connection has been classified.

You can use this functionality, for example, to restrict access to clients that connect from a specified IP address only. The following figure shows how you can use Directory Proxy Server connection handlers to deny write operations from specific IP addresses.

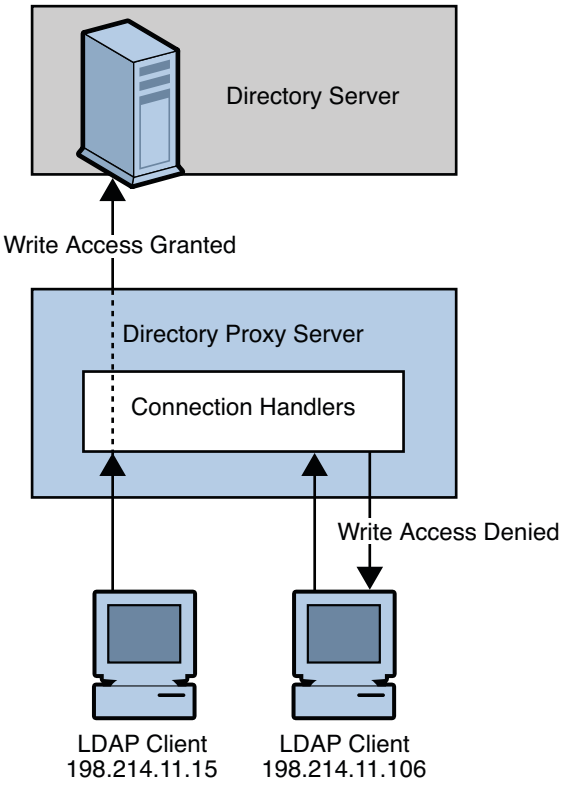



**FIGURE 7–2** Directory Proxy Server Connection Handler Logic

## How Connection Handlers Work

A connection handler consists of a list of criteria and a list of policies. Directory Proxy Server determines a connection's class membership by matching the origination attributes of the connection with the criteria of the class. When the connection has been matched to a class, Directory Proxy Server applies the policies that are contained in that class to the connection.

Connection handler criteria can include the following:

- Client physical address
- Domain name or host name
- Client DN pattern
- Authentication method
- SSL

The following policies can be associated with a connection handler:

- **Administrative limits policy.** Enables you to set certain limits on, for example, the number of open connections from clients of a specific class.
- **Content adaptation policy.** Enables you to restrict the kind of operations a connection can perform, for example, attribute renaming.
- **Data distribution policy.** Enables you to use a specific distribution scheme for a connection.

For more information about Directory Proxy Server connection handlers and how to set them up, see Chapter 20, "Connections Between Clients and Directory Proxy Server," in *Sun Java System Directory Server Enterprise Edition 6.3 Reference*.

# Grouping Entries Securely

Roles and CoS require special consideration with regard to security.

## Using Roles Securely

Not every role is suitable for use within a security context. When creating a role, consider how easily it can be assigned to and removed from an entry. Sometimes, users should be able to add themselves to or remove themselves from a role. However, in some security contexts such open roles are inappropriate. For more information, see "Directory Server Roles" in *Sun Java System Directory Server Enterprise Edition 6.3 Reference*.

## Using CoS Securely

Access control for reading applies to both the real attributes and the virtual attributes of an entry. A virtual attribute generated by the Class of Service (CoS) mechanism is read like a normal attribute. Virtual attributes should therefore be given read protection in the same way. However, to make the CoS value secure, you must protect all of the sources of information the CoS value uses: the definition entries, the template entries, and the target entries. The same is true for update operations. Write access to each source of information must be controlled to

protect the value that is generated from these sources. For more information, see Chapter 9, "Directory Server Class of Service," in *Sun Java System Directory Server Enterprise Edition 6.3 Reference*.

# Using Firewalls

Firewall technology is typically used to filter or block network traffic to and from an internal network. If LDAP requests are coming from outside a perimeter firewall, you need to specify what ports and protocols are allowed to pass through the firewall.

The ports and protocols that you specify depend on your directory architecture. As a general rule, the firewall must be configured to allow TCP and UDP connections on ports 389 and 636.

Host-based firewalls can be installed on the same server that is running Directory Server. The rules for host-based firewalls are similar to the rules for perimeter defense firewalls.

# Running as Non-Root

You can create and run server instances as a non-root user. By running server instances as a non-`root` user, you limit any potential damage that an exploit could cause. Furthermore, servers running as non-`root` users are subject to access control mechanisms on the operating system.

# Other Security Resources

For more information about designing a secure directory, see the following resources:

- Sun Developer Security Resources
  http://developers.sun.com/techtopics/security/index.html
- *Understanding and Deploying LDAP Directory Services.* T. Howes, M. Smith, G. Good, Macmillan Technical Publishing, 1999
- SecurityFocus.com http://www.securityfocus.com/
- Computer Emergency Response Team (CERT) Coordination Center http://www.cert.org/

◆ ◆ ◆ CHAPTER 8

# Identifying Administration and Monitoring Requirements

Directory Server Enterprise Edition administration has changed significantly since the 5.x version of Directory Server. These changes are described in detail in the *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

This chapter provides an overview of these changes and describes the administrative decisions that you must make in the planning phase of your deployment:



## Directory Server Enterprise Edition Administration Model

Directory Server Enterprise Edition gives the administrator more control over instance creation and administration. This control is achieved by using two new commands, `dsadm` and `dsconf`. These commands provide all the functionality previously supplied by the `directoryserver` command plus additional functionality.

The `dsadm` command enables the administrator to create, start, and stop a Directory Server instance. This command combines all operations that require file system access to the Directory Server instance. The command must be run on the machine that hosts the instance. It does not perform any operation that requires LDAP access to the instance or access to an agent.

In the new administration model, a Directory Server instance is no longer tied to a *ServerRoot*. Each Directory Server instance is a standalone directory that can be manipulated in the same manner as an ordinary standalone directory.

The `dsconf` command combines the administration operations that require write access to `cn=config`. The `dsconf` command is an LDAP client. It can only be executed on an active

Directory Server instance. The command can be run remotely, enabling administrators to configure multiple instances from a single remote machine.

Directory Proxy Server provides two comparable commands, `dpadm` and `dpconf`. The `dpadm` command enables the administrator to create, start, and stop a Directory Proxy Server instance. The `dpconf` command enables the administrator to configure Directory Proxy Server by using LDAP and to access the Directory Server configuration through Directory Proxy Server.

In addition to these command-line utilities, Directory Server Enterprise Edition is integrated into the Java Web Console. The Console enables Directory Server Enterprise Edition and other Sun products to be managed from a centralized user interface. Directory Service Control Center (DSCC) is a service of the Java Web Console that is specifically for managing Directory Servers and Directory Proxy Servers. DSCC provides the same functionality as the command-line utilities, as well as wizards that enable you to configure several servers simultaneously. In addition, DSCC provides a replication topology drawing tool that enables you to monitor replication topologies graphically. This tool simplifies replication monitoring by providing a real-time view of individual masters, hubs, and consumers, and the replication agreements between them.

## Remote Administration

The Directory Server Enterprise Edition administration model, described in the previous section, also enables remote administration of any Directory Server or Directory Proxy Server in the topology. Servers can be administered remotely using both the command-line utilities and the Java Web Console.

The `dsadm` and `dpadm` utilities cannot be run remotely. These utilities must be installed and run on the same machine as the server instance that is being administered. For details of the functionality provided with `dsadm` and `dpadm`, see the `dsadm`(1M) and `dpadm`(1M) man pages.

The `dsconf` and `dpconf` utilities can be run remotely. For details of the functionality provided with `dsconf` and `dpconf`, see the `dsconf`(1M) and `dpconf`(1M) man pages.

The following figure illustrates how the new administration model facilitates remote administration. This illustration shows that the console and configuration commands can be installed and run remotely from the Directory Server and Directory Proxy Server instances. The administration commands must be run locally to the instances.

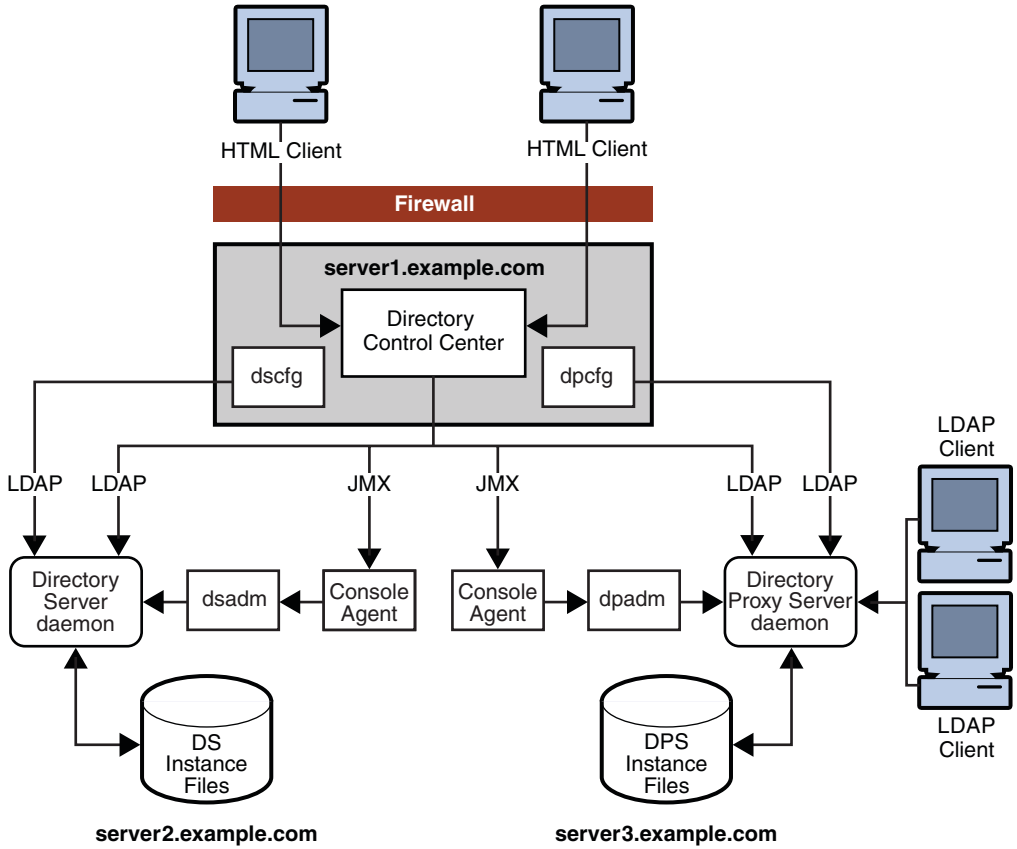



**FIGURE 8–1** Directory Server Enterprise Edition Administration Model

# Designing Backup and Restore Policies

In any failure situation that involves data corruption or data loss, it is imperative that you have a recent backup of your data. Avoid reinitializing servers from other servers where possible. For information about how to back up data, seeChapter 9, "Directory Server Backup and Restore," in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

This section provides an overview of what to consider when planning a backup and recovery strategy.

# High-Level Backup and Recovery Principles

Apply the following high-level principles when designing a backup strategy:

- Identify the data that must be backed up.

  For Directory Server Enterprise Edition this data includes the following:

  - Shared binaries and plug-ins
  - Certificate database files
  - Configuration files
  - Log files and the change log database
  - Schema files
  - User data
- Ensure that your backup and recovery strategy includes the hardware, operating system, and software components.
- Decide whether you will keep binary backups or LDIF backups.

  A general recommendation is that you keep both. For more information, see "Choosing a Backup Method" on page 132 and "Choosing a Restoration Method" on page 136.
- Build automation around backup and recovery tools, and ensure that automatic scripts are maintained.

  This strategy avoids unnecessary delays if you have to restore from a backup in an emergency.
- Determine a retention and rotation strategy.

  This strategy includes how often you perform backups and how long you keep them. When determining retention and rotation of backups, be aware of the *purge delay* and its impact on backups in a replicated topology. As modifications occur on a supplier, changes are recorded in the change log. Without a method of emptying the change log, its size would continue to increase until the change log consumed all available disk space. By default, changes are purged every seven days. This period is known as the purge delay. When a change has been purged, the change can no longer be replicated. For this reason, make sure that databases are backed up at least as often as the purge delay.
- Use the backup and recovery tools provided with Directory Server Enterprise Edition rather than merely performing a system backup and recovery.

# Choosing a Backup Method

Directory Server Enterprise Edition provides two methods of backing up data: binary backup and backup to an LDIF file. Both of these methods have advantages and limitations, and knowing how to use each method will assist you in planning an effective backup strategy.

## Binary Backup

Binary backup produces a copy of the database files, and is performed at the file-system level. The output of a binary backup is a set of binary files containing all entries, indexes, the change log, and the transaction log. A binary backup does not contain configuration data.

Binary backup is performed using one of the following commands:

- `dsadm backup` must be run offline, that is, when the Directory Server instance is stopped. The command must be run on the local server containing the Directory Server instance.
- `dsconf backup` can be run online and remote to the Directory Server instance.

Binary backup has the following advantages:

- All suffixes can be backed up at the same time.
- Binary backup is significantly faster than a backup to LDIF.
- The replication change log is backed up.

Binary backup has one limitation. Restoration from a binary backup can be performed only on a server with an *identical* configuration.

This limitation implies the following:

- Both machines must use the same hardware and the same operating system, including any service packs or patches.
- Both machines must have the same version of Directory Server installed, including binary format (32 bits or 64 bits), service packs and patch levels.
- Both servers must have the same directory tree that is divided into the same suffixes. The database files for all suffixes must be copied together Individual suffixes cannot be copied.
- Each suffix must have the same indexes configured on both servers, including virtual list view (VLV) indexes. The database files for the suffixes must have the same name.
- Each server must have the same suffixes configured as replicas. If fractional replication is configured, fractional replication must be configured identically on all master servers.
- Attribute encryption must not be used on either server.

At a minimum, you need to perform a regular binary backup on each set of coherent machines. Coherent machines are machines that have an identical configuration, as defined previously.

---

**Note –** Because restoration from a local backup is easier, perform a binary backup on each server.

---

These abbreviations are used in the remaining diagrams in this chapter:

M = master replica

RA = replication agreement

The following figure assumes that M1 and M2 have an identical configuration and that M3 and M4 have an identical configuration. In this scenario, a binary backup would be performed on M1 and on M3. In the case of failure, M1 or M2 could be restored from the binary backup of M1 (db1). M3 or M4 could be restored from the binary backup of M3 (db2). M1 and M2 could not be restored from the binary backup of M3. M3 and M4 could not be restored from the binary backup of M1.

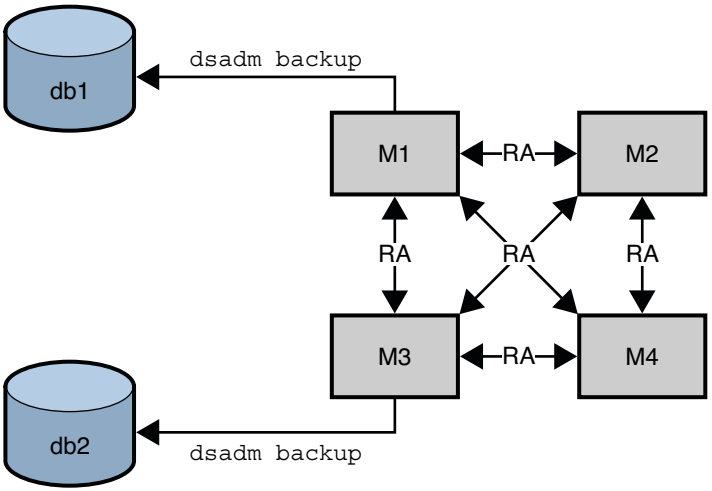



**FIGURE 8–2** Offline Binary Backup

For details on how to use the binary backup commands, see "Binary Backup" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

## Backup to LDIF

Backup to LDIF is performed at the suffix level. The output of a backup to LDIF is a formatted LDIF file, which is a copy of the data contained in the suffix. As such, this process takes longer than a binary backup.

Backup to LDIF is performed using one of the following commands:

- `dsadm export` must be run offline, that is, when the Directory Server instance is stopped. This command must be run on the local server containing the Directory Server instance.
- `dsconf export` can be run online and remote to the Directory Server instance.

**Note –** Replication information is backed up unless you use the `-Q` option when running these commands.

The `dse.ldif` configuration file is not backed up in a backup to LDIF. To enable you to restore a previous configuration, back this file up manually.

Backup to LDIF has the following advantages:

- Backup to LDIF can be performed from any server, regardless of its configuration.
- Restoration from an LDIF backup can be performed on any server, regardless of its configuration.

Backup to LDIF has one limitation. In situations where rapid backup and restoration are required, backup to LDIF might take too long to be viable.

You need to perform a regular backup by using backup to LDIF for each replicated suffix, on a single master in your topology.

In the following figure, `dsadm export` is performed for each replicated suffix, on one master only (M1).

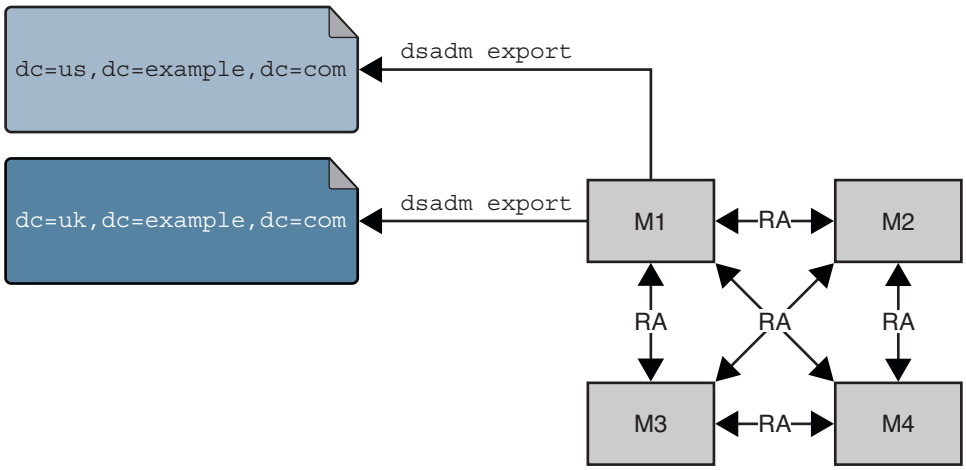



**FIGURE 8–3** Offline Backup to LDIF

For information about how to use the backup to LDIF commands, see "Backing Up to LDIF" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

# Choosing a Restoration Method

Directory Server Enterprise Edition provides two methods of restoring data: binary restore and restoration from an LDIF file. As with the backup methods, both of these methods have advantages and limitations.

## Binary Restore

Binary restore copies data at the database level. Binary restore is performed using one of the following commands:

- `dsadm restore` must be run offline, that is, when the Directory Server instance is stopped. This command must be run on the local server containing the Directory Server instance.
- `dsconf restore` can be run online and remote to the Directory Server instance.

Binary restore has the following advantages:

- All suffixes can be restored at the same time.
- The replication change log is restored.
- Binary restore is significantly faster than restoring from an LDIF file.

Binary restore has the following limitations:

- Restoration can be performed only on a server with an identical configuration, as defined in "Binary Backup" on page 133. For more information about restoring data with the binary restore feature, see "Binary Restore" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.
- If you are not aware that your database was corrupt when you performed the binary backup, you risk restoring a corrupt database. Binary backup creates an exact copy of the database.

Binary restore is the preferred restoration method if the machines have an identical configuration and time is a major consideration.

The following figure assumes that M1 and M2 have an identical configuration and that M3 and M4 have an identical configuration. In this scenario, M1 or M2 can be restored from the binary backup of M1 (db1). M3 or M4 can be restored from the binary backup of M3 (db2).

