Sun Java™ System

Directory Server 5.2 Performance Tuning Guide

2005Q1

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

Part No: 817-7609-10

Contents

List of Tables

List of Figures

Preface

This guide contains the information you need in order to carry out post-installation configuration and tuning. While Directory Server offers great performance out of the box, you can tune it for even better performance, sometimes with spectacular results.

For information about how to access Sun™ documentation and how to use Sun documentation, see the following sections:

- Conventions
- Related Books
- Documentation, Support, and Training
- Related Third-Party Web Site References
- Sun Welcomes Your Comments

Conventions

Table 1 describes the typeface conventions used in this document.

Table 1 Typeface Conventions

Typeface	Meaning	Examples
`AaBbCc123` (Monospace)	API and language elements, HTML tags, web site URLs, command names, file names, directory path names, on-screen computer output, sample code.	Edit your `.login` file. Use `ls -a` to list all files. `% You have mail.`
`AaBbCc123` (Monospace bold)	What you type, as contrasted with on-screen computer output.	`%` **`su`** `Password:`

Table 1 Typeface Conventions *(Continued)*

Typeface	Meaning	Examples
AaBbCc123	Book titles.	Read Chapter 6 in the *Developer's Guide*.
(Italic)	New words or terms.	These are called *class* options.
	Words to be emphasized.	You *must* be superuser to do this.
	Command-line variables to be replaced by real names or values.	The file is located in the *ServerRoot* directory.

Table 2 describes placeholder conventions used in this guide.

Table 2 Placeholder Conventions

Item	Meaning	Examples
install-dir	Placeholder for the directory prefix under which software binaries reside after installation.	The default *install-dir* prefix on Solaris systems is `/`. The default *install-dir* prefix on Red Hat systems is `/opt/sun`.
ServerRoot	Placeholder for the directory where server instances and data reside. You can manage each server under a *ServerRoot* remotely through your client-side Server Console. The Server Console uses the server-side Administration Server to perform tasks that must execute directly on the server-side system.	The default *ServerRoot* directory is `/var/opt/sun/serverroot.`
`slapd-`*serverID*	Placeholder for the directory where a specific server instance resides under the *ServerRoot* and its associated data resides by default.	The default *serverID* is the host name.

Table 3 describes the symbol conventions used in this book.

Table 3 Symbol Conventions

Symbol	Meaning	Notation	Example
`[ ]`	Contain optional command options.	`O[`*n*`]`	`O4, O`
`{ }`	Contain a set of choices for a required command option.	`d{y|n}`	`dy`
`|`	Separates command option choices.		

Table 3 Symbol Conventions *(Continued)*

Symbol	Meaning	Notation	Example
+	Joins simultaneous keystrokes in keyboard shortcuts that are used in a graphical user interface.		Ctrl+A
-	Joins consecutive keystrokes in keyboard shortcuts that are used in a graphical user interface.		Esc-S
>	Indicates menu selection in a graphical user interface.		File > New File > New > Templates

Table 4 describes the shell prompt conventions used in this book.

Table 4 Shell Prompts

Shell	Prompt
C shell	*machine-name*%
C shell superuser	*machine-name*#
Bourne shell and Korn shell	$
Bourne shell and Korn shell superuser	#

Input and output of Directory Server commands are usually expressed using the LDAP Data Interchange Format (LDIF) [RFC 2849]. Lines are wrapped for readability.

Related Books

The following books can be found in HTML and PDF at
`http://www.sun.com/documentation/`.

Directory Server Books

Directory Server Release Notes

Directory Server Technical Overview

Directory Server Deployment Planning Guide

Directory Server Installation and Migration Guide

Directory Server Performance Tuning Guide

Directory Server Administration Guide

Directory Server Administration Reference

Directory Server Plug-in Developer's Guide

Directory Server Plug-in Developer's Reference

Directory Server Man Page Reference

Administration Server Books

Administration Server Release Notes

Administration Server Administration Guide

Administration Server Man Page Reference

Directory Proxy Server Books

Directory Proxy Server Release Notes

Directory Proxy Server Administration Guide

Related Java Enterprise System Books

Java Enterprise System Installation Guide

Java Enterprise System Upgrade and Migration Guide

Java Enterprise System Glossary

Documentation, Support, and Training

Table 5 provides links to Sun documentation, support, and training information.

Table 5 Documentation, Support, and Training links

Typeface	Meaning	Examples
Documentation	`http://www.sun.com/documentation/`	Download PDF and HTML documents, and order printed documents.
Support and Training	`http://www.sun.com/supportraining/`	Obtain technical support, download patches, and learn about Sun courses.

Related Third-Party Web Site References

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

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. Use the web-based form to provide feedback to Sun:

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

Please provide the full document title and part number in the appropriate fields. The part number is a seven-digit or nine-digit number that can be found on the title page of the book or at the top of the document. For example, the part number of this document is 817-7609-10.

Chapter 1

Top Tuning Tips

Tuning performance implies modifying the default configuration to reflect specific deployment requirements.

This guide describes how to tune a single Directory Server instance. It is assumed here that your overall directory service design including the replication topology is complete, and that you use the information here to tune the Directory Server instances to meet the design requirements. If you have not yet completed the overall directory service design, refer to the *Directory Server Deployment Planning Guide* for suggestions on how to do so.

Tuning performance takes time, effort, and thought as reflected in Table 1-1.

Table 1-1 Tuning Process

Phase	Description
Define goals	Define specific, measurable objectives for tuning, based on deployment requirements. Consider questions such as: • Which applications use Directory Server? • Is the system dedicated to Directory Server? Does it run other applications? If so, which other applications? • How many *entries* does the deployment call for? How large are such entries? • How many *searches* per second must the Directory Server support? What types of searches are expected? • How many *updates* per second must the Directory Server support? What types of updates are expected? • What sort of peak update and search rates are expected? What sort of average rates are expected? • Does the deployment call for repeated bulk import initialization on this system? If so, how often are imports performed? How many entries are imported at a time? What types of entries? Must initialization be performed online with the server running? This list is not exhaustive. Ensure yours is.
Select methods	Determine how you plan to implement tuning optimizations and how you plan to measure and analyze them. Can you change the hardware configuration of the system? Are you limited to using existing hardware, tuning only the underlying operating system and Directory Server itself? 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 planned. For large and complex deployments, this phase may take considerable time.
Verify results	Check whether the potential optimizations tested reach the goals defined at the outset of the process. If they reach the goals, document the results. If they do not reach the goals, profile and monitor the Directory Server you are tuning.
Profile and monitor	Profile and monitor the behavior of Directory Server after applying the potential modifications. Collect measurements of all relative behavior.

Table 1-1 Tuning Process *(Continued)*

Phase	Description
Plot and analyze	Plot and analyze the behavior observed while profiling and monitoring. Attempt to find evidence and patterns that suggest further tests. You may 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	Once the optimizations applied reach the goals defined at the outset of the process, document them well so they can be easily reproduced.