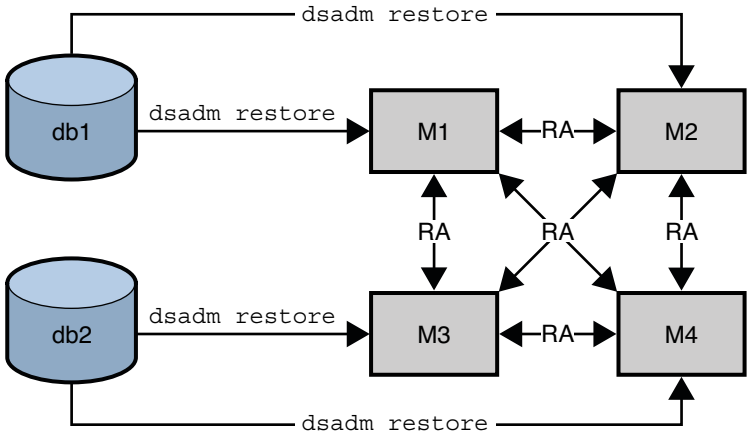



**FIGURE 8–4** Offline Binary Restore

## Restoration From LDIF

Restoration from an LDIF file is performed at the suffix level. As such, this process takes longer than a binary restore. Restoration from LDIF can be performed using one of the following commands:

- `dsadm import` must be run offline, that is, when the Directory Server instance is stopped. This command must be run on the local server containing the Directory Server instance.
- `dsconf import` can be run online and remote to the Directory Server instance.

Restoration from an LDIF file has the following advantages:

- This command can be performed on any server, regardless of its configuration.
- A single LDIF file can be used to deploy an entire directory service, regardless of its replication topology. This functionality is particularly useful for the dynamic expansion and contraction of a directory service according to anticipated business needs.

Restoration from an LDIF file has one limitation. In situations where rapid restoration is required, this method might take too long to be viable. For more information about restoring data from an LDIF file, see "Importing Data From an LDIF File" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

In the following figure, `dsadmin import` is performed for each replicated suffix, on one master only (M1).

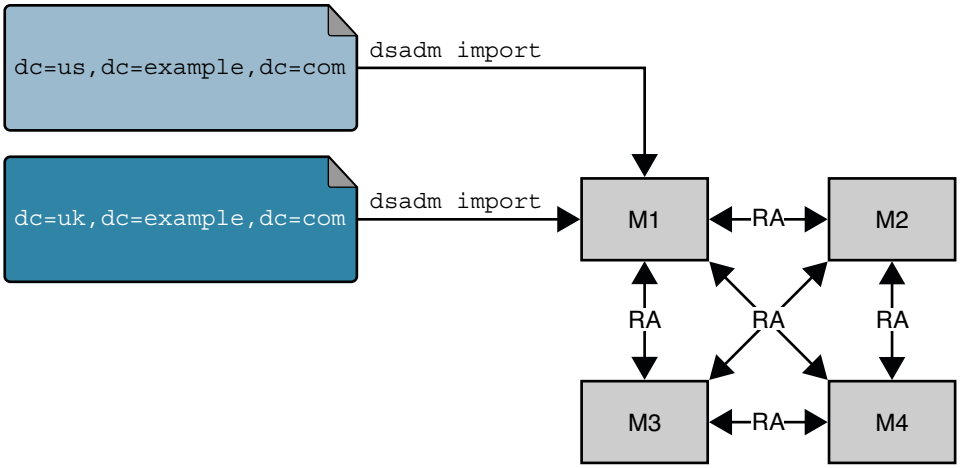



**FIGURE 8–5** Offline Restoration From LDIF

# Designing a Logging Strategy

Logging is managed and configured at the individual server level. While logging is enabled by default, it can be reconfigured or disabled according to the requirements of your deployment. Designing a logging strategy assists with planning hardware requirements. For more information, see "Hardware Sizing For Directory Server" on page 74.

This section describes the logging facility of Directory Server Enterprise Edition.

## Defining Logging Policies

Each Directory Server in a topology stores logging information in three files:

- **Access log.** Lists the clients that connect to the server and the operations requested.
- **Error log.** Provides information about server errors.
- **Audit log.** Gives details about modifications to suffixes and to the configuration.

Each Directory Proxy Server in a topology stores logging information in two files:

- **Access log.** Lists the clients that connect to Directory Proxy Server and the operations requested.
- **Error log.** Contains server error messages.

You can manage the log files for both Directory Server and Directory Proxy Server in these ways:

- Defining log file creation policies
- Defining log file deletion policies
- Manually creating and deleting log files
- Defining log file permissions

## Defining Log File Creation Policies

A log file creation policy enables you to periodically archive the current log and start a new log file. Log file creation policies can be defined for Directory Server and Directory Proxy Server from the Directory Control Center or using the command-line utilities.

When defining a log file creation policy, consider the following:

- How many logs do you want to keep?

  When this number of logs is reached, the oldest log file in the folder is deleted before a new log is created. If this value is set to `1`, the logs are not rotated and grow indefinitely.

- What is the maximum size, in Megabytes, for each log file?

  When a log file reaches this maximum size or the maximum age defined in the next item, the file is archived. A new log file is started.

- How often should the current log file be archived?

  The default is every day.

- At what time of day should log files be rotated?

  Time-based rotation makes operations like log analysis and trending easier, because each log file covers the same time period.

Log file rotation can also be based on a combination of criteria. For example, you can specify that logs be rotated at 23h30 *only* if the file size is greater than 10 Megabytes.

For details on how to set up a log file creation policy, see "Configuring Logs for Directory Server" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

## Defining Log File Deletion Policies

A log file deletion policy enables you to automatically delete old archived logs. Log file deletion policies can be defined for Directory Server and Directory Proxy Server from the Directory Service Control Center or using the command-line utilities. A log file deletion policy is not applied unless you have defined a log file creation policy. Log file deletion will not work if you have just one log file. The server evaluates and applies the log file deletion policy at the time of log rotation.

When defining a log file deletion policy, consider the following:

- What is the maximum size of the combined archived logs?

  When the maximum size is reached, the oldest archived log is automatically deleted.
- What is the minimum free disk space that should be available?

  When the free disk space reaches this minimum value, the oldest archived log is automatically deleted.
- What is the maximum age of log files?

  When a log file reaches this maximum age, the log file is automatically deleted.

For details on how to set up a log file deletion policy, see "Configuring Logs for Directory Server" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

### Manually Creating and Deleting Log Files

Manual file rotation and forced log rotation do not apply to Directory Proxy Server.

If you do not want to define automatic creation and deletion policies for Directory Server, you can create and delete log files manually. In addition, Directory Server provides a task that enables you to rotate any log immediately, regardless of the defined creation policy. This functionality might be useful if, for example, an event occurs that needs to be examined in more detail. The immediate rotation function causes the server to create a new log file. The previous file can therefore be examined without the server appending logs to this file.

For information about how to rotate logs manually and how to force log rotation, see "Rotating Directory Server Logs Manually" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

### Defining Permissions on Log Files

In 5.x version of Directory Server, log files could only be read by the directory manager. Directory Server Enterprise Edition enables server administrators to define the permissions with which log files are created. For information about how to define log file permissions, see "Configuring Logs for Directory Server" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

# Designing a Monitoring Strategy

An effective monitoring and event management strategy is crucial to a successful deployment. Such a strategy defines which events should be monitored, which tools to use, and what action to take should an event occur. If you have a plan for commonplace events, possible outages and reduced levels of service can be prevented. This strategy improves the availability and quality of service of your directory.

To design a monitoring strategy, do the following:

- Select the appropriate monitoring tools. See "Monitoring Tools Provided With Directory Server Enterprise Edition" on page 141.
- Identify the key areas to be monitored in the directory architecture.

  These areas are frequently the same as the sizing and tuning attributes. See "Identifying Monitoring Areas" on page 142.
- Define what triggers an event or alarm condition when monitoring the key performance measure.

  This strategy implies defining an acceptable level of performance or operation for each performance measure.
- Determine what action should be taken when an alarm condition occurs.

# Monitoring Tools Provided With Directory Server Enterprise Edition

This section provides a summary of the monitoring tools that are available in Directory Server Enterprise Edition as well as additional tools that can be used to monitor server activity.

The monitoring areas described in "Identifying Monitoring Areas" on page 142 can be monitored using one or more of these tools.

- **Command-line tools.** Include operating system-specific tools to monitor performance such as disk usage, LDAP tools such as `ldapsearch` to collect server statistics stored in the directory, third-party tools, or custom shell or Perl scripts.
- **Directory Server and Directory Proxy Server logs.** Include the access, audit, and error logs. These logs can be monitored manually or parsed using custom scripts to extract monitoring information that is relevant to your deployment. The Directory Server Resource Kit provides a log analyzer tool, `logconv`, that enables you to analyze the access logs. The log analyzer tool extracts usage statistics and counts the occurrences of significant events. For more information about this tool, see `logconv`(1). For information about viewing and configuring log files, see Chapter 15, "Directory Server Logging," in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.
- **Directory Service Control Center (DSCC).** Is a graphical user interface that enables you to monitor directory operations in real time. DSCC provides general server information, including a resource summary, current resource usage, connection status, and global database cache information. It also provides general database information, such as the database type, status, and entry cache statistics. Cache information and information relative to each index file within the database is also provided. In addition, DSCC provides information relative to the connections and the operations performed on each chained suffix.

- **Replication monitoring tools.** Include the command-line tools, `repldisc`, `insync` and `entrycmp`.

  These tools enable you to do the following:

  - Monitor the state of synchronization between a master replica and one or more consumer replicas
  - Compare the same entry on two or more different replicas so that you can assess replication status
  - Depict your complete replication topology, which is particularly beneficial when dealing with complex directory deployments

  For more information, see `repldisc`(1), `insync`(1) and `entrycmp`(1).

  You can also monitor replication status by using the DCC. For more information about monitoring replication, see "Getting Replication Status" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.
- **Simple Network Management Protocol (SNMP).** Is the standard mechanism for global network control and monitoring, and enables network administrators to centralize network monitoring activity.

  For information about monitoring using an SNMP agent, see Chapter 16, "Directory Server Monitoring," in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.
- **Java ES Monitoring Framework.** Enables monitoring of performance and other statistics through JMX. For more information, see "Directory Server and CMM/JMX" in *Sun Java System Directory Server Enterprise Edition 6.3 Reference*.

# Identifying Monitoring Areas

What you monitor, and to what extent, depends on your specific deployment. In general, however, include the following elements in your monitoring strategy:

- **Server activity** such as resource usage, server status, and connection information
- **Database activity** such as cache, transactions, locks, and log information
- **Disk status** including available disk space and threshold information
- **Replication activity** including status (whether or not replication is running), and the state of synchronization
- **Indexing efficiency** including unindexed searches, search filters, and frequently used indexes
- **Security status** including failed bind attempts, open connections, and effective rights

# Data Administration With Directory Editor

The Directory Editor component of Directory Server Enterprise Edition is a Java web application that enables you to manage directory data by using a web browser. Directory Editor provides all users with remote access to directory data without having to install any client software.

Directory Editor offers the following functionality:

- Enables administrators and end users to create and edit directory users, groups, and containers.
- Supports several concurrent users, depending on the application server and underlying hardware.
- Supports large enterprise directory installations.
- Enables customization, branding, and embedding of the interface.

  Customization dynamically adapts to the Directory Server schema.
- Enables customization through the configuration of forms, rather than by direct programming.
- Supports SSL-encrypted transmissions between the client browser and Directory Server.
- Limits access to menus and functions, based on roles.

  Roles are scanned to match group names. Roles have access to certain *capabilities*, which are high-level actions such as Browse, Configure, Debug, Edit, Create, and Search.
- Limits access to the data based on the existing ACIs in Directory Server. It is not necessary to define ACIs that are specific to Directory Editor.
- Enables paged display of large volumes of data, based on the virtual list view (VLV) index.

For details on installing, configuring, and using Directory Editor, see the Directory Editor Documentation Collection (`http://docs.sun.com/app/docs/coll/DirEdit_05q1`).

PART III

# Logical Design

A logical architecture identifies the components of a Directory Server Enterprise Edition deployment, and shows interrelationships between the components. Typically, use cases developed during the technical requirements phase indicate which components the deployment requires. However, the required components can often be derived directly from the business requirements.

This part provides sample logical architectures that are based on typical Directory Server Enterprise Edition deployment scenarios. The information in this part flows from a basic, single-server deployment to more complex deployments that span multiple data centers. The architectures discussed in the later chapters of this part build on the simpler architectures discussed in the earlier chapters.

This part includes the following chapters:

- Chapter 9, "Designing a Basic Deployment," describes a basic Directory Server Enterprise Edition deployment.
- Chapter 10, "Designing a Scaled Deployment," describes a deployment scaled to meet additional service requirements.
- Chapter 11, "Designing a Global Deployment," covers deployment considerations for deployments across multiple data centers.
- Chapter 12, "Designing a Highly Available Deployment," describes deployments designed to meet availability requirements.

◆ ◆ ◆ CHAPTER 9

# Designing a Basic Deployment

In the simplest Directory Server Enterprise Edition deployment, your directory service requirements can be fulfilled by a single Directory Server, installed on one machine, in a single data center. Such a scenario might occur in a small organization or if, you are running Directory Server for demonstration or evaluation purposes. Note that the technical requirements discussed in the previous chapters apply equally to all deployments.

This chapter describes a basic deployment, involving a single Directory Server. The chapter covers the following topics:



## Basic Deployment Architecture

A basic Directory Server Enterprise Edition deployment includes the following elements:

- Directory Server instance files
- Directory Server daemon
- `dsadm` and `dsconf` command-line utilities
- Directory Service Control Center (DSCC), if GUI access is required
- Console Agent, if DSCC is used

These elements can all be installed on a single machine. The following figure illustrates the high-level architecture of a basic Directory Server Enterprise Edition deployment.

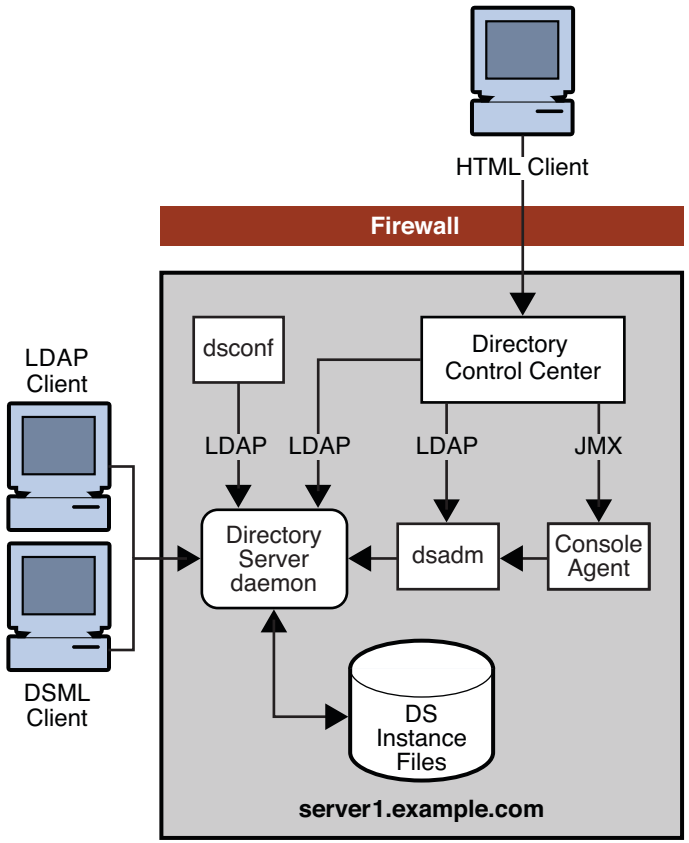



**FIGURE 9–1** Basic Directory Server Enterprise Edition Architecture on a Single Machine

In this scenario, internal LDAP and DSML clients can be configured to access Directory Server directly. External HTML clients can be configured to access DSCC over a firewall.

Although all of the components described previously can be installed on a single machine, this is unlikely in a real deployment. A more typical scenario would be the installation of DSCC and the `dsconf` command-line utility on separate remote machines. All Directory Server hosts could then be configured remotely from these machines. The following figure illustrates this more typical scenario.

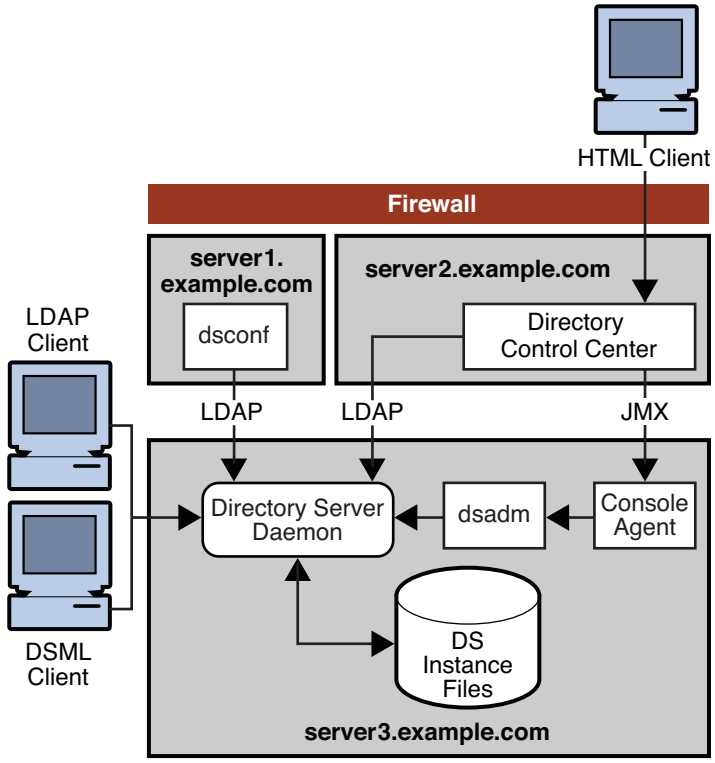



**FIGURE 9–2** Basic Directory Server Enterprise Edition Architecture With Remote Directory Service Control Center

The Directory Server instance stores server and application configuration settings, as well as user information. Typically, server and application configuration information is stored in one suffix of Directory Server while user and group entries are stored in another suffix. A *suffix* refers to the name of the entry in the directory tree, below which data is stored.

Directory Service Control Center (DSCC) is a centralized, web-based user interface for all servers, and is the Directory component of Java Web Console. DSCC locates all servers and applications that are registered with it. DSCC displays the servers in a graphical user interface, where you can manage and configure the servers. The Directory Service Control Center might not be required in a small deployment because all functionality is also provided through a command-line interface.

In the chapters that follow, it is assumed that the Directory Service Control Center is installed on a separate machine. This aspect of the topology is not referred to again in the remaining chapters.

# Basic Deployment Setup

Complete installation information is provided in the *Sun Java System Directory Server Enterprise Edition 6.3 Installation Guide*. The purpose of this section is to provide a clear picture of the elements that make up a basic deployment and how these elements work together.

This section lists the main tasks for setting up the basic deployment described in the previous section.

- Install the required shared components, including the security packages.
- Install Directory Server, the Console Agent, and the command-line interface.
- If you want to manage the server by using the command-line utilities, do the following:
  - Create and start a standalone Directory Server instance by using the `dsadm` command.
  - Create and configure a suffix in the new instance, by using the `dsconf` command.
- If you want to manage the server through a graphical user interface, do the following:
  - Initialize the Directory Service Control Center.
  - Create a Directory Server instance by using the Directory Service Control Center.
  - Create and configure a suffix in the new instance by using the Directory Service Control Center.

# Improving Performance in a Basic Deployment

In even the most basic deployment, you might want to tune Directory Server to improve performance in specific areas. The following sections describe basic tuning strategies that can be applied to a simple single-server deployment. These strategies can be applied to each server in larger, more complex deployments, for improved performance across the topology.

## Using Indexing to Speed Up Searches

Indexes speed up searches by effectively reducing the number of entries a search has to check to find a match. An index contains a list of values. Each value is associated with a list of entry identifiers. Directory Server can look up entries quickly by using the lists of entry identifiers in indexes. Without an index to manage a list of entries, Directory Server must check every entry in a suffix to find matches for a search.

Directory Server processes each search request as follows:

1. Directory Server receives a search request from a client.
2. Directory Server examines the request to confirm that the search can be processed.

   If Directory Server cannot perform the search, it returns an error to the client and might refer the search to another instance of Directory Server.
3. Directory Server determines whether it manages one or more indexes that are appropriate to the search.
   - If Directory Server manages indexes that are appropriate to the search, the server looks in all of the appropriate indexes for candidate entries. A candidate entry is an entry that might be a match for the search request.
   - If Directory Server does not manage an index appropriate to the search, the server generates the set of candidate entries by checking all of the entries in the database.

     When Directory Server cannot use indexes, this process consumes more time and system resources.
4. Directory Server examines each candidate entry to determine whether the entry matches the search criteria.
5. Directory Server returns matching entries to the client application as it finds the entries.

You can optimize search performance by doing the following:

- Preventing Directory Server from performing searches on non-indexed entries
- Ensuring that cache sizes are appropriately tuned
- Limiting the length of an index

For a comprehensive overview of how indexes work, see Chapter 6, "Directory Server Indexing," in *Sun Java System Directory Server Enterprise Edition 6.3 Reference*. For information about defining indexes, see Chapter 13, "Directory Server Indexing," in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

## Optimizing Cache for Search Performance

For improved search performance, cache as much directory data as possible in memory. By preventing the directory from reading information from disk, you limit the disk I/O bottleneck. Different possibilities exist for doing this, depending on the size of your directory tree, the amount of memory available, and the hardware used. Depending on the deployment, you might choose to allocate more or less memory to entry and database caches to optimize search performance. You might alternatively choose to distribute searches across Directory Server consumers on different servers.

Consider the following scenarios:



## All Entries and Indexes Fit Into Memory

In the optimum case, the database cache and the entry cache fit into the physical memory available. The entry caches are large enough to hold all entries in the directory. The database cache is large enough to hold all indexes and entries. In this case, searches find everything in cache. Directory Server never has to go to file system cache or to disk to retrieve entries.

Ensure that database cache can contain all database indexes, even after updates and growth. When space runs out in the database cache for indexes, Directory Server must read indexes from disk for every search request, severely impacting throughput. You can monitor paging and cache activity with DSCC or through the command line.

Appropriate cache sizes must be determined through empirical testing with representative data. In general, the database cache size can be calculated as `(total size of database files) x 1.2`. Start by allocating a large amount of memory for the caches. Then exercise and monitor Directory Server to observe the result, repeating the process as necessary. Entry caches in particular might use *much more* memory than you allocate to these caches.

## Sufficient Memory For 32-Bit Directory Server

Imagine a system with sufficient memory to hold all data in entry and database caches, but no support for a 64-bit Directory Server process. If hardware constraints prevent you from deploying Directory Server on a Solaris system with 64-bit support, size caches appropriately with respect to memory limitations for 32-bit processes. Then leave the remaining memory to the file system cache.

As a starting point when benchmarking performance, size the entry cache to hold as many entries as possible. Size the database cache relatively small such as 100 Mbytes without completely minimizing it, but letting file system cache hold the database pages.

---

**Note –** File system cache is shared with other processes on the system, especially file-based operations. Thus, controlling file system cache is more difficult than controlling other caches, particularly on systems that are not dedicated to Directory Server.

The system might reallocate file system cache to other processes.

---

Avoid online import in this situation because import cache is associated with the Directory Server process.

### Insufficient Memory

Imagine a system with insufficient memory to hold all data in entry and database caches. In this case, avoid causing combined entry and database cache sizes to exceed the available physical memory. This might result in heavy virtual memory paging that could bring the system to a virtual halt.

For small systems, start benchmarking by devoting available memory to entry cache and database caches, with sizes no less than 100 Mbytes each. Try disabling the file system cache by mounting Solaris UFS file systems with the `-o forcedirectio` option of the `mount_ufs` command. For more information, see the `mount_ufs(1M)` man page. Disabling file system cache can prevent the file system cache from using memory needed by Directory Server.

For large Directory Servers running on large machines, maximize the file system cache and reduce the database cache. Verify and correct assumptions through empirical testing.

## Optimizing Cache for Write Performance

In addition to planning a deployment for write scalability from the outset, provide enough memory for the database cache to handle updates in memory. Also, minimize disk activity. You can monitor the effectiveness of the database cache by reading the hit ratio in the Directory Service Control Center.

After Directory Server has run for some time, the caches should contain enough entries and indexes that disk reads are no longer necessary. Updates should affect the database cache in memory, with data from the large database cache in memory being flushed only infrequently.

Flushing data to disk during a checkpoint can be a bottleneck. The larger the database cache size, the larger the bottleneck. Storing the database on a separate RAID system, such as a Sun StorEdge™ disk array, can help improve update performance. You can use utilities such as `iostat` on Solaris systems to isolate potential I/O bottlenecks. For more information, see the `iostat(1M)` man page.

The following table shows database and log placement recommendations for systems with 2, 3, and 4 disks.

**TABLE 9–1** Isolating Databases and Logs on Different Disks

| Disks Available | Recommendations |
|---|---|
| 2 | ■ Place the Directory Server database on one disk.<br>■ Place the transaction log, the access, audit, and error logs and the retro change log on the other disk. |

**TABLE 9–1** Isolating Databases and Logs on Different Disks *(Continued)*

| Disks Available | Recommendations |
|---|---|
| 3 | ■ Place the Directory Server database on one disk.<br>■ Place the transaction log on the second disk.<br>■ Place the access, audit, and error logs and the retro change log on the third disk. |
| 4 | ■ Place the Directory Server database on one disk.<br>■ Place the transaction log on the second disk.<br>■ Place the access, audit, and error logs on the third disk.<br>■ Place the retro change log on the fourth disk. |

◆ ◆ ◆ CHAPTER 10

# Designing a Scaled Deployment

The basic deployment described in Chapter 9, "Designing a Basic Deployment," assumes that a single Directory Server is enough to satisfy the read and write requirements of your organization. Organizations that have large read or write requirements, that is, several clients attempting to access directory data simultaneously, need to use a scaled deployment.

Generally, the number of searches a Directory Server instance can perform per second is directly related to the number and speed of the server's CPUs, provided there is sufficient memory to cache all data. Horizontal read scalability can be achieved by spreading the load across more than one server. This usually means providing additional copies of the data so that clients can read the data from more than one source.

Write operations do not scale horizontally because a write operation to a master server results in a write operation to every replica. The only way to scale write operations horizontally is to split the directory data among multiple databases and place those databases on different servers.

This chapter describes the different ways of scaling a Directory Server Enterprise Edition deployment to handle more reads and writes. The chapter covers the following topics:



## Using Load Balancing for Read Scalability

Load balancing increases performance by spreading the read load across multiple servers. Load balancing can be achieved using replication, Directory Proxy Server, or a combination of the two.

# Using Replication for Load Balancing

Replication is the mechanism that automatically copies directory data and changes from one directory server to another directory server. With replication, you can copy a directory tree or subtree that is stored in its own suffix between servers.

**Note –** You cannot copy the configuration or monitoring information subtrees.

By replicating directory data across servers, you can reduce the access load on a single machine, improving server response time and providing read scalability. Replicating directory entries to a location close to your users also improves directory response time. Replication is generally *not* a solution for write scalability.

## Basic Replication Concepts

The replication mechanism is described in detail in Chapter 4, "Directory Server Replication," in *Sun Java System Directory Server Enterprise Edition 6.3 Reference*. The following section provides basic information that you need to understand before reviewing the sample topologies described later in this chapter.

### Master, Consumer, and Hub Replicas

A database that participates in replication is defined as a *replica*.

Directory Server distinguishes between three kinds of replicas:

- **Master or read-write replica.** A read-write database that contains a master copy of the directory data. A master replica can process update requests from directory clients. A topology that contains more than one master is called a *multi-master* topology.
- **Consumer replica.** A read-only database that contains a copy of the information in the master replica. A consumer replica can process search requests from directory clients but refers update requests to master replicas.
- **Hub replica.** A read-only database (like a consumer replica) that is stored on a Directory Server that *supplies* one or more consumer replicas.

The following figure illustrates the role of each of these replicas in a replication topology.

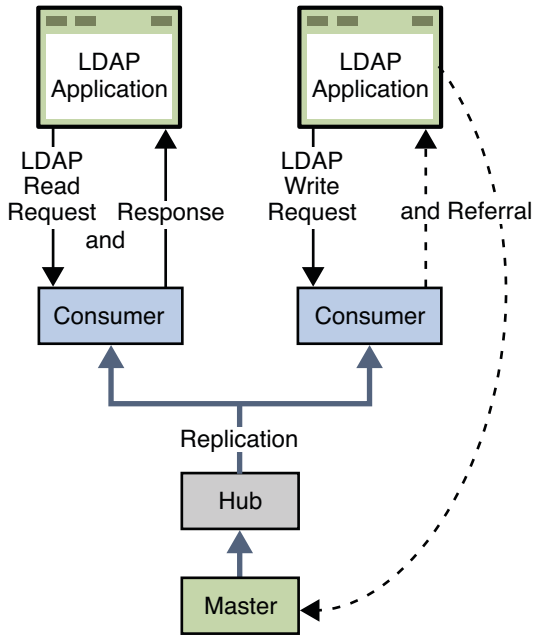



**FIGURE 10–1** Role of Replicas in a Replication Topology

**Note –** The previous figure is for illustration purposes only and is not necessarily a recommended topology. Directory Server6.x supports an unlimited number of masters in a multi-master topology. A master-only topology is recommended in most cases.

## Suppliers and Consumers

A Directory Server that replicates to other servers is called a *supplier*. A Directory Server that is updated by other servers is called a *consumer*. The supplier replays all updates on the consumer through specially designed LDAP v3 extended operations. In terms of performance, a supplier is therefore likely to be a demanding client application for the consumer.

A server can be both a supplier and a consumer, as in the following situations:

- In multi-master replication, a master replica is mastered on two different Directory Servers. Each server acts as a supplier and a consumer of the other server.
- When the server contains a hub replica, the server receives updates from a supplier and replicates the changes to consumers.

A server that plays the role of a consumer only is called a *dedicated consumer*.

For a *master replica*, the server must do the following:

- Respond to update requests from directory clients
- Maintain historical information and a change log
- Initiate replication to consumers

The server that contains the master replica is responsible for recording any changes made to the master replica and for replicating these changes to consumers.

For a *hub replica*, the server must do the following:

- Respond to read requests
- Refer update requests to the servers that contain a master replica
- Maintain historical information and a change log
- Initiate replication to consumers

For a *consumer replica*, the server must do the following:

- Respond to read requests
- Maintain historical information
- Refer update requests to the servers that contain a master replica

## Multi-Master Replication

In a multi-master replication configuration, data can be updated simultaneously in different locations. Each master maintains a change log for its replica. The changes that occur on each master are replicated to the other servers.

Multi-master configurations have the following advantages:

- Automatic write failover occurs when one master is inaccessible.
- Updates can be made on a local master in a geographically distributed environment.

Multi-master replication uses a loose consistency replication model. This means that the same entries may be modified simultaneously on different servers. When updates are sent between the two servers, any conflicting changes must be resolved. Various attributes of a WAN, such as latency, can increase the chance of replication conflicts. Conflict resolution generally occurs automatically. A number of conflict rules determine which change takes precedence. In some cases conflicts must be resolved manually. For more information, see "Solving Common Replication Conflicts" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

The number of masters that are supported in a multi-master topology is theoretically unlimited. The number of consumers and hubs is also theoretically unlimited. However, the number of consumers to which a single supplier can replicate depends on the capacity of the supplier server. You can use the SLAMD Distributed Load Generation Engine (SLAMD) to assess the capacity of the supplier server. For information about SLAMD, and to download the SLAMD software, see http://www.slamd.com (`http://www.slamd.com/`).

## Unit of Replication

The smallest unit of replication is the suffix. The replication mechanism requires one suffix to correspond to one database. You cannot replicate a suffix, or namespace, that is distributed over two or more databases using custom distribution logic. The unit of replication applies to both consumers and suppliers, which means that you cannot replicate two suffixes to a consumer that holds only one suffix.

## Change Log

Every server that acts as a supplier maintains a change log. A *change log* is a record that describes the modifications that have occurred on a master replica. The supplier replays these modifications to its consumers. When an entry is modified, renamed, added, or deleted, a change record that describes the LDAP operation is recorded in the change log.

## Replication Agreement

Directory Server uses replication agreements to define how replication occurs between two servers. A *replication agreement* describes replication between *one* supplier and *one* consumer.

A replication agreement identifies the following:

- The suffix to replicate
- The consumer server to which the data is pushed
- The times during which replication can occur
- The bind DN and credentials that the supplier must use to bind to the consumer
- How the connection is secured, SSL or client authentication, for example
- Information about the replication status for this particular agreement
- Information about replication filtering

## Replication Priority

In versions of Directory Server prior to Directory Server 6.x, updates were replicated in chronological order. In this version of the product, updates can be prioritized for replication. Priority is a boolean feature, it is on or off. There are no levels of priority. In a queue of updates waiting to be replicated, updates with priority are replicated before updates without priority. In a queue of updates waiting to be replicated, updates with priority are replicated before updates without priority.

The priority rules are configured according to the following parameters:

- The identity of the client
- The type of update
- The entry or subtree that was updated
- The attributes changed by the update

For more information, see "Prioritized Replication" in *Sun Java System Directory Server Enterprise Edition 6.3 Reference*.

## Assessing Initial Replication Requirements

A successful replicated directory service requires comprehensive testing and analysis in a production environment. However, the following basic calculation enables you to start designing a replicated topology. The sections that follow use the result of this calculation as the basis of the replicated topology design.

### ▼ To Determine Initial Replication Requirements

1. **Estimate the maximum number of searches per second that are required at peak usage time.**

   This estimate can be called `Total searches`.

2. **Test the number of searches per second that a single host can achieve.**

   This estimate can be called `Searches per host`. Note that this should be evaluated *with replication enabled*.

   The number of searches that a host can achieve is affected by several variables. Among these are the size of the entries, the capacity of the host, and the speed of the network. A number of third party performance testing tools are available to assist you in conducting these tests. The SLAMD Distributed Load Generation Engine (SLAMD) is an open source Java application designed for stress testing and performance analysis of network-based applications. SLAMD can be used effectively to perform this part of the replication assessment. For information about SLAMD, and to download the SLAMD software, see http://www.slamd.com (`http://www.slamd.com/`).

3. **Calculate the number of hosts that are required.**

   ```
   Number of hosts = Total searches / Searches per host
   ```

## Load Balancing With Multi-Master Replication in a Single Data Center

Replication can balance the load on Directory Server in the following ways:

- By spreading search activities across several servers
- By dedicating specific servers to specific tasks or applications

Generally, if the `Number of hosts` calculated in "Assessing Initial Replication Requirements" on page 160 is about 16, or not significantly larger, your topology should include only master servers in a fully connected topology. Fully connected means that every master replicates to every other master in the topology.

---

**Note –** The `Number of hosts` is approximate and depends on the hardware and other details of the deployment.

---

The following figure assumes that the `Number of hosts` is two. LDAP operations are divided between two master servers, based on the type of client application. This strategy reduces the load that is placed on each server and increases the total number of operations that can be served by the deployment.

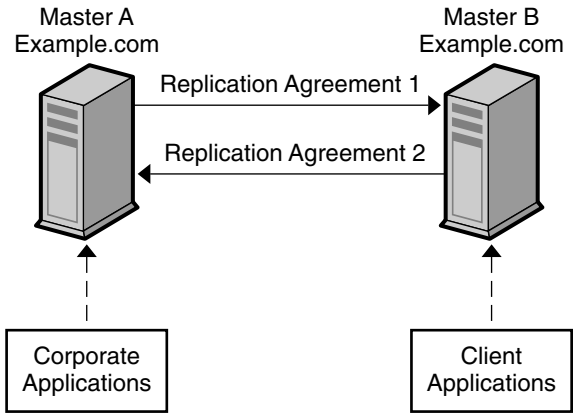



**FIGURE 10–2** Using Multi-Master Replication for Load Balancing

For a similar scenario in a global deployment, see "Using Multi-Master Replication Over a WAN" on page 177.

## Load Balancing With Replication in Large Deployments

If your deployment requires a `Number of hosts` significantly larger than 16, you might need to add dedicated consumers to the topology.

The following figure assumes that the `Number of hosts` is 24 and, for simplicity, shows only a portion of the topology. (The remaining 10 servers would have an identical configuration, with a total of 8 masters and 16 consumers.

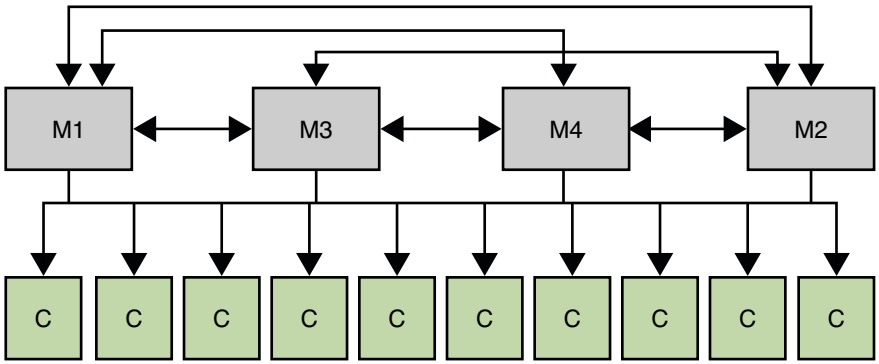



**FIGURE 10–3** Using Multi-Master Replication for Load Balancing in a Large Deployment

A change log can be enabled on any of these consumers if you need to do the following:

- Promote the consumer to a master in the event of an outage
- Perform a binary initialization from a master to any one of the consumers

If the `Number of hosts` is several hundred, you might want to add hubs to the topology. In such a case, there should be more hubs than masters, with up to 10 hubs for each master. Each hub should handle replication to only 20 consumers at most.

No topology should have the same number of hubs as masters, or the same number of hubs as consumers.

## Using Server Groups to Simplify Multi-Master Topologies

When the `Number of hosts` is large, the use of *server groups* can simplify the topology and improve resource usage. In a topology with 16 masters, the use of four server groups, each containing four masters, is easier to manage than 16 fully meshed masters.

Setting up a such a topology involves the following steps:

- Configure the 16 masters, without any replication agreements.
- Create four server groups and include four masters in each group.
- Set up replication agreements between all the masters in a single group.
- Set up replication agreements between the first master of each group, the second master of each group, and so forth.

The following figure shows the resulting topology.