This chapter lists basic recommendations that apply almost every time you tune a Directory Server instance. Although the recommendations presented here are in general valid, avoid the temptation to apply them without understanding how they impact the specific deployment at hand. This chapter 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, and an appropriate amount of extra available physical memory to account for future growth. In the case where plenty of physical memory is available, set the entry cache size large enough to hold all entries in the directory, and set the database cache size large enough to hold all indexes including the content of all `*_id2entry.db3` files.

   Refer to Chapter 3, "Tuning Cache Sizes," for more information.

2. Optimize indexing.

   a. Remove unnecessary indexes and add additional indexes to support expected requests.

      From time to time, it may become necessary to add additional indexes that support requests from new applications. It is possible to add, remove, and modify indexes while Directory Server is running. Directory Server gradually indexes data after you make changes to the indexes. You can also force Directory Server to rebuild indexes.

      Refer to "Benefits: How Searches Use Indexes" on page 48 and "Costs: How Updates Affect Indexes" on page 49 for more information.

b. Allow only indexed searches.

Unindexed searches can have a strong negative impact on server performance and may consume significant server resources. Consider adding indexes to support specific searches applications may perform, and forcing the server to reject unindexed searches.

Refer to "Allowing Only Indexed Searches" on page 59 for more information.

c. Adjust the maximum length of index lists.

Refer to "Limiting Index List Length" on page 59 for more information.

3. Tune the underlying operating system.

Refer to Chapter 2, "Tuning the Operating System," for more information.

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.

Refer to Chapter 6, "Managing Use of Other Resources," for more information.

5. Distribute disk activity.

Especially for deployments supporting large numbers of updates, Directory Server can be extremely disk I/O intensive. If possible, consider spreading the load across multiple disks using separate controllers.

6. Disable unnecessary logging.

Disk access being slower than memory access, heavy logging can 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, and leave error logging at a minimal level when not using the error log to troubleshoot problems. You may also reduce the impact of logging by putting log files on a lesser used disk, such as the disk used for the replication changelog.

Refer to Chapter 5, "Tuning Logging," for more information.

7. When replicating large numbers of updates, consider adjusting network configuration parameters related to replication as described in the *Directory Server Administration Guide*.

8. (Solaris systems) Move the database home directory to a `tmpfs` file system.

   The database home directory, specified by `nsslapd-db-home-directory`, indicates where Directory Server locates database cache backing files. (Data files continue to reside by default under *ServerRoot*/`slapd-`*serverID*/`db`.) By placing the database cache backing files on a `tmpfs` file system, you keep the system from flushing the database cache backing files to disk from time to time, therefore avoiding a performance bottleneck for updates, and in some cases even for searches. As 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 or reboot. This cost is probably not one you can avoid, however, if you expect a restart or reboot to happen only after a crash. After a crash, the database cache has to be rebuilt anyway.

9. Enable transaction batching if you can afford to lose updates during a crash.

NOTE With transaction batching enabled, you lose up to (`nsslapd-db-transaction-batch-val - 1`) updates during a crash, because Directory Server waits for the batch to fill (or 1 second, whichever is sooner) before flushing content to the transaction log.

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

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

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. Refer to "Configuring Server Plug-Ins" on page 79 for details.

Chapter 2

Tuning the Operating System

Default system and network settings may not be ideal for top directory service performance. When tuning the underlying operating system for optimum Directory Server performance, you should check that the latest recommended patches are installed on the system, enforce basic security measures, and adjust some system and network settings. This chapter addresses those tuning options.

The `idsktune` utility (`directoryserver -u 5.2 idsktune`) provided with the product may help you to diagnose basic system configuration shortcomings. The utility offers system tuning recommendations for support of high performance directory services. The utility does not actually implement any of the recommendations made. Tuning recommendations should be implemented by a qualified system administrator.

This chapter covers the following topics:

Checking Platform Support

Refer to the *Directory Server Release Notes* for an updated list of supported platforms.

Patching the System

In order to maintain overall system security, and to ensure proper installation and operation of Directory Server 5.2, install the latest recommended system patches, service packs, or fixes. Table 2-1 suggests where to look for required patches.

Table 2-1 Where to Obtain Patches, By Platform

Platform	Browse...
Sun Solaris™ Operating System	`http://sunsolve.sun.com/`
Red Hat Linux	`http://www.redhat.com/`

Enforcing Basic Security

The recommendations in this section do not eliminate all risk. Instead, they are intended as a short checklist to help you work to limit some of the most obvious security risks.

Isolate the System

If at all possible, isolate the system running Directory Server from the public Internet using a network firewall.

No Dual Boot

Do not dual boot or run other operating systems on the system running Directory Server. Other systems may permit access to otherwise restricted files.

Strong Passwords

Use a super user password at least 8 characters long that includes punctuation or other non-alphabetic characters.

If you choose to use longer operating system passwords, it may be necessary to configure the way passwords are handled by the system. Refer to the operating system documentation for instructions.

Users and Groups

For security reasons, it is recommended not to run Directory Server or the associated Administration Server with super user privileges. You may, for example, use the commands `groupadd` and `useradd` to create a user and group without login privileges, and then install and run the servers as this user and group.

For example, to add a group named "servers":

```
groupadd servers
```

To add a user named "server1" as a member of the group "servers":

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

Both Directory Server and the associated Administration Server must run as the same user and group (that is, they must run with the same uid and gid).

To facilitate debugging, you may choose to allow processes running with this user and group identity to dump core, using utilities such as `coreadm`(1M) on Solaris systems. For example, you can enable Directory Server running as this user and group to generate core files by allowing `setuid` processes to do so, and updating the `coreadm` configuration:

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

Furthermore, by adding the following line near the top of the *ServerRoot*`/slapd-`*serverID*`/start-slapd` script, you allow Directory Server to generate core files of the form `core.ns-slapd.`*pid*, where *pid* is the process ID:

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

If a particular deployment calls for sharing Directory Server files with other servers such as a messaging server, consider running those servers using the same user and group.

If you must run the Administration Server as super user, consider stopping the service when not using it. You must start the Administration Server again before you can use the Console, which depends on Administration Server running.

Disabling Unnecessary Services

For top performance and less risk, dedicate the system to Directory Server alone. Running additional services, especially network services, negatively affects server performance and scalability, and may increase security risks.

Disable as many network services as possible. Directory Server uses only TCP/IP, and UDP/IP if you run SNMP agents. It 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 in that they transmit user passwords in clear text over the network. You may be able to work around the need `telnet` and `ftp` by using clients such as Secure Shell (`ssh`) and Secure FTP (`sftp`) instead.

If the Directory Server instance does not itself provide the naming service for the network, consider enabling a naming service for the system, which Directory Server then uses for example when resolving ACIs. Remote administration tools such as Sun Java System Server Console rely on the naming service for some aspects of their operation such as translating between IP addresses and host names.

Refer to the operating system documentation for details on disabling network services.

Keeping Accurate Time

Ensure the system clock is reasonably in sync with those of other systems to facilitate replication and 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, for example.

Restarting After System Failure

When possible, stop Directory Server as described in the *Directory Server Administration Guide*. Database corruption may cause Directory Server to start slowly if stopped abruptly during system shutdown, rather than shut down appropriately. Time may be needed to recover the database.

Consider using a logging option with your file system, which generally both improves write performance and decreases the time required to perform a file system check when the file system is not cleanly unmounted as is typically the case during a crash. Also consider using RAID, as described in the *Directory Server Administration Guide*.

As part of the installation and configuration process, appropriate scripts enable restart at boot time.

For other platforms, refer to the operating system documentation for details on starting services at boot time.

Generating Basic Tuning Recommendations

Use the `idsktune` utility (`directoryserver -u 5.2 idsktune`) to generate basic tuning recommendations.

When you run the utility as super user, it gathers information about the system. It displays notices, warnings, and errors with recommended corrective actions. For example, the utility checks that:

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

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

Individual deployment requirements may exceed minimum requirements. You may opt to provide more resources than those identified as minimum system requirements by the `idsktune` utility.

For information about the `idsktune` command, see the *Directory Server Resource Kit Tools Reference*.

Tuning System Settings

You may use the `idsktune` tool that reads current system settings and recommends changes. In general, implementing the recommendations optimizes performance both on systems dedicated to running Directory Server and on systems running additional applications.

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

Table 2-2 Configuration Files to Check Prior to Deployment

Platform	File	Remarks
Solaris Operating System	`/etc/init.d/inetinit`	Add `ndd` statements for tuning
	`/etc/system`	Check system tuning
	`/etc/vfstab`	Ensure files are local
Red Hat Linux	`/etc/fstab`	Ensure files are local
	`/etc/security/limits.conf`	Add `nofile` hard limit directive
	`/etc/sysctl.conf`	Check, set kernel parameters
	`/proc/sys/fs/file-max`	Check file descriptor limits

File Descriptors

Directory Server uses file descriptors when handling concurrent client connections. Having a low maximum number of file descriptors available in the system or available to a process can thus limit the number of concurrent connections. Recommendations concerning the number of file descriptors therefore relate to the number of concurrent connections Directory Server may be able to handle on the system.

On Solaris systems, the number of file descriptors available is configured through the `rlim_fd_max` parameter, as described in the output of `directoryserver -u 5.2 idsktune`. Refer to the operating system documentation for further instructions on modifying the number of available file descriptors.

After modifying the maximum number of available file descriptors on the system, refer to Table 6-2 on page 76 for information on configuring Directory Server to use the available file descriptors.

Transmission Control Protocol (TCP) Settings

Specific network settings depend on the platform. On some systems, it is possible to enhance Directory Server performance by modifying TCP settings. This section discusses the reasoning behind `idsktune` recommendations concerning TCP settings.

Closed Connections in the TIME-WAIT State

Some systems allow you to configure how long a TCP connection is held in the kernel table after closure. While the connection is held, it may be opened again quickly. When set too high, the system may track a large number of connections in the kernel table over long intervals, reducing the number of available connections to Directory Server. For most deployments, setting this parameter to a value of 30 seconds (30,000 milliseconds) allows more concurrent connections to Directory Server.

On Solaris systems, this time interval is configured through the `tcp_time_wait_interval` parameter, as described in the output of `directoryserver -u 5.2 idsktune`.

Connections Pending Acceptance

Some systems allow you to configure the number of TCP connections pending acceptance by a TCP listener such as Directory Server. When set too low, this limits the number of pending connections Directory Server can accept. For most deployments, setting this parameter to a value of at least 1024 allows Directory Server to handle more concurrent connection requests.

On Solaris systems, the number of pending connections allowed is configured through the `tcp_conn_req_max_q0` parameter, as described in the output of `directoryserver -u 5.2 idsktune`. Consider increasing `tcp_conn_req_max_q0` to 2048.

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 may cause the system to use unnecessary resources keeping connections alive for clients that have become disconnected. For most deployments, setting this parameter to a value of 600 seconds (600,000 milliseconds = 10 minutes) allows more concurrent connections to Directory Server.

On Solaris systems, this time interval is configured through the `tcp_keepalive_interval` parameter, as described in the output of `directoryserver -u 5.2 idsktune`.

Incoming Connections

Some systems allow you to configure how long a system waits for an incoming connection not sending acknowledgements. When set too high, this can cause long delays in detecting connection failure. For intranet deployments on fast and reliable networks, setting this parameter to a value of 600 seconds (600,000 milliseconds = 10 minutes) may improve performance.

On Solaris systems, this time interval is configured through the `tcp_ip_abort_interval` parameter, as described in the output of `directoryserver -u 5.2 idsktune`.

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 and reliable networks, setting this parameter to a value of 10 seconds may improve performance.

On Solaris systems, this time interval is configured through the `tcp_ip_abort_cinterval` parameter, as described in the output of `directoryserver -u 5.2 idsktune`.

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 may be kept waiting on lost packets. For intranet deployments on fast and reliable networks, setting this parameter to a value of 500 milliseconds may improve performance.

On Solaris systems, this time interval is configured through the `tcp_rexmit_interval_initial` parameter, as described in the output of `directoryserver -u 5.2 idsktune`.

Sequence Numbers

Some systems allow you to configure how the system handles initial sequence number generation. For extranet and Internet deployments, set this parameter such that initial sequence number generation is based on RFC 1948 to prevent sequence number attacks.

On Solaris systems, this behavior is configured through the `tcp_strong_iss` parameter, as described in the output of `directoryserver -u 5.2 idsktune`.

Chapter 3

Tuning Cache Sizes

Directory Server caches directory information in memory and on disk in order to be able to respond more quickly to client requests. Properly tuned caching minimizes the need to access disk subsystems when handling client requests.

NOTE Unless caches are tuned and working properly, other tuning may have only limited impact on performance.

This chapter covers the following topics:

Types of Cache

Directory Server handles three types of cache as described in Table 3-1.