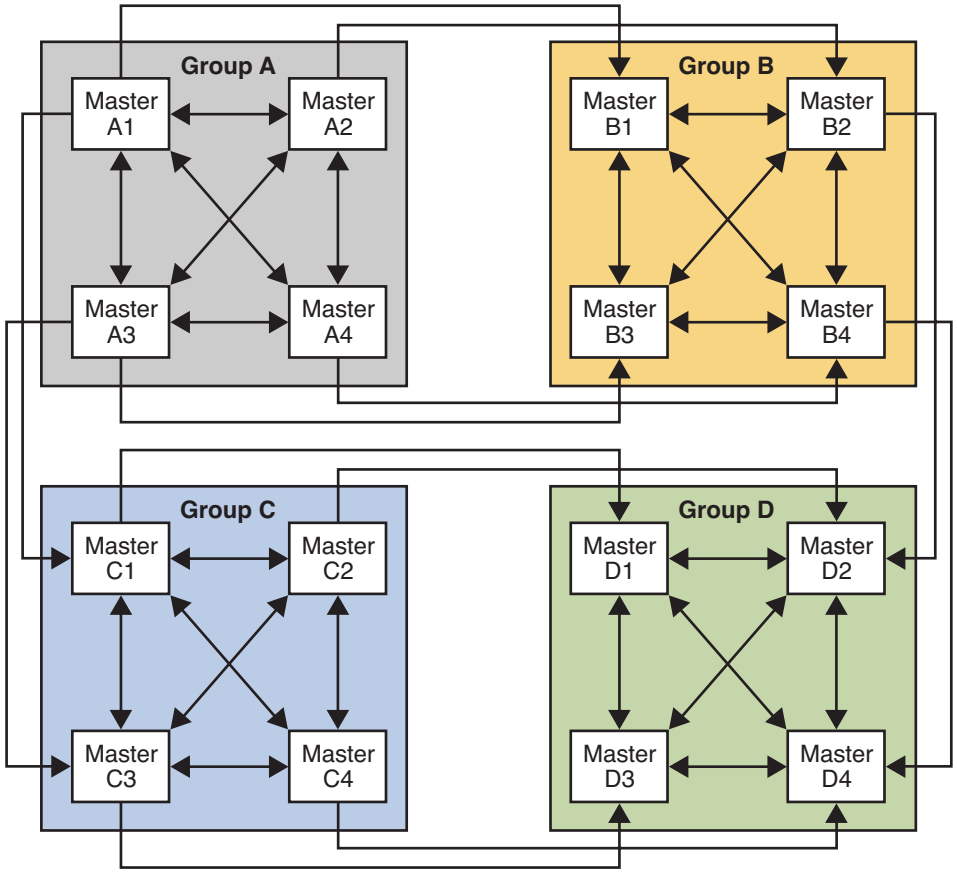



**FIGURE 10–4** Server Groups in Multi-Master Topologies

# Using Directory Proxy Server for Load Balancing

Directory Proxy Server can use multiple servers to distribute the load of a single source of data. Directory Proxy Server can also ensure that if one of the servers is unavailable, the data remains available. Apart from distributing data, Directory Proxy Server provides operation-based load balancing. That is, the server is able to route client operations to a specific Directory Server, based on the *type* of operation.

Directory Proxy Server supports operation-based load balancing, and a variety of load balancing algorithms that determine how the workload is shared between Directory Servers. For a detailed description of each of these algorithms, see Chapter 16, "Directory Proxy Server Load Balancing and Client Affinity," in *Sun Java System Directory Server Enterprise Edition 6.3 Reference*.

The following figure illustrates how the proportional algorithm is used to balance read load across two servers. Operation-based load balancing routes all writes to Master 1, unless that server fails. On failure all reads and writes are routed to Master 2.

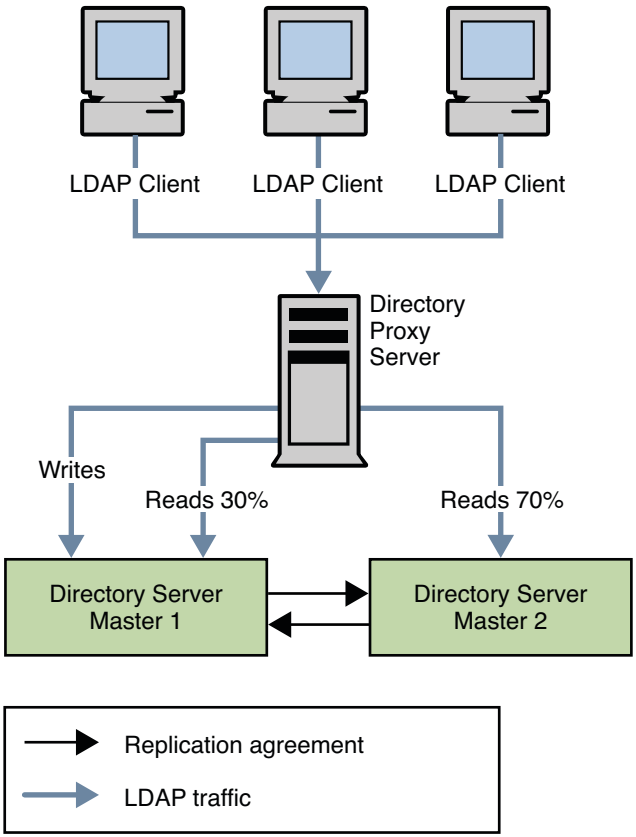



**FIGURE 10–5** Using Proportional and Operation-Based Load Balancing in a Scaled Deployment

Note that the configuration for load balancing is not recalculated when one server instance fails. You cannot use proportional load balancing to create a “hot standby” server by setting a server's load balancing weight to 0.

Imagine, for example, you have three servers A, B, and C. Proportional load balancing has been configured such that servers A and B each receive 50% of the load. Server C is configured to have 0% of the load as it is designed to be a standby server only. If server A fails, 100% of the load will go to server B automatically. Only if server B also fails, will the load be distributed to server C. So, either the instance participates in load balancing all the time, always ready to take part of the load, or all primary instances have to fail before that server will take any load.

You can achieve something like a hot standby by using the saturation load balancing algorithm and applying a low weight to the standby server. Although the server is not a true standby server, you can configure the algorithm such that requests are distributed to this server only if the primary servers are under heavy load. Effectively if one primary server is disabled, the load on the other primary servers increases to the extent that requests must be distributed to the standby server.

# Using Distribution for Write Scalability

Write operations are resource intensive. When a client requests a write operation, the follow sequence of events occurs on the database:

- The backend database is locked
- The entry is locked in the database cache
- The access control check plug-in is called
- Any backend pre-operation plug-ins are called
- The database transaction begins
- The database files are updated
- The old entry cache is replaced with new data
- The database transaction is committed
- Any backend post-operation plug-ins are called
- The backend database is unlocked

Because of this complex procedure, an increased number of writes can have a dramatic impact on performance.

As an enterprise grows, more client applications require rapid write access to the directory. Also, as more information is stored in a single Directory Server, the cost of adding or modifying entries in the directory database increases. This is because indexes become larger and it takes longer to manipulate the information that the indexes contain.

In some cases, the service level agreements might only be achieved by having all the data cached in memory. However, the data might be too large to fit on a single memory machine

When the volume of directory data increases to this extent, you need to break up the data so that it can be stored in multiple servers. One approach is to use a hierarchy to divide the information. By separating the information into multiple branches based on some criteria, each branch can be stored on a separate server. Each server can then be configured with chaining or referrals to enable clients to access all the information from a single point.

In this kind of division, each server is responsible for only a part of the directory tree. A distributed directory works in a similar way to the Domain Name Service (DNS). The DNS assigns each portion of the DNS namespace to a particular DNS server. In the same way, you can distribute your directory namespace across servers while maintaining, from a client standpoint, a single directory tree.

A hierarchy-based distribution mechanism has certain disadvantages. The main problem is that this mechanism requires that the clients know exactly where the information is. Alternatively, the clients must perform a broad search to find the data. Another problem is that some directory-enabled applications might not have the capability to deal with the information if it is broken up into multiple branches.

Directory Server supports hierarchy-based distribution in conjunction with the chaining and referral mechanisms. However, a distribution feature is also provided with Directory Proxy Server, which supports smart routing. This feature enables you to decide on the best distribution mechanism for your enterprise.

## Using Multiple Databases

Directory Server stores data in high-performance, disk-based LDBM databases. Each database consists of a set of files that contains all of the data that is assigned to this set. You can store different portions of your directory tree in different databases. Imagine, for example, that your directory tree contains three subsuffixes, as shown in the following figure.

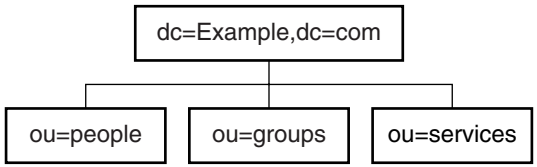



**FIGURE 10–6** Directory Tree With Three Subsuffixes

The data of the three subsuffixes can be stored in three separate databases as shown in the following figure.

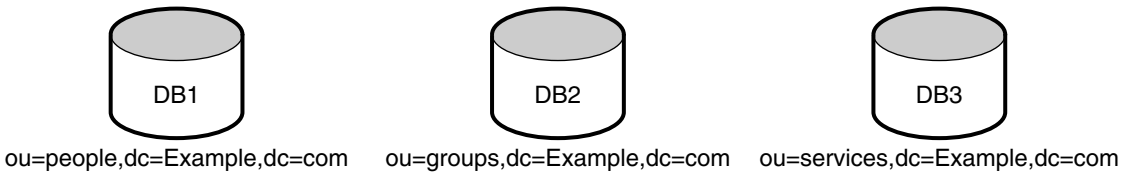



**FIGURE 10–7** Three Subsuffixes Stored in Three Separate Databases

When you divide your directory tree among databases, the databases can be distributed across multiple servers. This strategy generally equates to several physical machines, which improves performance. The three databases in the preceding illustration can be stored on two servers as shown in the following figure.

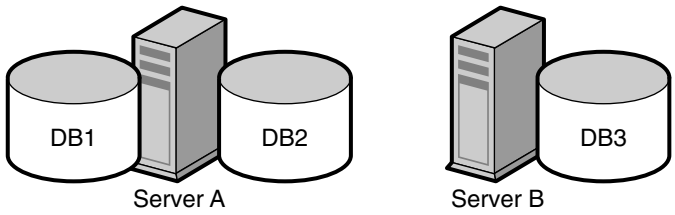



**FIGURE 10–8** Three Databases Stored on Two Separate Servers

When databases are distributed across multiple servers, the amount of work that each server needs to do is reduced. Thus, the directory can be made to scale to a much larger number of entries than would be possible with a single server. Because Directory Server supports dynamic addition of databases, you can add new databases as required, without making the entire directory unavailable.

# Using Directory Proxy Server for Distribution

Directory Proxy Server divides directory information into multiple servers but does not require that the hierarchy of the data be altered. An important aspect of data distribution is the ability break up the data set in a logical manner. However, distribution logic that works well for one client application might not work as well for another client application.

For this reason, Directory Proxy Server enables you to specify how data is distributed and how directory requests should be routed. For example, LDAP operations can be routed to different directory servers based on the directory information tree (DIT) hierarchy. The operations can also be routed based on operation type or on a custom distribution algorithm.

Directory Proxy Server effectively hides the distribution details from the client application. From the clients' standpoint, a single directory addresses their directory queries. Client requests are distributed according to a particular distribution method. Different routing strategies can be associated with different portions of the DIT, as explained in the following sections.

## Routing Based on the DIT

This strategy can be used to distribute directory entries based on the DIT structure. For example, entries in the subtree `o=sales,dc=example,dc=com` can be routed to Directory Server A, and entries in the subtree `o=hr,dc=example,dc=com` can be routed to Directory Server B.

## Routing Based on a Custom Algorithm

In some cases, you might want to distribute entries across directory servers without using the DIT structure. Consider, for example, a service provider who stores entries that represent subscribers under `ou=subscribers,dc=example,dc=com`. As the number of subscribers grows, there might be a need to distribute them across servers based on the range of the subscriber ID.

With a custom routing algorithm, subscriber entries with an ID in the range 1-10000 can be located in Directory Server A, and subscriber entries with an ID in the range 10001-infinity can be located in Directory Server B. If the data on server B grows too large, the distribution algorithm can be changed so that entries with an ID starting from 2000 can be located on a new server, Server C.

You can implement your own routing algorithm using the Directory Proxy Server `DistributionAlgorithm` interface.

## Using Directory Proxy Server to Distribute Requests Based on Bind DN

In this scenario, an enterprise distributes customer data between three master servers based on geographical location. Customers that are based in the United Kingdom have their data stored on a master server in London. French customers have their data stored on a master server in Paris. The data for Japanese customers is stored on a master server in Tokyo. Customers can update their own data through a single web-based interface.

Users can update their own information in the directory using a web-based application. During the authentication phase, users enter an email address. email addresses for customers in the UK take the form `*@uk.example.com`. For French customers, the email addresses take the form `*@fr.example.com`, and for Japanese customers, `*@ja.example.com`. Directory Proxy Server receives these requests through an LDAP-enabled client application. Directory Proxy Server then routes the requests to the appropriate master server based on the email address entered during authentication.

This scenario is illustrated in the following figure.

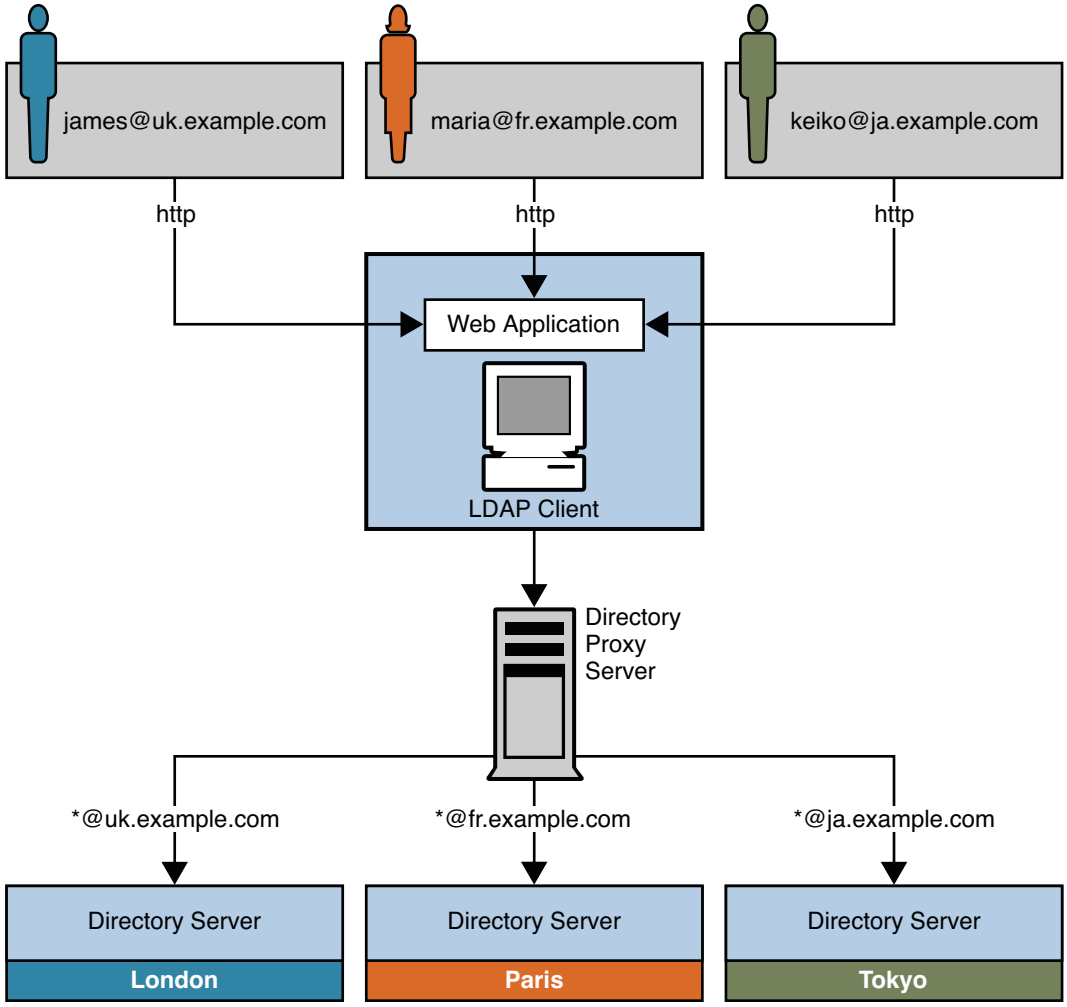



**FIGURE 10–9** Using Directory Proxy Server to Route Requests Based on Bind DN

# Distributing Data Lower Down in a DIT

In many cases, data distribution is not required at the top of the DIT. However, entries further up the tree might be required by the entries in the portion of the tree that has been distributed. This section provides a sample scenario that shows how to design a distribution strategy in this case.

# Logical View of Distributed Data

Example.com has one subtree for groups and a separate subtree for people. The number of group definitions is small and fairly static, while the number of person entries is large, and continues to grow. Example.com therefore requires only the people entries to be distributed across three servers. However, the group definitions, their ACIs, and the ACIs located at the top of the naming context are required to access all entries under the people subtree.

The following illustration provides a logical view of the data distribution requirements.

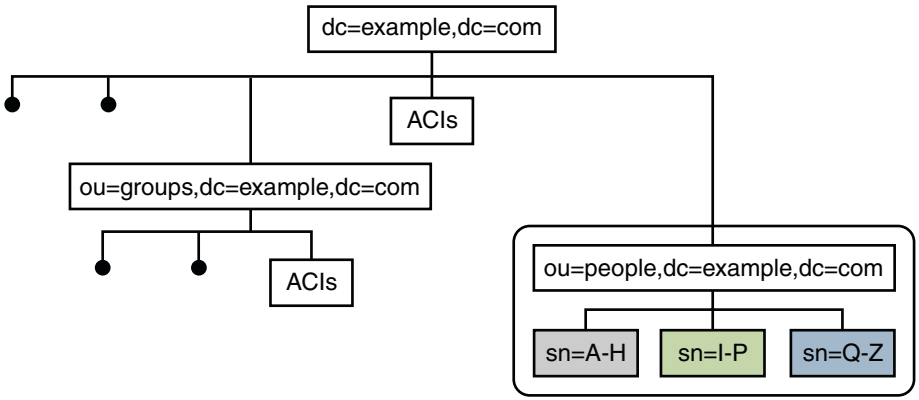



**FIGURE 10–10** Logical View of Distributed Data

# Physical View of Data Storage

The `ou=people` subtree is split across three servers, according to the first letter of the `sn` attribute for each entry. The naming context (`dc=example,dc=com`) and the `ou=groups` containers are stored in one database on each server. This database is accessible to entries under `ou=people`. The `ou=people` container is stored in its own database.

The following illustration shows how the data is stored on the individual Directory Servers.

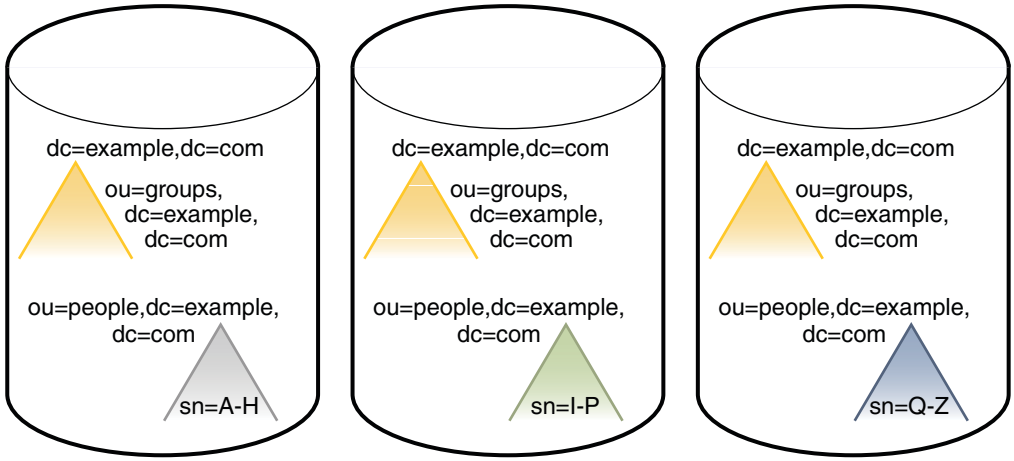



**FIGURE 10–11** Physical View of Data Storage

Note that the `ou=people` container is *not* a subsuffix of the top container.

## Directory Server Configuration for Sample Distribution Scenario

Each server described previously can be understood as a distribution *chunk*. The suffix that contains the naming context and the entries under `ou=groups`, is the same on each chunk. A multi-master replication agreement is therefore set up for this suffix across each of the three chunks.

For availability, each chunk is also replicated. At least two master replicas are therefore defined for each chunk.

The following illustration shows the Directory Server configuration with three replicas defined for each chunk. For simplification, the replication agreements are only shown for one chunk, although they are the same for the other two chunks.