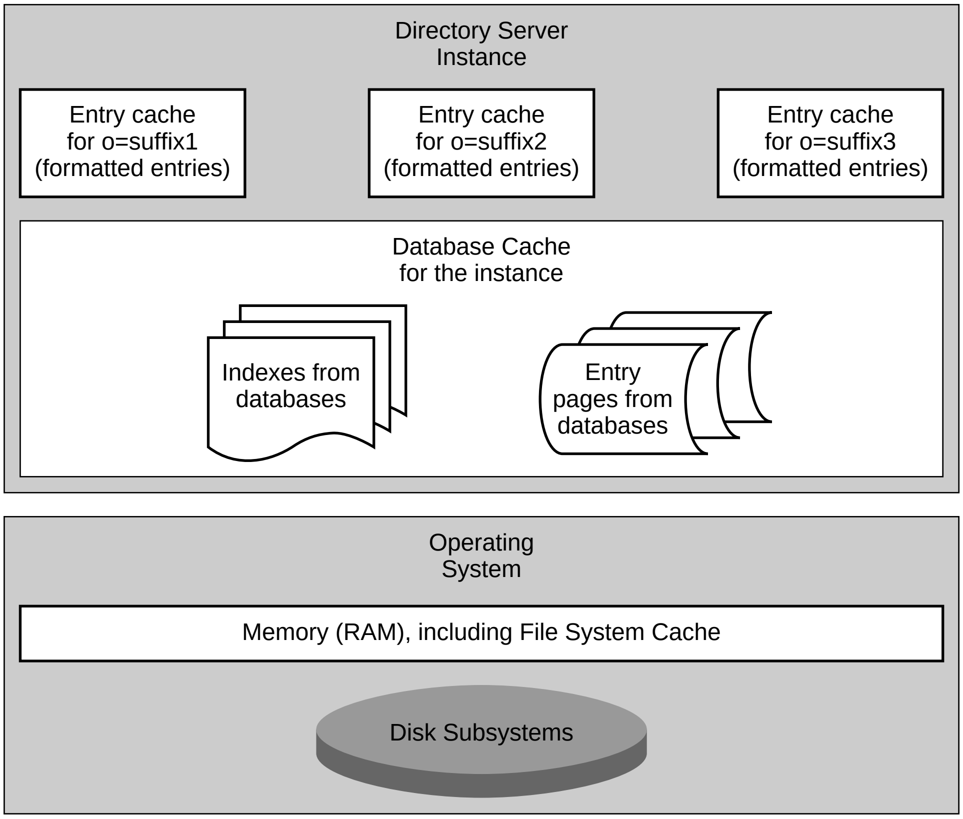
Table 3-1 Caches

Cache Type	Description
Database	Each Directory Server instance has one database cache that holds both indexes and entries in database format. Refer to "Database Cache" on page 33 for more information.
Entry	Each suffix has an entry cache that holds entries retrieved from the database during previous operations and formatted for quick delivery to client applications. Refer to "Entry Cache" on page 34 for more information.
Import	Each Directory Server instance has an import cache that is structurally similar to the database cache and is used during bulk loading. Refer to "Import Cache" on page 35 for more information.

Directory Server also benefits from file system cache, handled by the underlying operating system, and from I/O buffers in disk subsystems.

Figure 3-1shows caches for an instance of Directory Server handling three suffixes, each with its own entry cache. The instance is configured to handle significant disk activity.

Figure 3-1 Entry and Database Caches in Context


Directory Server Instance
Entry cache for o=suffix1 (formatted entries)
Entry cache for o=suffix2 (formatted entries)
Entry cache for o=suffix3 (formatted entries)
Database Cache for the instance
Indexes from databases
Entry pages from databases
Operating System
Memory (RAM), including File System Cache
Disk Subsystems

Database Cache

Each Directory Server instance has one database cache. The database cache holds pages from the database containing indexes and entries. Each page is not an entry, but a slice of memory containing a portion of the database. You specify database cache size (`nsslapd-dbcachesize`) in bytes. The change to database cache size takes effect after you restart the server, with database cache space allocated at server startup.

Directory Server moves pages between the database files and the database cache to maintain maximum database cache size. The actual amount of memory used by Directory Server for database cache may be up to 25 percent larger than the size you specify, due to additional memory needed to manage the database cache itself.

When using a very large database cache, verify through empirical testing and by monitoring memory use with tools such as `pmap`(1) on Solaris systems that the memory used by Directory Server does not exceed the size of available physical memory. Exceeding available physical memory causes the system to start paging repeatedly, resulting in severe performance degradation.

The `ps`(1) utility can also be used with the `-p` *pid* and `-o` *format* options to view current memory used by a particular process such as Directory Server (`ns-slapd`). Refer to the operating system documentation for details.

For 32-bit servers, database cache size must be limited such that the total Directory Server (`ns-slapd`) process size is less than the maximum process size allowed by the operating system. In practice, this limit is generally in the 2-3 GB range.

Refer to the *Directory Server Administration Reference* for further information about the valid range of `nsslapd-dbcachesize` values.

Entry Cache

The entry cache holds recently accessed entries, formatted for delivery to client applications. You specify entry cache size for a suffix (`nsslapd-cachememsize`) in bytes. Entry cache is allocated as needed.

Directory Server can return entries from an entry cache extremely efficiently, as entries stored in this cache are already formatted. Entries in the database must be formatted (and stored in the entry cache) before delivery to client applications.

When specifying entry cache size, know that `nsslapd-cachememsize` indicates how much memory Directory Server requests from the underlying memory allocation library. Depending on how the memory allocation library handles such requests, actual memory used may be much larger than the effective amount of memory ultimately available to Directory Server for the entry cache.

Actual memory used by the Directory Server process depends primarily on the memory allocation library used, and on the entries cached. Entries with many small attribute values usually require more overhead than entries with a few large attribute values.

For 32-bit servers, entry cache size must be limited such that the total Directory Server (`ns-slapd`) process size is less than the maximum process size allowed by the operating system. In practice, this limit is generally in the 2-3 GB range.Refer to the *Directory Server Administration Reference* for further information about the valid range of `nsslapd-cachememsize` values.

Import Cache

The import cache is created and used during suffix initialization only, also known as bulk loading or importing. If the deployment involves *offline* suffix initialization only, import cache and database cache are not used together, so you need not add them together when aggregating cache size as described in "Total Aggregate Cache Size" on page 36. You specify import cache size (`nsslapd-import-cachesize`) in bytes. Changes to import cache size take effect the next time the suffix is reset and initialized, with import cache allocated for the initialization, then released after the initialization.

Directory Server handles import cache as it handles database cache. Ensure therefore that sufficient physical memory is available to prevent swapping. Furthermore, benefits of larger import cache tend to diminish for cache sizes larger than 1 GB, so do not allocate more than 1-2 GB for import cache.

Refer to the *Directory Server Administration Reference* for further information about the valid range of `nsslapd-import-cachesize` values.

File System Cache

The operating system allocates available memory not used by Directory Server caches and other applications to the file system cache. This cache holds data recently read from the disk, making it possible for subsequent requests to obtain data copied from cache rather than to read it again from the disk. As memory access is many times faster than disk access, leaving some physical memory available to the file system cache can boost performance.

For 32-bit servers, consider using file system cache as a replacement for some of the database cache. Database cache is more efficient for Directory Server use than file system cache, but file system cache is not directly associated with the Directory Server (`ns-slapd`) process, so you can potentially make a larger total cache available to Directory Server than would be available using database cache alone.

64-bit servers do not have the same process size limit issue. Use database cache instead of file system cache with 64-bit servers.

Refer to the operating system documentation for details on file system cache.

Total Aggregate Cache Size

The sum of all caches used simultaneously must remain smaller than the total size of available physical memory, less the memory intended for file system cache, and for other processes, such as Directory Server itself. For 32-bit servers, this means total aggregate cache size must be limited such that the total Directory Server (`ns-slapd`) process size is less than the maximum process size allowed by the operating system. In practice, this limit is generally in the 2-3 GB range. *Total cache used may well be significantly larger than the size you specify*. Refer to "Database Cache" on page 33 for hints on how to check that the cache size and thus Directory Server process size does not exceed available physical memory.

If suffixes are initialized (bulk loaded) while Directory Server is online, the sum of database, entry, and import cache sizes should remain smaller than the total size of available physical memory.

Table 3-2 Suffix Initialization (Import) Operations and Cache Use

Cache Type	Offline Import	Online Import
Database	no	yes
Entry[1]	yes	yes
Import	yes	yes

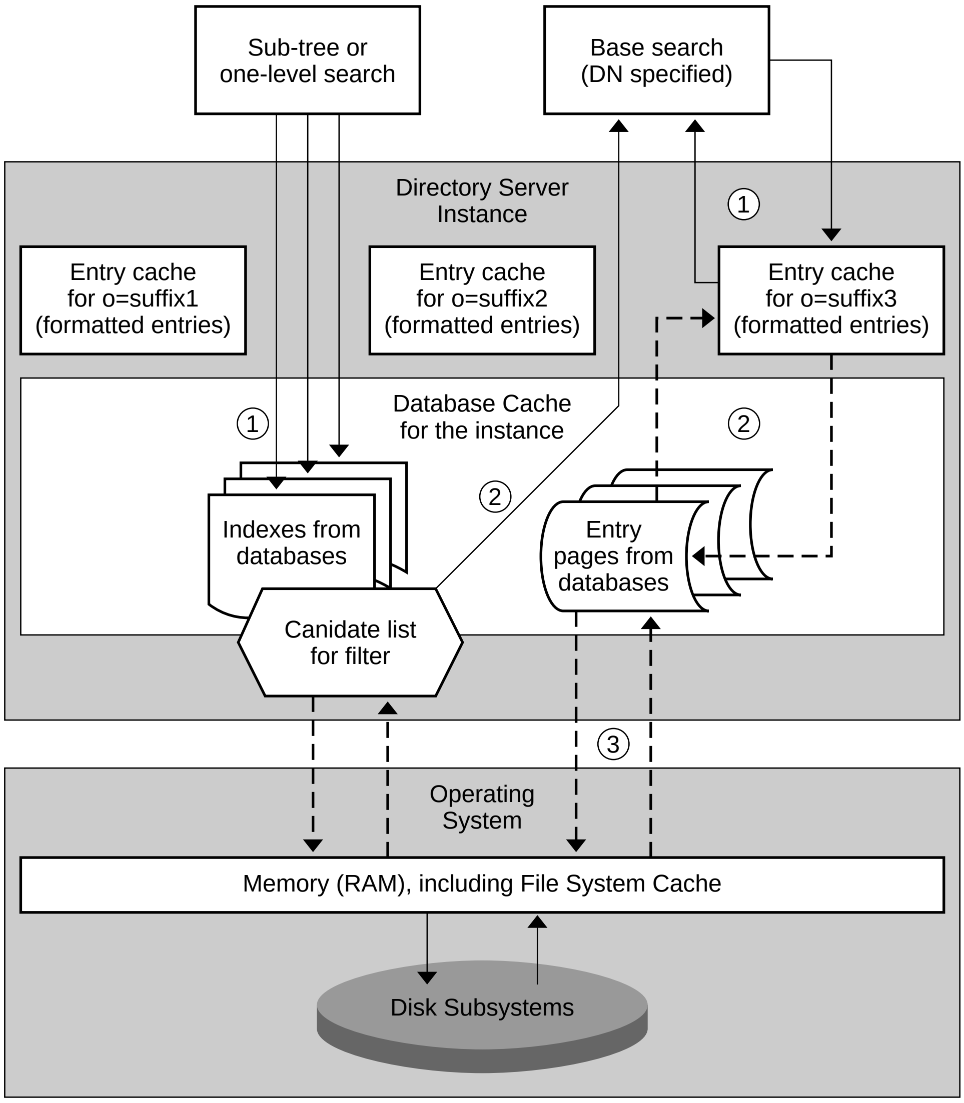
1. As shown in Figure 3-1 on page 33, you have one entry cache for each suffix.

If all suffix initialization takes place offline with Directory Server stopped, you may be able to work around this limitation. In this case import cache does not coexist with database cache, so you may allocate the same memory to import cache for offline suffix initialization and to database cache for online use. If you opt to implement this special case, however, ensure that no one performs online bulk loads on the production system. The sum of the caches used simultaneously must still remain smaller than the total size of available physical memory.

How Searches Use Cache

Figure 3-2 illustrates how Directory Server handles both searches specifying a base DN and searches using filters. Individual lines represent threads accessing different levels of memory, with broken lines representing steps to minimize through effective tuning.

Figure 3-2 Searches and Cache



Sub-tree or one-level search
Base search (DN specified)
Directory Server Instance
1
Entry cache for o=suffix1 (formatted entries)
Entry cache for o=suffix2 (formatted entries)
Entry cache for o=suffix3 (formatted entries)
Database Cache for the instance
1
2
2
Indexes from databases
Entry pages from databases
Canidate list for filter
3
Operating System
Memory (RAM), including File System Cache
Disk Subsystems

Base Search Process

As shown, base searches (those specifying a base DN) are the simplest type of searches for Directory Server to handle. To process such searches, Directory Server:

1. Attempts to retrieve the entry having the specified base DN from the entry cache.

   If the entry is found there, Directory Server checks whether the candidate entry matches the filter provided for the search.

   If the entry matches, Directory Server then quickly returns the formatted, cached entry to the client application.

2. Attempts to retrieve the entry from the database cache.

   If the entry is found there, Directory Server copies the entry to the entry cache for the suffix, and then proceeds as if the entry had been found in the entry cache.

3. Attempts to retrieve the entry from the database itself.

   If the entry is found there, Directory Server copies the entry to the database cache, then proceeds as if the entry had been found in the database cache.

Subtree and One-Level Search Process

Also as shown in Figure 3-2 on page 37, searches on a subtree or a level of a tree involve additional processing to handle sets of entries. To process such searches, Directory Server:

1. Attempts to build a set of candidate entries that match the filter from indexes in the database cache.

   If no appropriate index is present, the set of candidate entries must be generated from the relevant entries in the database itself.

2. Handles each candidate entry by:

   a. Performing a base search to retrieve the entry.

   b. Checking whether the entry matches the filter provided for the search.

   c. Returning the entry to the client application if the entry matches the filter.