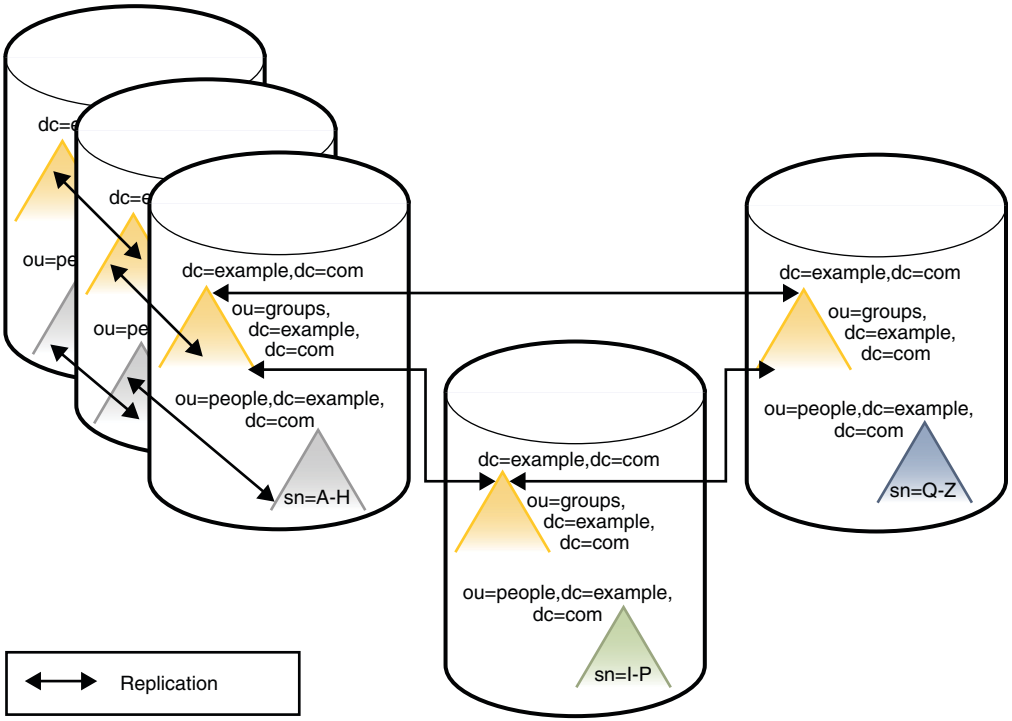



**FIGURE 10–12** Directory Server Configuration

# Directory Proxy Server Configuration for Sample Distribution Scenario

Client access to directory data through Directory Proxy Server is provided through *data views*. For information about data views see Chapter 17, "Directory Proxy Server Distribution," in *Sun Java System Directory Server Enterprise Edition 6.3 Reference*.

For this scenario, one data view is required for each distributed suffix, and one data view is required for the naming context (`dc=example,dc=com`) and the `ou=groups` subtrees.

The following illustration shows the configuration of Directory Proxy Server data views to provide access to the distributed data.

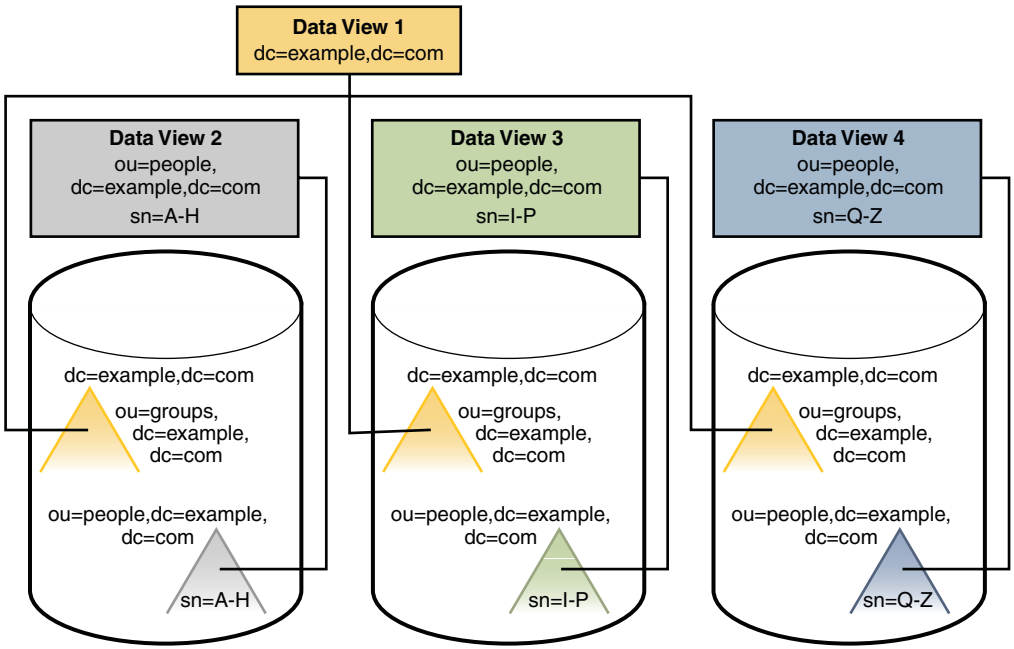



**FIGURE 10–13** Directory Proxy Server Configuration

## Considerations for Data Growth

Distributed data is split according to a distribution algorithm. When you decide which distribution algorithm to use, bear in mind that the volume of data might change, and that your distribution strategy must be scalable. Do not use an algorithm that necessitates complete redistribution of data.

A numeric distribution algorithm based on `uid`, for example, can be scaled fairly easily. If you start with two data segments of `uid=0-999` and `uid=1000–1999`, it is easy to add third segment of `uid=2000–2999` at a later stage.

# Using Referrals For Distribution

A referral is information returned by a server that tells a client application which server to contact to proceed with an operation request. If you do not use Directory Proxy Server to manage distribution logic, you must define the relationships between distributed data in another way. One way to define relationships is using *referrals*.

Directory Server supports three ways of configuring how and when referrals are returned:

- **Default referrals.** The directory returns a default referral when a client application presents a DN for which the server does not have a matching suffix.
- **Suffix referrals.** When an entire suffix has been taken offline for maintenance or security reasons, the server returns the referrals defined by that suffix. Read-only replicas of a suffix also return referrals to the master server when a client requests a write operation.
- **Smart referrals.** These referrals are stored on entries within the directory. Smart referrals point to Directory Servers that have knowledge of the subtree whose DN matches the DN of the entry that contains the smart referral.

The following figure illustrates how referrals are used to direct clients from the UK to the appropriate server in a global topology. In this scenario, the client application must be able to connect to all the servers in the topology (at the TCP/IP level), to enable it to follow the referral.

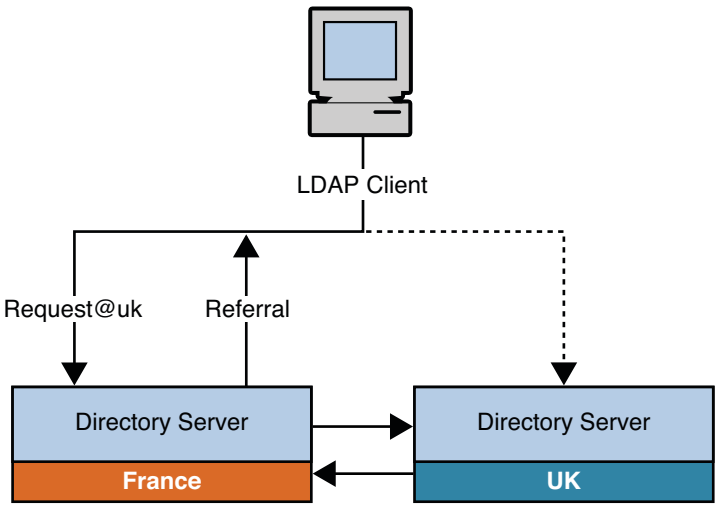



**FIGURE 10–14** Using Referrals to Direct Clients to a Specific Server

# Using Directory Proxy Server With Referrals

You can use Directory Proxy Server in conjunction with the referral mechanism to achieve the same result. The advantage of using Directory Proxy Server in this regard is that the load and complexity of client applications is reduced. Client applications are only aware of the Directory Proxy Server URL. If the distribution logic is changed, for any reason, this change is transparent to client applications.

The following figure illustrates how the scenario described previously can be simplified with the use of Directory Proxy Server. Client applications always connect to the Proxy Server, which handles the referrals itself.

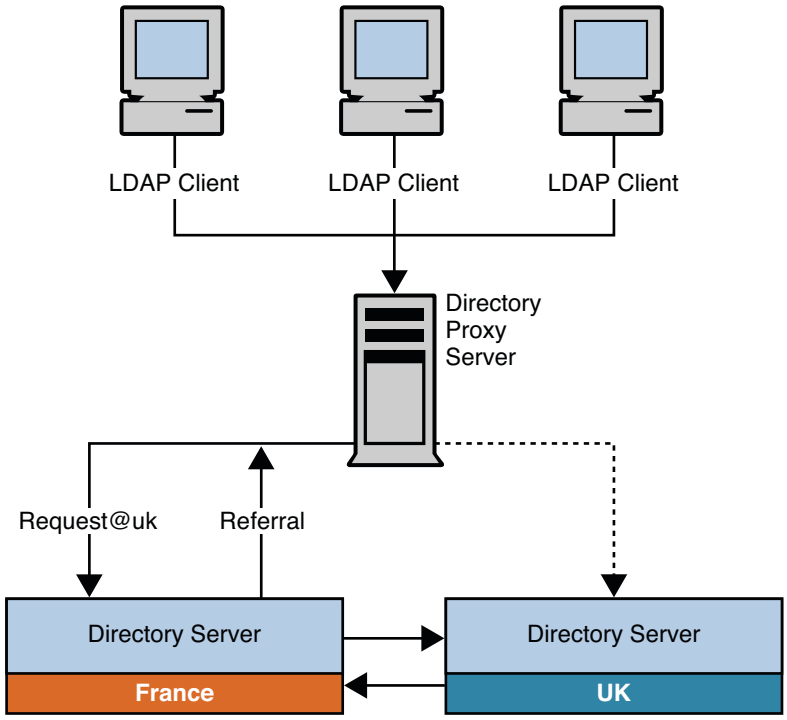



**FIGURE 10–15** Using Directory Proxy Server With Referrals

◆ ◆ ◆ CHAPTER 11

# Designing a Global Deployment

In a global deployment, access to directory services is required in more than one geographical location, or data center. This chapter provides strategies for effectively deploying Directory Server Enterprise Edition across multiple data centers. The strategies ensure that the quality of service requirements identified in Chapter 5, "Defining Service Level Agreements," are not compromised.

This chapter covers the following topics:



## Using Replication Across Multiple Data Centers

One of the goals of replication is to enable geographic distribution of the LDAP service. Replication enables you to have identical copies of information on multiple servers and across more than one data center. Replication concepts are outlined in Chapter 10, "Designing a Scaled Deployment," in this guide, and described in detail in Chapter 4, "Directory Server Replication," in *Sun Java System Directory Server Enterprise Edition 6.3 Reference*. This chapter focuses on the replication features that are used in a global deployment.

### Using Multi-Master Replication Over a WAN

Directory Server supports multi-master replication over a WAN. This feature enables multi-master replication configurations across geographical boundaries in international, multiple data center deployments.

Generally, if the `Number of hosts` calculated in "Assessing Initial Replication Requirements" on page 160 is less than 16, or not significantly larger, your topology should include only master servers in a fully connected topology, that is, every master replicates to every other master in the topology. In a multi-master replication over WAN configuration, *all* Directory Server instances

separated by a WAN must *not* be running versions prior to Directory Server 5.2. For a multi-master topology with more than 4 masters, Directory Server 6.x is required.

The replication protocol provides full asynchronous support, as well as window, grouping, and compression mechanisms. These features make multi-master replication over a WAN viable. Replication data transfer rates will always be less than what the available physical medium allows in terms of bandwidth. If the update volume between replicas cannot physically be made to fit into the available bandwidth, tuning will not prevent replicas from diverging under heavy update load. Replication delay and update performance are dependent on many factors, including but not limited to modification rate, entry size, server hardware, average latency and average bandwidth.

Internal parameters of the replication mechanism are optimized by default for WANs. However, if you experience slow replication due to the factors mentioned above, you may wish to empirically adjust the window size and group size parameters. You may also be able to schedule your replication to avoid peak network times, thus improving your overall network usage. Finally, Directory Server supports the compression of replication data to optimize bandwidth usage.

When you replicate data over a WAN link, some form of security to ensure data integrity and confidentiality is advised. For more information on security methods available in Directory Server, see Chapter 2, "Directory Server Security," in *Sun Java System Directory Server Enterprise Edition 6.3 Reference*.

## Group and Window Mechanisms

Directory Server provides group and window mechanisms to optimize replication flow. The group mechanism enables you to specify that changes are sent in groups, rather than individually. The group size represents the maximum number of data modifications that can be bundled into a single update message. If the network connection appears to be the bottleneck for replication, increase the group size and check replication performance again. For information on configuring the group size, see "Configuring Group Size" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

The window mechanism specifies that a certain number of update requests are sent to the consumer, without the supplier having to wait for an acknowledgement from the consumer before continuing. The window size represents the maximum number of update messages that can be sent without immediate acknowledgement from the consumer. It is more efficient to send many messages in quick succession instead of waiting for an acknowledgement after each one. Using the appropriate window size, you can eliminate the time replicas spend waiting for replication updates or acknowledgements to arrive. If your consumer replica is lagging behind the supplier, increase the window size to a higher value than the default, such as 100, and check replication performance again before making further adjustments. When the replication update rate is high and the time between updates is therefore small, even replicas connected by a

LAN can benefit from a higher window size. For information on configuring the window size, see "Configuring Window Size" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

Both the group and window mechanisms are based on change size. Therefore, optimizing replication performance with these mechanisms might be impractical if the size of your changes varies considerably. If the size of your changes is relatively constant, you can use the group and window mechanisms to optimize incremental and total updates.

### Replication Compression

In addition to the grouping and window mechanisms, you can configure replication compression on Solaris and Linux platforms. Replication compression streamlines replication flow, which substantially reduces the incidence of bottlenecks in replication over a WAN. Compression of replicated data can increase replication performance in specific cases, such as networks with sufficient CPU but low bandwidth, or when there are bulk changes to be replicated. You can also benefit from replication compression when initializing a remote replica with large entries. Do not set this parameter in a LAN (local area network) where there is wide network bandwidth, because the compression and decompression computations will slow down replication.

The replication mechanism uses the Zlib compression library. Empirically test and select the compression level that gives you best results in your WAN environment for your expected replication usage.

For more information on configuring replication compression, see "Configuring Replication Compression" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

## Using Fractional Replication

A global topology (with data centers in different countries) might require restricting replication for security or compliance reasons. For example, legal restrictions might state that specific employee information cannot be copied outside of the U.S.A. Or, a site in Australia might require Australian employee details only.

The fractional replication feature enables only a subset of the attributes that are present in an entry to be replicated. Attribute lists are used to determine which attributes can and cannot be replicated. Fractional replication can only be applied to read-only consumers.

For detailed information about how fractional replication works, see "Fractional Replication" in *Sun Java System Directory Server Enterprise Edition 6.3 Reference*. For information about how to configure fractional replication, see "Fractional Replication" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

## Using Prioritized Replication

Prioritized replication can be used when there is a strong business requirement to have tighter consistency for replicated data on specific attributes. In 5.x version of Directory Server, updates were replicated in the order in which they were received. With prioritized replication, you can specify that updates to certain attributes take precedence when they are replicated to other servers in the topology.

Prioritized replication provides the following benefits:

- Improved security. Prioritized replication is used by default for account lockout. Imagine for example that an employee leaves your organization, and you lock the employee's account. To ensure that the employee cannot log in to a remote server to which the account lockout has not been replicated, account lockout changes are replicated before other changes are replicated.
- Improved consistency. Directory Server replication is *loosely consistent*. With prioritized replication, you can assure stronger consistency for certain attributes that are considered important in your organization.

## Sample Replication Strategy for an International Enterprise

In this scenario, an enterprise has two major data centers, one in London and the other in New York, separated by a WAN. The scenario assumes that the network is very busy during normal business hours.

In this scenario, the `Number of hosts` has been calculated to be eight. A fully connected, 4-way multi-master topology is deployed in each of the two data centers. These two topologies are also fully connected to each other. For ease of comprehension, not all replication agreements between the two data centers are shown in the following diagram.

The replication strategy for this scenario includes the following:

- Master copies of directory data are held on servers in both data centers.
- A multi-master replication topology is deployed between the data centers to provide high availability and write-failover across the deployment.
- Replication across the WAN link is scheduled so that it occurs only during off-peak hours to optimize bandwidth.
- To increase performance, client applications are directed to local servers. Clients in the U.S. read from and write to masters in the New York data center. Clients in the UK read from and write to masters in the London data center.

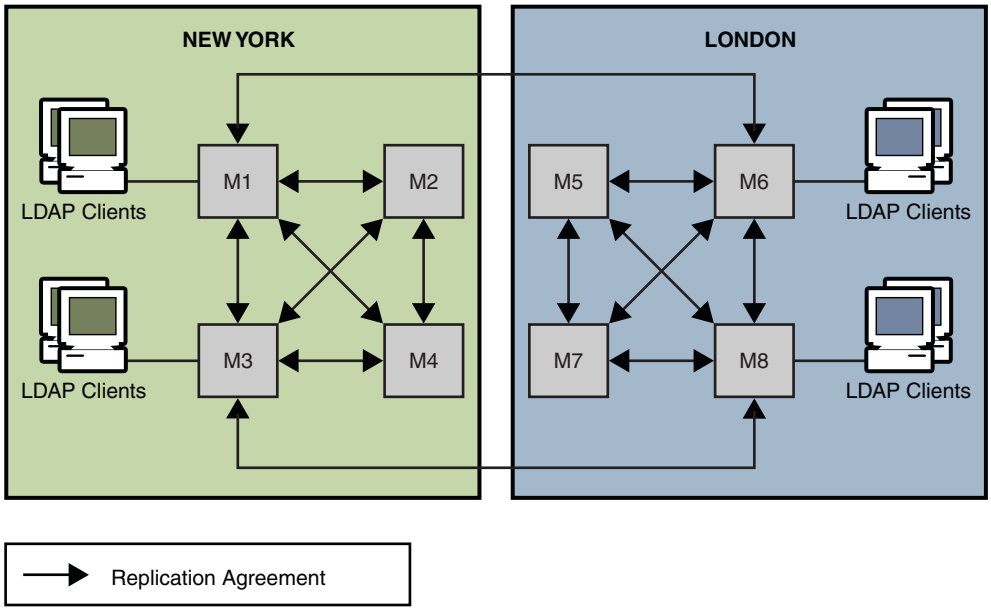



**FIGURE 11–1** Using Multi-Master Replication for Load Balancing in Two Data Centers

# Using Directory Proxy Server in a Global Deployment

In a global enterprise, a centralized data model can cause scalability and performance issues. Directory Proxy Server can be used in such a situation to distribute data efficiently and to route search and update requests appropriately.

## Sample Distribution Strategy for a Global Enterprise

In the architecture shown here, a large financial institution has its headquarters in London. The organization has data centers in London, New York, and Hong Kong. Currently, the vast majority of the data that is available to employees resides centrally in legacy RDBMS repositories in London. All access to this data from the financial institution's client community is over the WAN.

The organization is experiencing scalability and performance problems with this centralized model and decides to move to a distributed data model. The organization also decides to deploy an LDAP directory infrastructure at the same time. Because the data in question is considered "mission critical" it must be deployed in a highly available, fault-tolerant infrastructure.

An analysis of client application profiles has revealed that the data is customer-based. Therefore, 95 percent of the data accessed by a geographical client community is specific to that

community. Clients in Asia rarely access data for a customer in North America, although this does happen infrequently. The client community must also update customer information from time to time.

The following figure shows the logical architecture of the distributed solution.