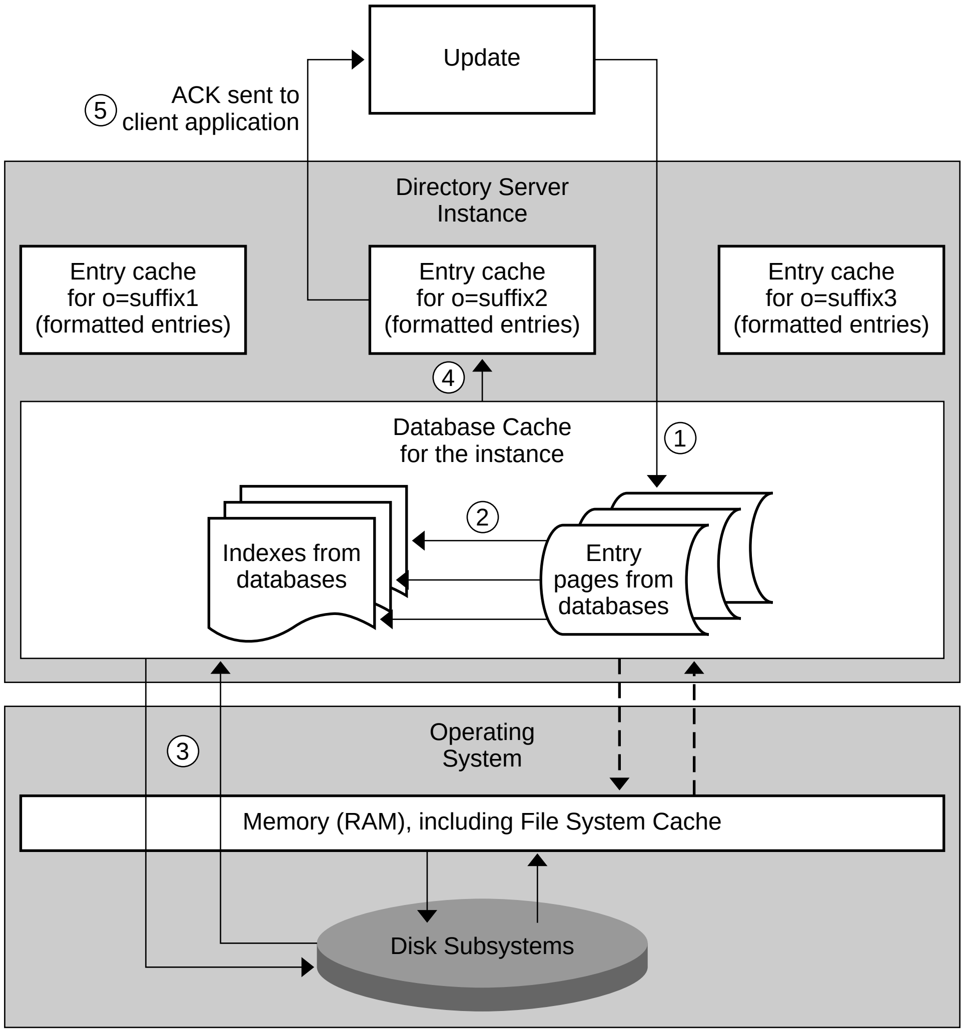
   In this way, Directory Server avoids constructing the set in memory.

Ideally, you know what searches to expect before tuning Directory Server. In practice, verify assumptions through empirical testing.

How Updates Use Cache

Figure 3-3 illustrates how Directory Server handles updates. Individual lines represent threads accessing different levels of memory, with broken lines representing steps to minimize through effective tuning.

Figure 3-3 Updates and Cache


Update
5
ACK sent to client application
Directory Server Instance
Entry cache for o=suffix1 (formatted entries)
Entry cache for o=suffix2 (formatted entries)
Entry cache for o=suffix3 (formatted entries)
4
Database Cache for the instance
1
2
Indexes from databases
Entry pages from databases
3
Operating System
Memory (RAM), including File System Cache
Disk Subsystems

Notice that Figure 3-3 does not show the potential impact on the entry cache of an internal search performed to retrieve the entry for a modify or delete operation. Figure 3-2 on page 37 shows how searches use cache.

Updates involve more processing than searches. To process updates, Directory Server:

1. Performs a base DN search to retrieve the entry to update or verify in the case of an add operation that it does not already exist.

2. Changes the database cache, updating in particular any indexes affected by the update.

   If the data affected by the update has not been loaded into the database cache, this step can result in disk activity while the relevant data are loaded into the cache.

3. Writes information about the changes to the transaction log, waiting for the information to be flushed to disk.

   Refer to "Transaction Logging" on page 69 for details.

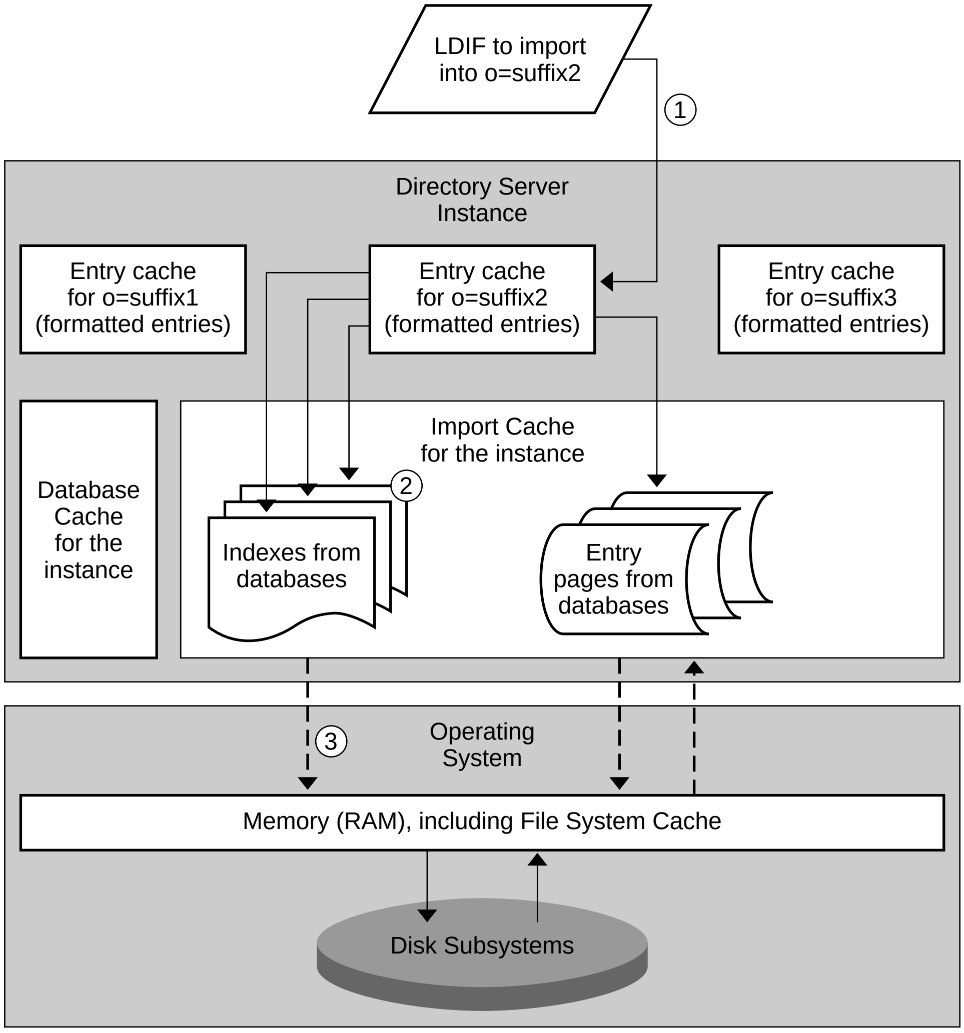
4. Formats and copies the updated entry to the entry cache for the suffix.

5. Returns an acknowledgement of successful update to the client application.

How Suffix Initialization Uses Cache

Figure 3-4 illustrates how Directory Server handles suffix initialization, also known as bulk load import. Individual lines represent threads accessing different levels of memory, with broken lines representing steps to minimize through effective tuning.

Figure 3-4 Suffix Initialization (Bulk Loading) and Cache


LDIF to import into o=suffix2
1
Directory Server Instance
Entry cache for o=suffix1 (formatted entries)
Entry cache for o=suffix2 (formatted entries)
Entry cache for o=suffix3 (formatted entries)
Database Cache for the instance
Import Cache for the instance
2
Indexes from databases
Entry pages from databases
3
Operating System
Memory (RAM), including File System Cache
Disk Subsystems

To initialize a suffix, Directory Server:

1. Starts a thread to feed an entry cache, used as a buffer, from LDIF.
2. Starts a thread for each index affected and a thread to create entries in the import cache. These threads consume entries fed into the entry cache.

3. Reads from and writes to the database files when import cache runs out.

   Directory Server may also write log messages during suffix initialization, but does not write to the transaction log.

Tools for suffix initialization such as `ldif2db` (`directoryserver -u 5.2 ldif2db`) delivered with Directory Server provide feedback concerning cache hit rate and import throughput. Having both cache hit rate and import throughput drop together suggests that import cache may be too small. Consider increasing import cache size.

Optimizing For Searches

For top performance, cache as much directory data as possible in memory. In preventing the directory from reading information from disk, you limit the disk I/O bottleneck. There are a number of different possibilities 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 may choose to allocate more or less memory to entry and database caches to optimize search performance. You may alternatively choose to distribute searches across Directory Server consumers on different servers.

This section covers the following scenarios:

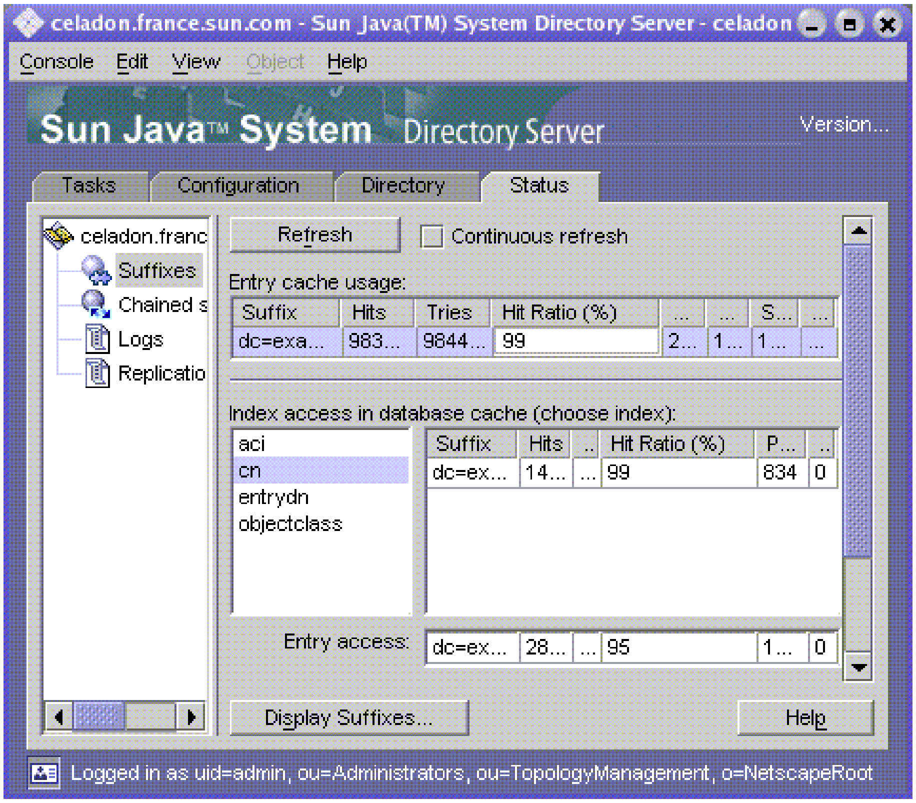
- All Entries and Indexes in Memory
- Plenty of Memory, 32-Bit Directory Server
- Not Enough Memory

All Entries and Indexes in Memory

Imagine the optimum case. Database and entry caches 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.

In this case, 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 activity with Directory Server Console, which displays useful information under the Status tab as shown in Figure 3-5.

Figure 3-5 Monitoring Cache Hit Rate Using Directory Server Console


celadon.france.sun.com - Sun Java(TM) System Directory Server - celadon
Console Edit View Object Help
Sun Java™ System Directory Server
Version...
Tasks Configuration Directory Status
celadon.franc
Suffixes
Chained s
Logs
Replicatio
Refresh
Continuous refresh
Display Suffixes...
Help

Entry cache usage:

Suffix	Hits	Tries	Hit Ratio (%)	...	...	S...	...
dc=exa...	983...	9844...	99	2...	1...	1...	...

Index access in database cache (choose index):

aci
cn
entrydn
objectclass

Suffix	Hits	..	Hit Ratio (%)	P...	..
dc=ex...	14...	...	99	834	0
Entry access: dc=ex...	28...	...	95	1...	0

Logged in as uid=admin, ou=Administrators, ou=TopologyManagement, o=NetscapeRoot

Alternatively, paging and cache activity can be monitored by searching from the command line:

```
$ ldapsearch -D admin -w password \
-b "cn=monitor,cn=database_name,cn=ldbm database,cn=plugins,cn=config"
```

Finding appropriate caches sizes must be done through empirical testing with representative data. Start by allocating a large amount of memory for the caches, and then exercise and monitor Directory Server to observe the result, repeating the process as necessary. Entry caches in particular may use *much more* memory than you allocate to them.

Plenty of Memory, 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 on a Solaris system with 64-bit support, the key is sizing caches appropriately with respect to memory limitations for 32-bit processes, then leaving 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, and size the database cache relatively small such as 100 MB 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. It is thus considerably more difficult to control than other caches, particularly on systems not dedicated to Directory Server.

The system may reallocate file system cache to other processes.