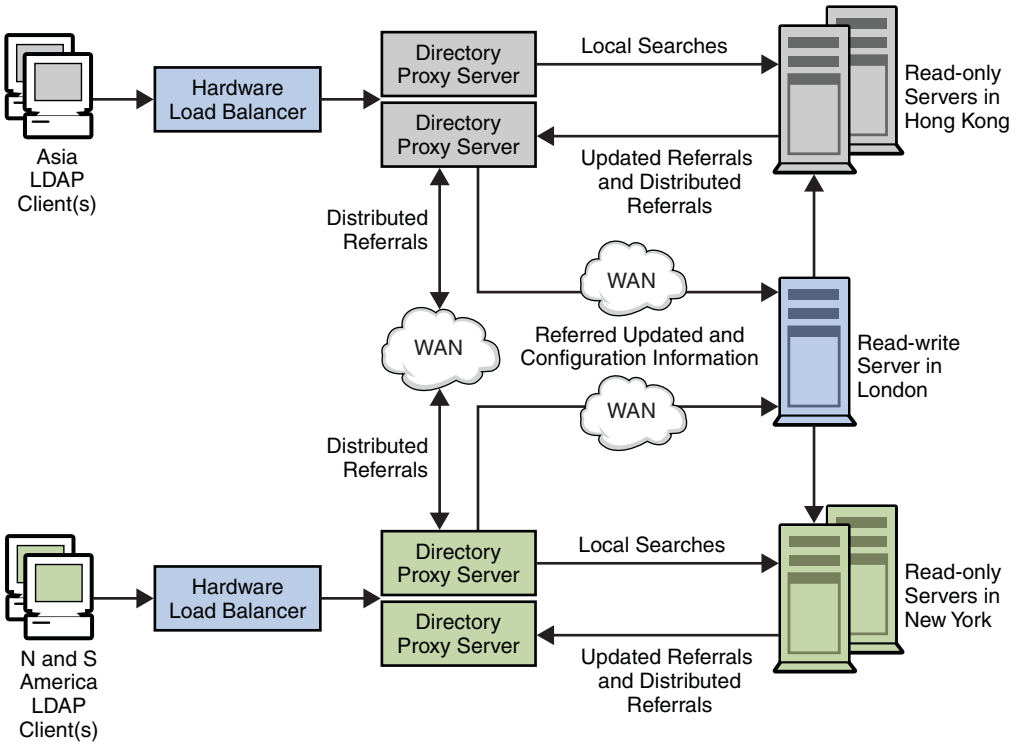



**FIGURE 11–2** Distributed Directory Infrastructure

Given the profile of 95 percent local data access, the organization decides to distribute the directory infrastructure geographically. Multiple directory consumers are deployed in each geographical location: Hong Kong, New York, and London. London consumers are not shown in the diagram for ease of understanding. Each of these consumers is configured to hold the customer data specific to the location. Data for European and Middle East customers is held in the London consumers. Data for North and South American customers is held in the New York consumers. Data for Asian and Pacific Rim customers is held in the Hong Kong consumers.

With this deployment, the overwhelming data requirement of the local client community is located in the community. This strategy provides significant performance improvements over the centralized model. Client requests are processed locally, reducing network overhead. The local directory servers effectively partition the directory infrastructure, which provides increased directory server performance and scalability. Each set of consumer directory servers

is configured to return referrals if a client submits an update request. Referrals are also returned if a client submits a search request for data that is located elsewhere.

Client LDAP requests are sent to Directory Proxy Server through a hardware load balancer. The hardware load balancer ensures that clients always have access to at least one Directory Proxy Server. The locally deployed Directory Proxy Server initially routes all requests to the array of local directory servers that hold the local customer data. The instances of Directory Proxy Server are configured to load balance across the array of directory servers. This load balancing provides automatic failover and failback.

Client search requests for local customer information are satisfied by a local directory. Appropriate responses are returned to the client through Directory Proxy Server. Client search requests for geographically "foreign" customer information are initially satisfied by the local directory server by returning a referral back to Directory Proxy Server.

This referral contains an LDAP URL that points to the appropriate geographically distributed Directory Proxy Server instance. The local Directory Proxy Server processes the referral on behalf of the local client. The local Directory Proxy Server then sends the search request to the appropriate distributed instance of Directory Proxy Server. The distributed Directory Proxy Server forwards the search request on to the distributed Directory Server and receives the appropriate response. This response is then returned to the local client through the distributed and the local instances of Directory Proxy Server.

Update requests received by the local Directory Proxy Server are also satisfied initially by a referral returned by the local Directory Server. Directory Proxy Server follows the referral on behalf of the local client. However, this time the proxy forwards the update request to the supplier directory server located in London. The supplier Directory Server applies the update to the supplier database and sends a response back to the local client through the local Directory Proxy Server. Subsequently, the supplier Directory Server propagates the update down to the appropriate consumer Directory Server.

◆ ◆ ◆ CHAPTER 12

# Designing a Highly Available Deployment

High availability implies an agreed minimum "up time" and level of performance for your directory service. Agreed service levels vary from organization to organization. Service levels might depend on factors such as the time of day systems are accessed, whether or not systems can be brought down for maintenance, and the cost of downtime to the organization. Failure, in this context, is defined as anything that prevents the directory service from providing this minimum level of service.

This chapter covers the following topics:



## Availability and Single Points of Failure

Directory Server Enterprise Edition deployments that provide high availability can quickly recover from failures. With a high availability deployment, component failures might impact individual directory queries but should not result in complete system failure. A single point of failure (SPOF) is a system component which, upon failure, renders an entire system unavailable or unreliable. When you design a highly available deployment, you identify potential SPOFs and investigate how these SPOFs can be mitigated.

SPOFs can be divided into three categories:

- Hardware failures, for example, server crashes, network failures, power failures, or disk drive crashes
- Software failures, for example, Directory Server or Directory Proxy Server crashes
- Database corruption

# Mitigating SPOFs

You can ensure that failure of a single component does not cause an entire directory service to fail by using *redundancy*. Redundancy involves providing redundant software components, hardware components, or both. Examples of this strategy include deploying multiple, replicated instances of Directory Server on separate hosts, or using redundant arrays of independent disks (RAID) for storage of Directory Server databases. Redundancy with replicated Directory Servers is the most efficient way to achieve high availability.

You can also use *clustering* to provide a highly available service. Clustering involves providing pre-packaged high availability hardware and software. An example of this strategy is deploying Sun Cluster hardware and software.

## Deciding Between Redundancy and Clustering

The remainder of this chapter describes in more detail the use of redundancy and clustering to ensure high availability. This section summarizes the advantages and disadvantages of each solution.

### Advantages and Disadvantages of Redundancy

The more common approach to providing a highly available directory service is to use redundant server components and replication. Redundant solutions are usually less expensive and easier to implement than clustering solutions. These solutions are also generally easier to manage. Note that replication, as part of a redundant solution, has numerous functions other than availability. While the main advantage of replication is the ability to split the read load across multiple servers, this advantage causes additional overhead in terms of server management. Replication also offers scalability on read operations and, with proper design, scalability on write operations, within certain limits. For an overview of replication concepts, see Chapter 4, "Directory Server Replication," in *Sun Java System Directory Server Enterprise Edition 6.3 Reference*.

During a failure, a redundant system might provide poorer availability than a clustering solution. Imagine, for example, an environment in which the load is shared between two redundant server components. The failure of one server component might put an excessive load on the other server, making this server respond more slowly to client requests. A slow response might be considered a failure for clients that rely on quick response times. In other words, the availability of the service, even though the service is operational, might not meet the availability requirements of the client.

### Advantages and Disadvantages of Clustering

The main advantage of a clustered solution is automatic recovery from failure, that is, recovery without user intervention. Disadvantages of clustering are complexity and inability to recover from database corruption.

In a clustered environment, the cluster uses the same IP address for Directory Server and Directory Proxy Server, regardless of which cluster node is actually running the service. That is, the IP address is transparent to the client application. In a replicated environment, each machine in the topology has its own IP address. In this case, Directory Proxy Server can be used to provide a single point of access to the directory topology. The replication topology is therefore effectively hidden from client applications. To increase this transparency, Directory Proxy Server can be configured to follow referrals and search references automatically. Directory Proxy Server also provides load balancing and the ability to switch to another machine when one fails.

### How Redundancy and Clustering Handle SPOFs

In terms of the SPOFs that are described at the beginning of this chapter, redundancy and clustering handle failure in the following ways:

- **Single hardware failure.** In a clustered environment, this kind of failure has no impact on the directory service. Only multiple hardware failures impact the service in a cluster.

  A single hardware failure is fatal to a machine that is not in a clustered environment. Therefore, even if you have redundant hardware, manual intervention is required to repair the failure.
- **Directory Server or Directory Proxy Server failure.** In a clustered environment, the server is automatically restarted. Software failure must occur multiple times in quick succession to trigger the service group to switch to another node in the cluster. This handling of a software failure is also true in a redundant environment.
- **Database corruption.** A cluster cannot survive this kind of failure. Depending on the architecture, a redundant solution should be able to survive database corruption.

## Redundancy at the Hardware Level

This section provides basic information about hardware redundancy. Many publications provide comprehensive information about using hardware redundancy for high availability. In particular, see "Blueprints for High Availability" published by John Wiley & Sons, Inc.

Hardware SPOFs can be broadly categorized as follows:

- Network failures
- Failure of the physical servers on which Directory Server or Directory Proxy Server are running
- Load balancer failures
- Storage subsystem failures
- Power supply failures

Failure at the network level can be mitigated by having redundant network components. When designing your deployment, consider having redundant components for the following:

- Internet connection
- Network interface card
- Network cabling
- Network switches
- Gateways and routers

You can mitigate the load balancer as an SPOF by including a redundant load balancer in your architecture.

In the event of database corruption, you must have a database failover strategy to ensure availability. You can mitigate against SPOFs in the storage subsystem by using redundant server controllers. You can also use redundant cabling between controllers and storage subsystems, redundant storage subsystem controllers, or redundant arrays of independent disks.

If you have only one power supply, loss of this supply could make your entire service unavailable. To prevent this situation, consider providing redundant power supplies for hardware, where possible, and diversifying power sources. Additional methods of mitigating SPOFs in the power supply include using surge protectors, multiple power providers, and local battery backups, and generating power locally.

Failure of an entire data center can occur if, for example, a natural disaster strikes a particular geographic region. In this instance, a well-designed multiple data center replication topology can prevent an entire distributed directory service from becoming unavailable. For more information, see "Using Replication and Redundancy for High Availability" on page 189.

## Redundancy at the Software Level

Failure in Directory Server or Directory Proxy Server can include the following:

- Excessive response time
- Write overload
  - Maximized file descriptors
  - Maximized file system
  - Poor storage configuration
  - Too many indexes
- Read overload
- Cache issues
- CPU constraints
- Replication issues
  - Synchronicity
  - Replication propagation delay

  - Replication flow
  - Replication overload
- Large wildcard searches

These SPOFs can be mitigated by having redundant instances of Directory Server and Directory Proxy Server. Redundancy at the software level involves the use of replication. Replication ensures that the redundant servers remain synchronized, and that requests can be rerouted with no downtime. For more information, see "Using Replication and Redundancy for High Availability" on page 189.

# Using Replication and Redundancy for High Availability

Replication can be used to prevent the loss of a single server from causing your directory service to become unavailable. A reliable replication topology ensures that the most recent data is available to clients across data centers, even in the case of a server failure. At a minimum, your local directory tree needs to be replicated to at least one backup server. Some directory architects say that you should replicate three times per physical location for maximum data reliability. In deciding how much to use replication for fault tolerance, consider the quality of the hardware and networks used by your directory. Unreliable hardware requires more backup servers.

Do not use replication as a replacement for a regular data backup policy. For information about backing up directory data, see "Designing Backup and Restore Policies" on page 131 and Chapter 9, "Directory Server Backup and Restore," in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

LDAP client applications are usually configured to search one LDAP server only. Custom client applications can be written to rotate through LDAP servers that are located at different DNS host names. Otherwise, LDAP client applications can only be configured to look at a single DNS host name for Directory Server. You can use Directory Proxy Server, DNS round robins, or network sorts to provide failover to backup Directory Servers. For information about setting up and using DNS round robins or network sorts, see your DNS documentation. For information about how Directory Proxy Server is used in this context, see "Using Directory Proxy Server as Part of a Redundant Solution" on page 190.

To maintain the ability to read data in the directory, a suitable load balancing strategy must be put in place. Both software and hardware load balancing solutions exist to distribute read load across multiple replicas. Each of these solutions can also determine the state of each replica and to manage its participation in the load balancing topology. The solutions might vary in terms of completeness and accuracy.

To maintain write failover over geographically distributed sites, you can use multiple data center replication over WAN. This entails setting up at least two master servers in each data center, and configuring the servers to be fully meshed over the WAN. This strategy prevents

loss of service if any of the masters in the topology fail. Write operations must be routed to an alternative server if a writable server becomes unavailable. Various methods can be used to reroute write operations, including Directory Proxy Server.

The following sections describe how replication and redundancy are used to ensure high availability:



## Using Redundant Replication Agreements

Redundant replication agreements enable rapid recovery in the event of failure. The ability to enable and disable replication agreements means that you can set up replication agreements that are used only if the original replication topology fails. Although this intervention is manual, the strategy is much less time consuming than waiting to set up the replication agreement when it is needed. The use of redundant replication agreements is explained and illustrated in "Sample Topologies Using Redundancy for High Availability" on page 191.

## Promoting and Demoting Replicas

Promoting or demoting a replica changes its role in the replication topology. In a very large topology that contains dedicated consumers and hubs, online promotion and demotion of replicas can form part of a high availability strategy. Imagine, for example, a multi-master replication scenario, with two hubs configured for additional load balancing and failover. If one master goes offline, you can promote one of the hubs to a master to maintain optimal read-write availability. When the master replica comes back online, a simple demotion back to a hub replica returns you to the original topology.

For more information, see "Promoting or Demoting Replicas" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

## Using Directory Proxy Server as Part of a Redundant Solution

Directory Proxy Server is designed to support high availability directory deployments. The proxy provides automatic load balancing as well as automatic failover and fail back among a set of replicated Directory Servers. Should one or more Directory Servers in the topology become unavailable, the load is proportionally redistributed among the remaining servers.

Directory Proxy Server actively monitors the Directory Servers to ensure that the servers are still online. The proxy also examines the status of each operation that is performed. Servers might not all be equivalent in throughput and performance. If a primary server becomes unavailable, traffic that is temporarily redirected to a secondary server is directed back to the primary server as soon as the primary server becomes available.

Note that when data is distributed, multiple disconnected replication topologies must be managed, which makes administration more complex. In addition, Directory Proxy Server relies heavily on the proxy authorization control to manage user authorization. A specific administrative user *must* be created on each Directory Server that is involved in the distribution. These administrative users must be granted proxy access control rights.

# Using Application Isolation for High Availability

Directory Proxy Server can also be used to protect a replicated directory service from failure due to a faulty client application. To improve availability, a limited set of masters or replicas is assigned to each application.

Suppose a faulty application causes a server shutdown when the application performs a specific action. If the application fails over to each successive replica, a single problem with one application can result in failure of the entire replicated topology. To avoid such a scenario, you can restrict failover and load balancing of each application to a limited number of replicas. The potential failure is then limited to this set of replicas, and the impact of the failure on other applications is reduced.

# Sample Topologies Using Redundancy for High Availability

The following sample topologies show how redundancy is used to provide continued service in the event of failure.

## Using Replication for Availability in a Single Data Center

The data center that is illustrated in the following figure has a multi-master topology with three masters. In this scenario, the third master is used only for availability in thdse event of failure. Read and write operations are routed to Masters 1 and 2 by Directory Proxy Server, unless a problem occurs. To speed up recovery and to minimize the number of replication agreements, recovery replication agreements are created. These agreements are disabled by default but can be enabled rapidly in the event of a failure.

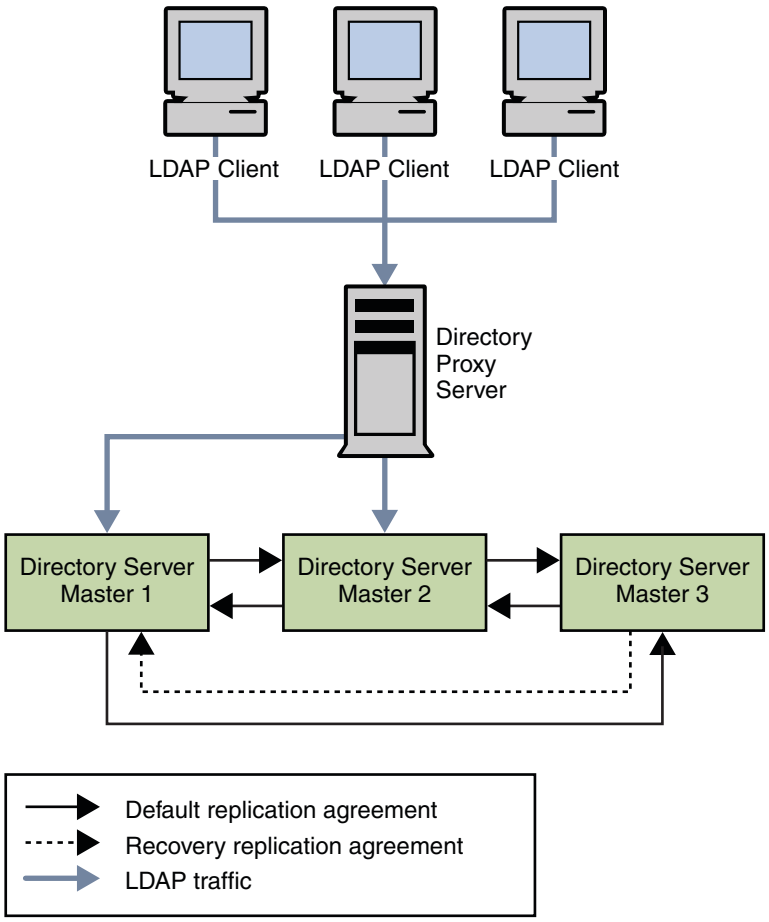



**FIGURE 12–1** Multi-Master Replication in a Single Data Center

## Single Data Center Failure Matrix

In the scenario depicted in Figure 12–1, various components might become unavailable. These potential points of failure and the related recovery actions are described in this table.

**TABLE 12–1** Single Data Center Failure Matrix

| Failed Component | Action |
|---|---|
| Master 1 | Read and write operations are rerouted to Masters 2 and 3 through Directory Proxy Server while Master 1 is repaired. The recovery replication agreement between Master 2 and Master 3 is enabled so that updates to Master 3 are replicated to Master 2. |

**TABLE 12–1** Single Data Center Failure Matrix *(Continued)*

| Failed Component | Action |
|---|---|
| Master 2 | Read and write operations are rerouted to Masters 1 and 3 while Master 2 is repaired. The recovery replication agreement between Master 1 and Master 3 is enabled so that updates to Master 3 are replicated to Master 1. |
| Master 3 | Because Master 3 is a backup server only, the directory service is not affected if this master fails. Master 3 can be taken offline and repaired without interruption to service. |
| Directory Proxy Server | Failure of Directory Proxy Server results in severe service interruption. A redundant instance of Directory Proxy Server is advisable in this topology. For an example of such a topology, see "Using Multiple Directory Proxy Servers" on page 197. |

## Single Data Center Recovery Procedure

In a single data center with three masters, read and write capability is maintained if one master fails. This section describes a sample recovery strategy that can be applied to reinstate the failed component.

The following flowchart and procedure assume that one component, Master 1, has failed. If two masters fail simultaneously, read and write operations must be routed to the remaining master while the problems are fixed.

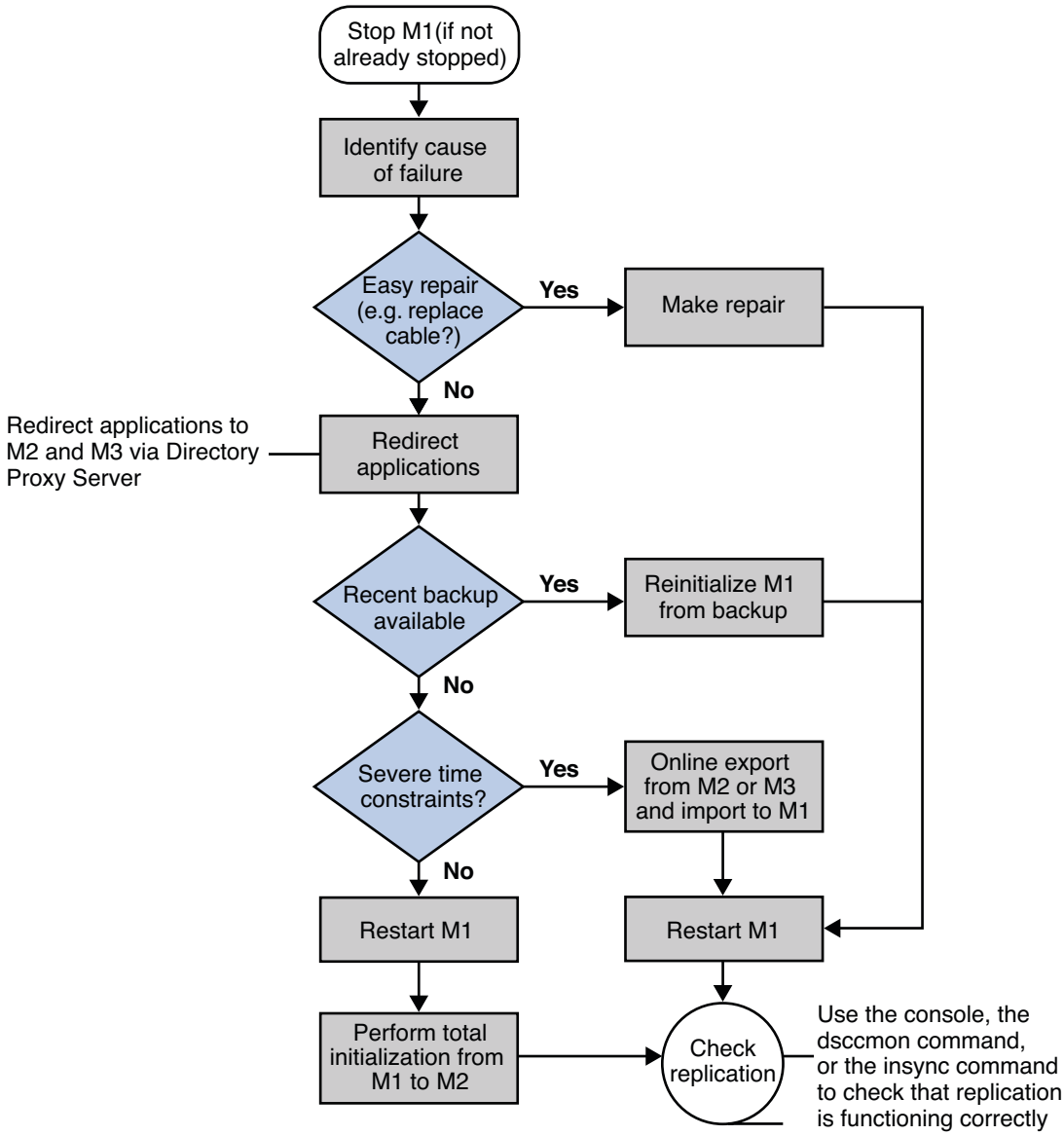



**FIGURE 12–2** Single Data Center Sample Recovery Procedure

### ▼ To Recover on Failure of One Component

1. **If Master 1 is not already stopped, stop it.**

2. **Identify the cause of the failure.**

   - If the failure is easily repaired, by replacing a network cable, for example, make the repair and go to Step 3.
   - If the problem is more serious, the failure might take more time to fix.

   a. **Ensure that any applications that access Master 1 are redirected to point to Master 2 or Master 3, through Directory Proxy Server.**

   b. **Check the availability of a recent backup.**

      - If a recent backup is available, reinitialize Master 1 from the backup and go to Step 3.
      - If a recent backup is *not* available, do one of the following:
        - Restart Master 1 and perform a total initialization from Master 2 or from Master 3 to Master 1.

          For details on this procedure, see "Initializing Replicas" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.
        - If performing a total initialization will take too long, perform an online export from Master 2, or Master 3, and an import to Master 1.

3. **Start Master 1, if it is not already started.**

4. **If Master 1 is in read-only mode, set it to read/write mode.**

5. **Check that replication is functioning correctly.**

   You can use DSCC, `dsccmon view-suffixes`, or the `insync` command to check replication.

   For more information, see "Getting Replication Status" in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*, `dsccmon`(1M), and `insync`(1).

## Using Replication for Availability Across Two Data Centers

Generally in a deployment with two data centers, the same recovery strategy can be applied as described for a single data center. If one or more masters become unavailable, Directory Proxy Server automatically reroutes local reads and writes to the remaining masters.

As in the single data center scenario described previously, recovery replication agreements can be enabled. These agreements ensure that both data centers continue to receive replicated updates in the event of failure. This recovery strategy is illustrated in Figure 12–3.

An alternative to using recovery replication agreements is to use a fully meshed topology in which every master replicates its changes to every other master. While fewer replication agreements might be easier to manage, no technical reason exists for not using a fully meshed topology.

The only SPOF in this scenario would be the Directory Proxy Server in each data center. Redundant Directory Proxy Servers can be deployed to eliminate this problem, as shown in Figure 12–4.

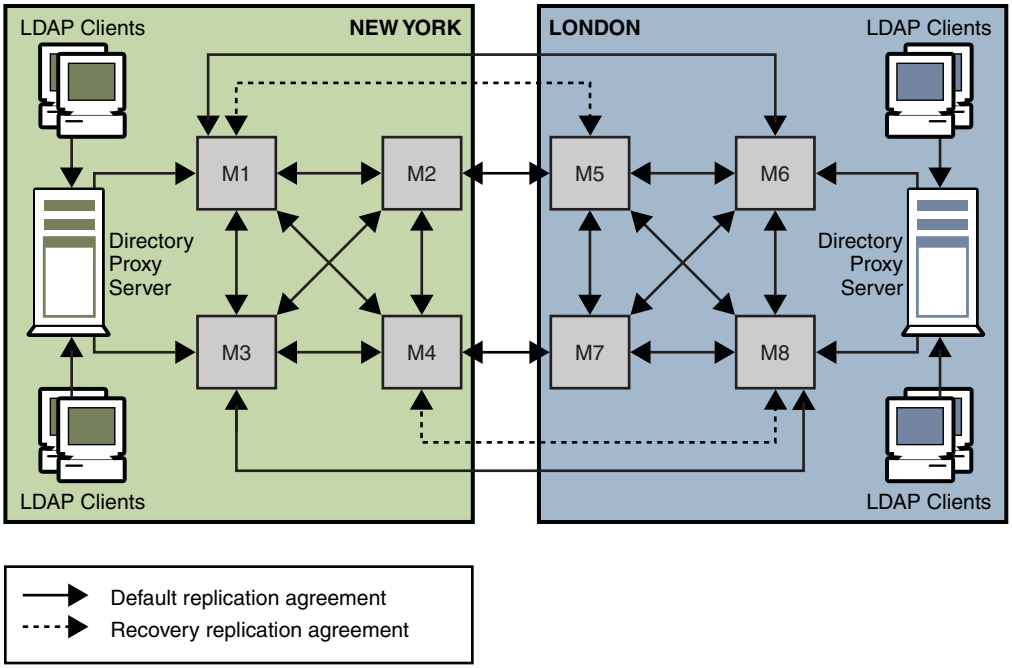



**FIGURE 12–3** Recovery Replication Agreements For Two Data Centers

The recovery strategy depends on which combination of components fails. However, after you have a basic strategy in place to cope with multiple failures, you can apply that strategy if other components fail.

In the sample topology depicted in Figure 12–3, assume that Master 1 and Master 3 in the New York data center fail.

In this scenario, Directory Proxy Server automatically reroutes reads and writes in the New York data center to Master 2 and Master 4. This ensures that local read and write capability is maintained at the New York site.

## Using Multiple Directory Proxy Servers

The deployment shown in the following figure includes an enterprise firewall that rejects outside access to internal LDAP services. Client LDAP requests that are initiated internally go through Directory Proxy Server by way of a network load balancer, ensuring high availability at the IP level. Direct access to the Directory Servers is prevented, except for the host that is running Directory Proxy Server. Two Directory Proxy Servers are deployed to prevent the proxy from becoming an SPOF.

A fully meshed multi-master topology ensures that all masters can be used at any time in the event of failure of any other master. For simplicity, not all replication agreements are shown in this diagram.

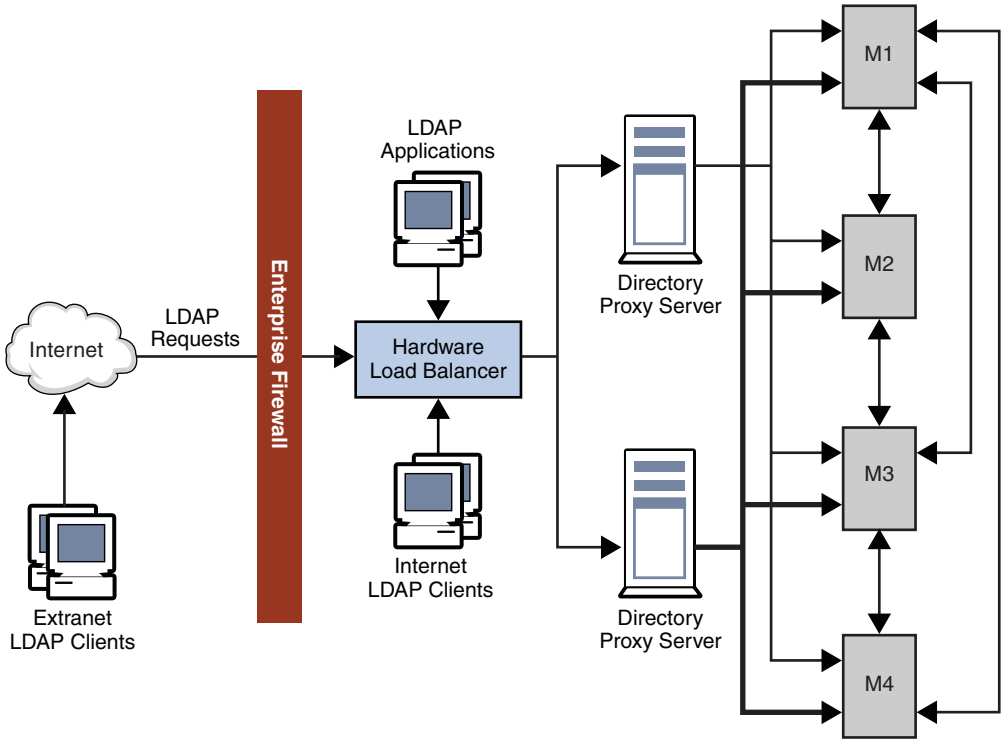



**FIGURE 12–4** Internal High Availability Configuration

## Using Application Isolation

In the scenario illustrated in the following figure a bug in Application 1 causes Directory Server to fail. The proxy configuration ensures that LDAP requests from Application 1 are only ever sent to Master 1 and to Master 3. When the bug occurs, Masters 1 and 3 fail. However, Applications 2, 3, and 4 are not disabled, because they can still reach a functioning Directory Server.

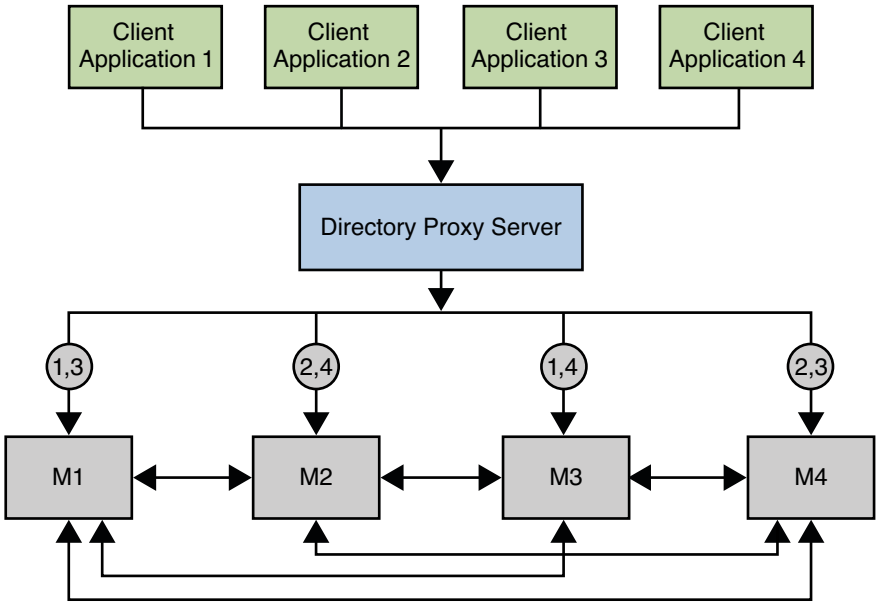



**FIGURE 12–5** Using Application Isolation in a Scaled Deployment

# Using Clustering for High Availability

From a physical perspective, a cluster consists of between one and eight servers that work together as a single entity. The servers work together to provide highly available access to applications, system resources, and data. Each server can be a symmetric multiprocessor with multiple CPUs.

A clustering solution can provide high availability for the following:

- Servers and software
- Storage subsystem
- Network adaptor

Clustering does not mitigate all SPOFs in a directory architecture. Failures in the external network, power generation, and data center must be mitigated outside of a clustering solution.

Using Sun Cluster 3.2 or 3.1 for directory service availability involves installing and configuring the Sun Cluster HA for Directory Server data service as a failover data service. This strategy allows Directory Server to fail over safely in a Sun Cluster environment.

The following figure shows the position of the Sun Cluster HA for Directory Server data service in the Sun Cluster architecture.

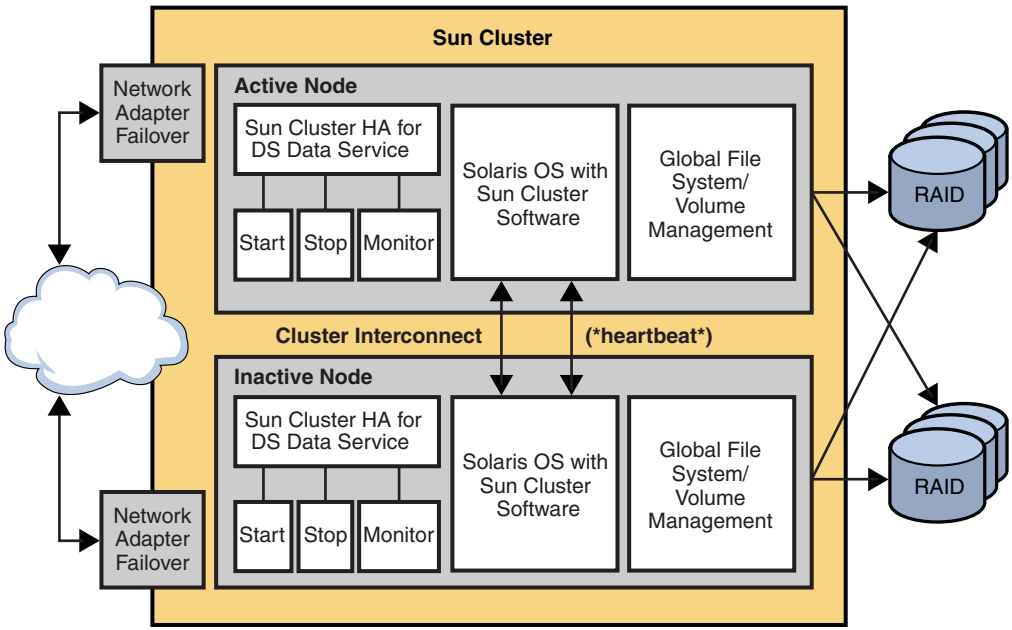



**FIGURE 12–6** Sun Cluster Architecture

## Hardware Redundancy

The architecture of a Sun Cluster hardware system is designed so that no SPOF can make a cluster unavailable. Redundant high-speed interconnects, storage system connections, and public networks ensure that cluster connectivity does not experience single failures.

Clients connect to the cluster through public network interfaces. If a network adapter card has multiple hardware interfaces, the card can connect to one or more public networks. You can set up nodes to include multiple network interface cards. The cards are configured so that one card is active, and the other cards operate as backups.

A cluster file system is a proxy between the kernel on one or more nodes and the underlying file system and volume manager. The cluster file system runs on a node that has a physical connection to the disks. For a cluster file system to be highly available, you must attach the disks to multiple nodes. A local file system that is made into a cluster file system is not highly available. A local file system implies a file system that is stored on a node's local disk.

A volume manager provides for mirrored or RAID 5 configurations for data redundancy of multihost disks. You can combine multihost disks with disk mirroring and striping to protect against both node failure and individual disk failure.

The cluster interconnect is a private network that transfers cluster-private communications and data service communications between cluster nodes. Redundant NICs, junctions, and cables protect against network failure.

## Monitoring in a Clustered Solution

The cluster continuously monitors all its members. It blocks failed nodes from participating in the cluster, which prevents any exchange of corrupt data. The cluster also monitors applications, and it fails over or restarts the applications in case of failures.

Public Network Management, a subsystem of the Sun Cluster software, monitors the active interface. If the active adapter fails, Network Adapter Failover software is called to fail over the interface to one of the backup adapters.

The Cluster Membership Monitor (CMM) is a distributed set of agents, with one set per cluster member or node. The agents exchange messages over the cluster interconnect to ensure full connectivity among all nodes. When the CMM detects a change in cluster membership because of a node failure, for example, the CMM reconfigures the cluster. If the CMM detects a critical problem with a node, the CMM contacts the cluster framework. The cluster framework then forcibly shuts down the node and removes it from the cluster membership.

## System Maintenance

You can minimize planned downtime for system maintenance by moving data and applications from the component that needs maintenance to another component on the system. When the maintenance is complete, you can move the data and applications back to the original component.

## Directory Server Failover Data Service

The Directory Server Failover Data Service runs on a single node in a cluster. However, nodes can have multiple CPUs for scalability. A fault monitor periodically monitors this failover service.

The Resource Group Manager (RGM) manages data services as resources. When a CMM changes a cluster's membership, the RGM might request changes to the cluster's online or offline resources. The RGM starts and stops failover data services.

## Disaster Recovery

The following sections describe how a service is recovered if the Directory Server Data Service fails and if the server fails.

## Recovery in the Event of Application Failure

If the fault monitor determines that the Directory Server Data Service has failed, the monitor initiates action to restart the service. The action that is taken depends on the service's configuration.

You can configure the failover data service to attempt to restart a failed service on the same node. Alternatively, the data service can be configured to immediately start a failed service on a different node. If the data service is configured to attempt to restart on the same node, the fault monitor contacts the local RGM. The local RGM then attempts to restart the failed service. If this action fails, the local RGM attempts to start the service on a different node.

If a failed data service cannot be restarted on the same node, the local node's RGM attempts to locate a version of the service on another node. This action also occurs if the data service is configured to start on a different node after failure. If the local RGM finds a version of the service, the local RGM contacts the local CMM and requests that it contact the remote node over the cluster interconnect. The remote CMM then contacts the local RGM and directs it to start the service.

The following figure illustrates recovery in the event of application failure.

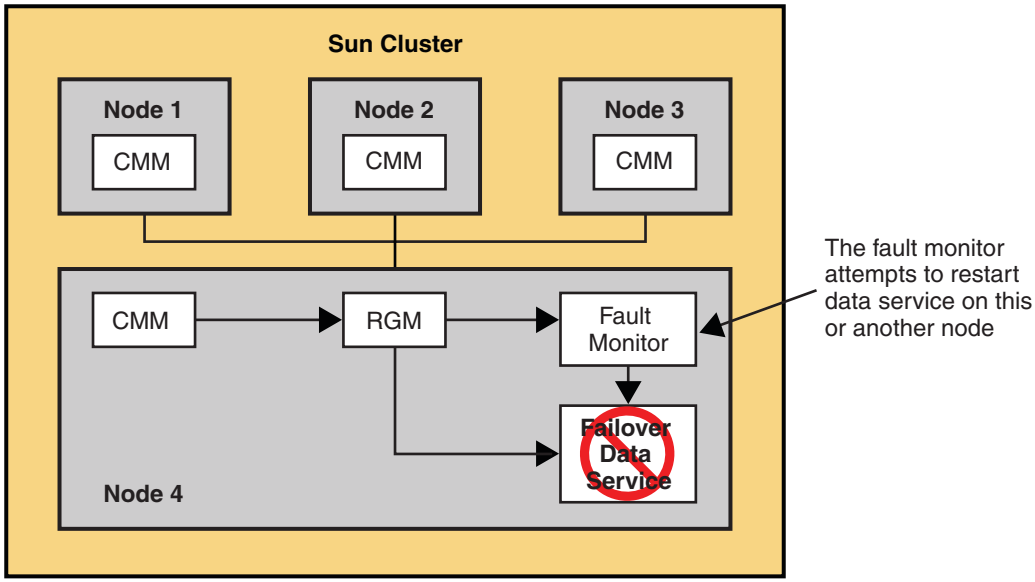



**FIGURE 12–7** Application Failure and Recovery in a Sun Cluster Architecture

## Recovery in the Event of Server Failure

If the server or node on which the Directory Server Data Service is running fails, the service is migrated to another working node. No user intervention is required. This service uses a failover resource group, a container that defines the Directory Server instances, and hosts that support the failover requirements.

The following figure illustrates recovery in the event of server failure.

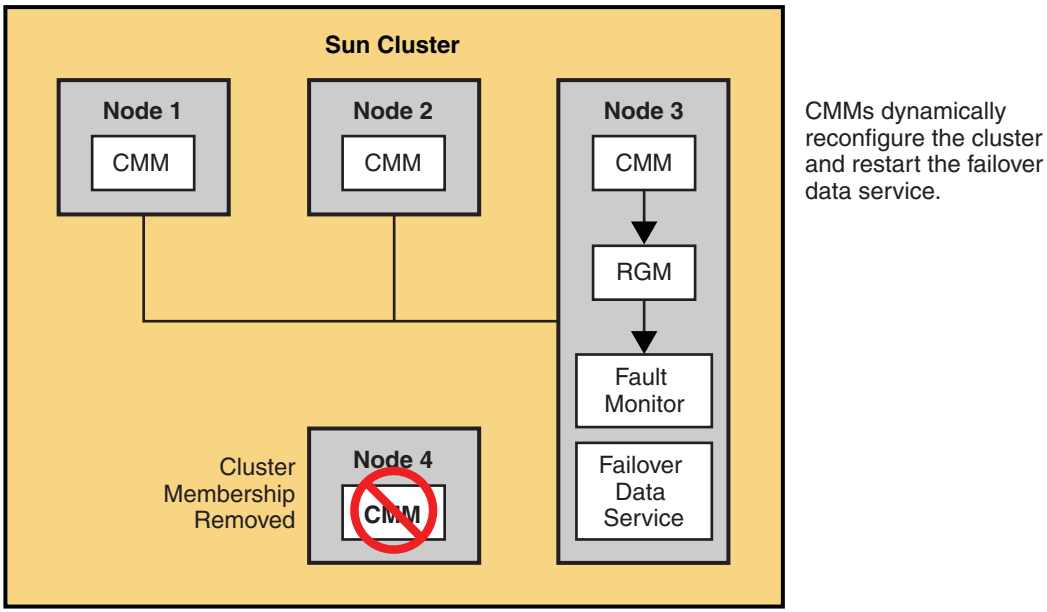



**FIGURE 12–8** Server Failure and Recovery in a Sun Cluster Architecture

PART IV

# Advanced Deployment Topics

This part discusses specialized deployment topics. It includes the following chapters:

- Chapter 13, "Using LDAP-Based Naming With Solaris," covers LDAP-Based Naming.
- Chapter 14, "Deploying a Virtual Directory," covers the virtualization functionality of Directory Proxy Server.
- Chapter 15, "Designing a Deployment With Synchronized Data," covers deployments that use Identity Synchronization for Windows.

◆ ◆ ◆ CHAPTER 13

# Using LDAP-Based Naming With Solaris

This chapter provides an overview of the LDAP naming service that is provided with the Solaris™ Operating System (Solaris OS). The naming services supported by the Solaris OS are described in detail in Part I, "About Naming and Directory Services," in *System Administration Guide: Naming and Directory Services (DNS, NIS, and LDAP)*.

This chapter covers the following topics:



## Why Use an LDAP-Based Naming Service?

A naming service stores information in a central place, which enables users, machines, and applications to communicate across the network. This information can include, for example, machine (host) names and addresses, user names, passwords, access permissions, group membership, and printers. Without a central naming service, each machine would have to maintain its own copy of this information. Naming service information can be stored in files, maps, or database tables. If you centralize all data, administration becomes easier.

The Solaris OS supports the following naming services:

- DNS, the Domain Name System
- `/etc` files, the original UNIX® naming system
- NIS, the Network Information Service
- NIS+, the Network Information Service Plus
- LDAP, the Lightweight Directory Access Protocol

However, Sun's strategic direction is to move to LDAP-based naming services.

The LDAP naming service has the following advantages over other naming services:

- Enables you to consolidate information by replacing application-specific databases, which reduces the number of distinct databases to be managed
- Allows data to be shared by different naming services
- Provides a central repository for data
- Allows for more frequent data synchronization between master servers and replicas
- Is multi-platform and multi-vendor compatible

The LDAP naming service has the following restrictions:

- Clients prior to Solaris 8 are not supported.
- Setting up and managing an LDAP naming service is more complex and requires careful planning.
- An NIS client and a Native LDAP client cannot coexist on the same client machine.

The Solaris OS supports LDAP naming in conjunction with Sun Java System Directory Server, as well as other LDAP directory servers. Although using Sun Java System Directory Server is recommended, it is not required.

# Migrating From NIS to LDAP

Moving from NIS to LDAP is a two-step process that involves data migration and client migration. The Solaris OS provides the NIS-to-LDAP transition service (N2L service), which accomplishes both steps.

The N2L service replaces existing NIS daemons on the NIS master server with NIS-to-LDAP transition daemons. The N2L service also creates an NIS-to-LDAP mapping file on that server. The mapping file specifies the mapping between NIS map entries and equivalent Directory Information Tree (DIT) entries in LDAP. An NIS master server that has gone through this transition is referred to as an N2L server.

The NIS slave servers continue to function in the usual manner. The slave servers periodically update their data from the N2L server as if the N2L server were a regular NIS master. A script, `inityp2l`, assists with the initial setup of these configuration files. When the N2L server has been established, you can maintain N2L by directly editing the configuration files.

The N2L service supports the following:

- Import of NIS maps into the LDAP DIT
- Client access to DIT information with the speed and extensibility of NIS

For details on how to migrate from NIS to LDAP, see Chapter 15, "Transitioning From NIS to LDAP (Overview/Tasks)," in *System Administration Guide: Naming and Directory Services (DNS, NIS, and LDAP)*.

# Migrating From NIS+ to LDAP

Although you can keep NIS+ data synchronized with LDAP, such synchronization has previously required an external agent. However, the NIS+ daemon now enables you to use an LDAP server as a data repository for NIS+ data. This feature enables NIS+ and LDAP clients to share the same naming service information. The transition from using NIS+ as the main naming service to using LDAP for the same role is therefore easier.

For details on how to migrate from NIS+ to LDAP, see Chapter 16, "Transitioning From NIS+ to LDAP," in *System Administration Guide: Naming and Directory Services (DNS, NIS, and LDAP)*.

◆ ◆ ◆ CHAPTER 14

# Deploying a Virtual Directory

The *virtual directory* is an advanced feature of Directory Proxy Server that aggregates information, in real time, from multiple data repositories. This chapter describes how you can use a virtual directory in a Directory Server Enterprise Edition deployment.

The architectural concepts of a virtual directory are described in Chapter 18, "Directory Proxy Server Virtualization," in *Sun Java System Directory Server Enterprise Edition 6.3 Reference*. Procedural information about setting up a virtual directory is provided in Chapter 23, "Directory Proxy Server Virtualization," in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

This chapter covers the following topics:

# When to Use a Virtual Directory

Virtual directory features can be deployed if your directory service has any of the following requirements:

- Client applications require an aggregated view of entries across multiple data repositories.

  For example, you might have several directory servers that contain the same users, but different data. The virtual directory can be used to create a single view of a user's entry across all directories. The virtual directory can also provide a single point of administration for each individual directory.

  Types of data repositories that are supported include LDAP directories, Java Database Connectivity (JDBC™) compliant sources such as MySQL, and LDIF flat files.

- Separate data stores are required for user credentials and application specific data.

  For example, an application might have specific data that you do not want to be stored in a corporate directory. The virtual directory enables you to separate the data but make it appear as one source for applications. This simplifies application development and data management because applications do not need to know the details of the data infrastructure. In addition, changes to backend data sources can be abstracted from applications.

- Your enterprise has acquired another company, or merged with another company.

  The virtual directory enables the two company directories to be merged so that they appear as a single directory. For example, imagine you have two directories, `dc=example,dc=com` and `dc=acquisition,dc=com`. You also have client applications that need both directories to look like `dc=example,dc=com`.

- Client applications require database tables to be displayed in the format of a DIT hierarchy.
- Read and write operations are required to multiple data repositories.
- Multiple field join criteria with dissimilar attribute names are required.
- Client applications require support for multi-valued attributes across directories and databases from multiple LDAP or JDBC backends.
- Attribute renaming, DN rewriting, and attribute value rewriting for DN syntax attributes are required.
- Multiple client applications require different views of a single data repository.

# Typical Virtual Directory Scenarios

This section provides simple scenarios that show how a virtual directory answers specific business requirements. For more complex sample scenarios, and for details of virtual directory configuration, see “Sample Virtual Configurations” in *Sun Java System Directory Server Enterprise Edition 6.3 Administration Guide*.

# Connecting User Identities From Different Data Sources

Example.com stores uses three different data repositories to store user data. Example.com's Directory Server contains the bulk of the user data. User email addresses are stored in an Active Directory, and HR data is stored in a MySQL database.

Example.com has several client applications that require a complete view of all user data. The following diagram illustrates how the virtual directory provides a complete view of a user's identity to the client application.

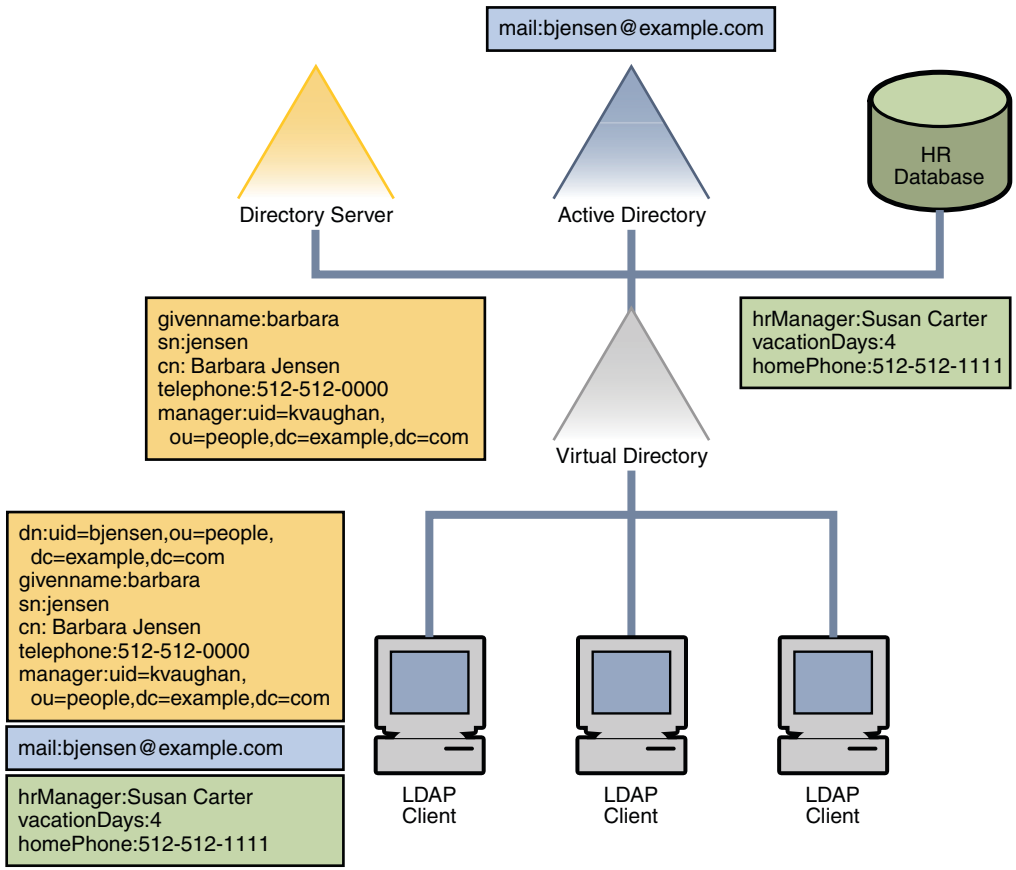



**FIGURE 14–1** Virtual View of Aggregated Data From Multiple Repositories

# Merging New Corporate Data Into an Existing Directory Structure

In this scenario, Example.com acquires a new company, Acquisition.com. The new company stores its user data in its own Directory Server. For management purposes, Example.com wants to retain this directory structure. However, certain client applications need to see the use data from Acquisition.com *as if it were* user data from Example.com.

The following diagram illustrates how the virtual directory provides a virtualized merge of the acquired company's data into the existing directory structure.

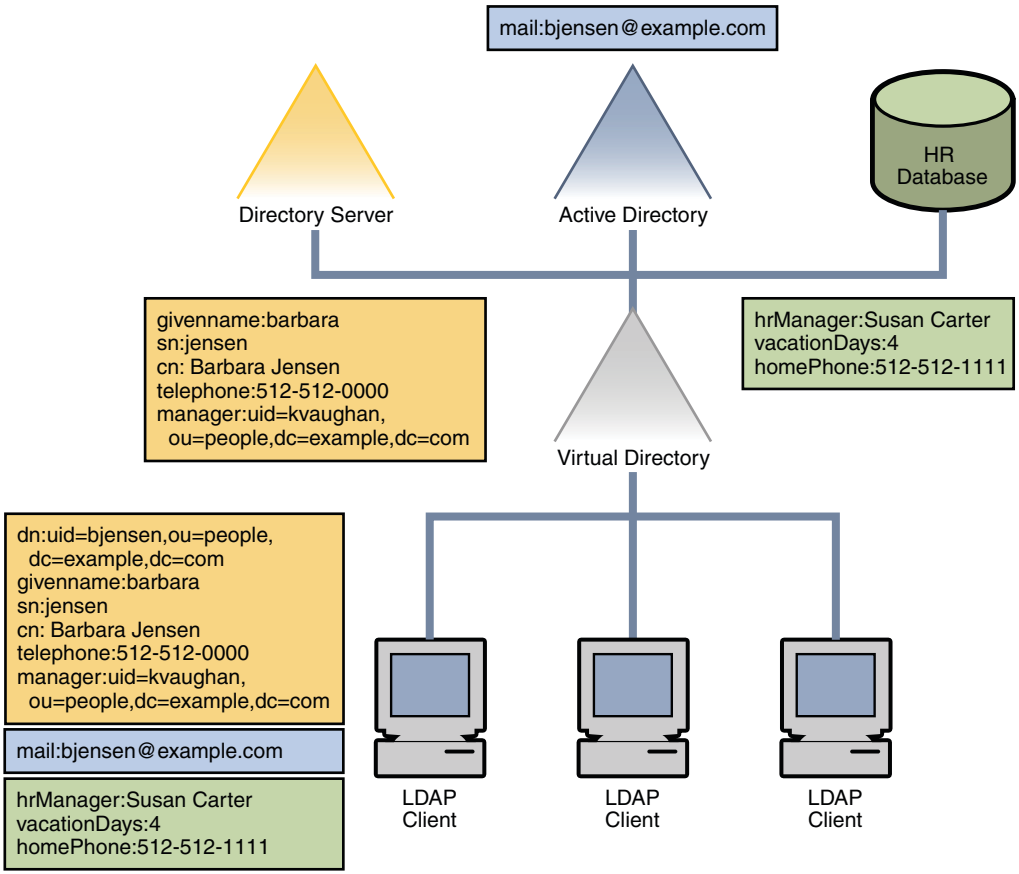



**FIGURE 14–2** Merging User Data From Acquired Directory

Acquisition.com's directory is seen as a separate branch under the `ou=people` branch. The DNs of the entry's in Acquisition.com's directory are transformed when they are viewed through the virtual directory.

◆ ◆ ◆ CHAPTER 15

# Designing a Deployment With Synchronized Data

Identity Synchronization for Windows is a component of Directory Server Enterprise Edition that synchronizes user account information, including passwords, between Directory Server and Windows. Both Windows Active Directory and Windows NT are supported. Identity Synchronization for Windows helps build a scalable and security-enriched password synchronization solution for an enterprise of any size.

For complete documentation on Identity Synchronization for Windows, see `http://docs.sun.com/coll/isw_04Q3`. If you are planning to use Identity Synchronization for Windows in your deployment, you must address the issues that are described in this chapter.

## Identity Synchronization for Windows Deployment Considerations

- **Synchronization direction of passwords.** If passwords are synchronized from Directory Server to Active Directory or in both directions, install the High Encryption Pack on Windows 2000. This installation enables 128-bit SSL, which is required when setting passwords in Active Directory over LDAP.
- **Synchronizing the creation of new users.** If Identity Synchronization for Windows does not synchronize the creation of new users, you must run the `idsync resync` command periodically to establish links between newly created users. Changes to newly created users are not synchronized until the users are explicitly linked by running `idsync resync`.
- **Population size.** While Identity Synchronization for Windows places no upper limit on the number of users that can be synchronized, the total number of users impacts the deployment. The primary impact is on the `idsync resync` command that must be run before synchronization is started. If more than 100,000 users are synchronized, run the `idsync resync` command in batches. This batch mode ensures optimal performance and limits the load on Sun Java System Message Queue.

- **Performance requirements.** The performance of Identity Synchronization for Windows is limited more by the synchronization rate than by the total number of users. The only exception to this requirement is when you run the `idsync resync` command.
- **Expected peak modification rate.** An Identity Synchronization for Windows deployment with a Core and two connectors that are running on the same system can easily sustain a modification rate of 10 synchronizations per second. If the required synchronization rate exceeds this rate, higher performance is achieved by distributing Identity Synchronization for Windows across multiple machines. For example, the connectors can be installed on a separate machine from the Identity Synchronization for Windows Core.
- **Number of Windows domains to be synchronized.** If more than one Windows domain is to be synchronized, the `activedirectorydomainname` attribute or the `USER_NT_DOMAIN_NAME` attribute must be synchronized to a Directory Server attribute. This synchronization is required to resolve ambiguity between Synchronization User List definitions.
- **Number of Directory Server masters, hubs, and read-only replicas in the deployment.** In a deployment with multiple Directory Servers, the Identity Synchronization for Windows Directory Server plug-in must be enabled on each master, each hub, and each read-only replica. When configuring Identity Synchronization for Windows, one Directory Server master is designated as the preferred master. The Directory Server connector detects and applies changes at the preferred master while the master is running. If this server is down, the connector can optionally apply changes at a second master. The Retro Changelog plug-in must be enabled on the preferred master. This master should be on the same LAN as the Identity Synchronization for Windows Core.
- **Security.** If the Directory Server or the Active Directory connectors connect to Directory Server or Active Directory over SSL, SSL must be enabled on these servers. If the connectors are configured to accept only trusted certificates, extra configuration steps must be taken. These steps import the appropriate Certificate Authority certificates into the connectors' certificate databases. If SSL is required between the Directory Server plug-in and Active Directory, SSL must be enabled in Directory Server. In addition, the Certificate Authority certificate that is used to sign the Active Directory SSL certificate must be imported into the Directory Server's certificate database.

For detailed deployment scenarios that incorporate Identity Synchronization for Windows, see *Sun Java System Identity Synchronization for Windows 6.0 Deployment Planning Guide*.

# Index

## P



## R



## S



## T



## V