Avoid online import in this situation, as import cache is associated with the Directory Server process.

Not Enough Memory

Imagine a system with insufficient available memory to hold all data in entry and database caches. The key in this case is to avoid causing combined entry and database cache sizes to exceed the available physical memory, resulting in heavy virtual memory paging that could bring the system to a virtual halt.

Start benchmarking by devoting available memory to entry cache and database caches, with sizes no less than 100 MB each. Try disabling the file system cache by mounting Solaris UFS file systems with the `-o forcedirectio` option, described in `mount`(1M). This can prevent the file system cache from using memory needed by Directory Server. Verify and correct assumptions through empirical testing.

Optimizing for Updates

For top update performance, first remove any transaction log bottlenecks observed. Refer to "Transaction Logging" on page 69 for details.

Next, attempt to provide enough memory for the database cache to handle updates in memory and minimize disk activity. You can monitor the effectiveness of the database cache by reading the hit ratio in Directory Server Console. Directory Server Console displays hit ratios for suffixes under the Status tab as shown in Figure 3-5 on page 43.

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 itself be a bottleneck, so storing the database on a separate RAID system such as a Sun StorEdge™ disk array can help improve update performance. You may use utilities such as `iostat`(1M) on Solaris systems to isolate potential I/O bottlenecks.

Table 3-3 shows recommendations for systems with 2, 3, and 4 disks.

Table 3-3 Isolating Databases and Logs on Different Disks

Disks Available	Recommendations
2	• Place the Directory Server database on one disk • Place the transaction log, the access, audit, error logs, and any changelogs on the other disk
3	• Place the Directory Server database on one disk • Place the transaction log on the second disk • Place the access, audit, error logs, and any changelogs on the third disk
4	• Place the Directory Server database on one disk • Place the transaction log on the second disk • Place the access, audit, and error logs on the third disk • Place changelogs on the fourth disk

Cache Priming and Monitoring

Priming caches means filling them with data such that subsequent Directory Server behavior reflects normal operational performance, rather than ramp up. Priming caches is typically useful for arriving at reproducible results when benchmarking, and measuring and analyzing potential optimizations. In most cases, do not actively prime the caches, but instead let the caches be primed by normal or typical. client interaction with Directory Server before you measure performance.

After caches are primed, you may run tests, and monitor whether cache tuning has produced the desired outcomes. Directory Server Console displays monitoring information for caches when you select the Suffixes node under the Status tab as shown in Figure 3-5 on page 43. Alternatively, paging and cache activity can be monitored by searching from the command line:

```
$ ldapsearch -D admin -w password \
-b "cn=monitor,cn=database_name,cn=ldbm database,cn=plugins,cn=config"
```

If database cache size is large enough and the cache is primed, then the hit ratio (`dbcachehitratio`) should be high, and number of pages read in (`dbcachepagein`) and clean pages written out (`dbcacheroevict`) should be low. Here, “high” and “low” must be understood relative to the deployment constraints.

If entry cache for a suffix is large enough and the cache is primed, then the hit ratio (`entrycachehitratio`) should be high. As the entry cache fills, entry cache size (`currententrycachesize`) approaches the maximum entry cache size (`maxentrycachesize`). Ideally, the size in entries (`currententrycachecount`) should be either equal to or very close to the total number of entries in the suffix.

Other Optimizations

Tuning cache sizes represent only one approach to improving search, update or bulk load rates. As you tune the cache, performance bottlenecks from cache move to other parts of the system. Refer to the other chapters in this guide for more information.

Chapter 4

Tuning Indexing

As Directory Server handles more and more entries, searches potentially consume more and more time and system resources. Indexes are one tool to improve search performance. This chapter covers how Directory Server indexes work so that you understand the costs and benefits of using a specific index in the context of a particular deployment. It includes the following sections:

- About Indexes
- Benefits: How Searches Use Indexes
- Costs: How Updates Affect Indexes
- Tuning Indexing for Performance

About Indexes

Indexes associate lookup information with Directory Server entries. Indexes take the form of files stored with Directory Server databases. A *database* in this context is the physical representation of a suffix. For most deployments, one suffix corresponds to one database. For some deployments, one suffix may be split across multiple databases. Directory Server stores databases under *ServerRoot*`/slapd-`*ServerID*`/db/` by default (the default value of `nsslapd-directory`). Here you find individual database instances having one index file per indexed attribute. For instance, a CN index file for a database, `example`, holding entries from the suffix `dc=example,dc=com`, is called *ServerRoot*`/slapd-`*ServerID*`/db/example/example_cn.db3`.

What you index depends upon how client applications access directory data. Table 4-1 includes short descriptions of standard index types.

Table 4-1 Standard Index Types

Index Type	Answers the question...
Approximate	Which entries have a value that sounds like `foobar` for this attribute?
Browsing	Which entries fit this virtual list view search?
Equality	Which entries have value `foobar` for this attribute?
International	Which entries match for this international locale?
Presence	Which entries have this attribute?
Substring	Which entries have a value matching `*foo*` for this attribute?

An index file for a particular attribute such as CN may contain multiple types of indexes. For instance, if CN is indexed in the `example` database for equality and for substring matching, then `example_cn.db3` contains both equality and substring indexes.

Refer to the *Directory Server Administration Guide* for:

- An overview of each index type
- Instructions on creating and deleting indexes
- A list of default indexes created by Directory Server
- A list of system indexes required by Directory Server

Default indexes improve search performance in many situations, and support searches performed by certain other applications. In some cases, you may choose to disable or even delete particular default indexes for performance reasons. System indexes are those on which Directory Server depends. Do not delete or modify them.

Benefits: How Searches Use Indexes

Indexes speed up searches. An index contains a list of values, each associated with a list of entry identifiers corresponding to the value. Directory Server can look up entries quickly using the lists of entry identifiers in indexes. Without an index to manage a list of entries, Directory Server may have to check every entry in a suffix to find matches for a search.

The reason an indexed search may require significantly less processing than an unindexed search becomes evident when search request processing is explained. Here is how Directory Server processes each search request:

1. A client application sends a search request to Directory Server.
2. Directory Server examines the request to ensure the search base corresponds to a suffix it can handle. If not, it returns an error to the client, and may return a referral to another Directory Server instance.
3. Directory Server determines whether it manages an index or indexes appropriate to the search.

   For each such index that exists, Directory Server looks up candidate entries — entries that might be a match for the search request — in the index, as shown in Figure 3-2 on page 37.

   Notice that if no such index exists, Directory Server generates the set of candidate entries from all entries in the database. For large deployments, this step may consume considerable time and system resources, depending on the search.
4. Directory Server examines each candidate entry to determine if it matches the search criteria. Directory Server returns matching entries to the client application as it finds them.

   Directory Server continues examining candidates either until all candidates have been examined, or until it reaches a resource limit such as `nsslapd-lookthroughlimit`, `nsslapd-sizelimit`, or `nsslapd-timelimit`, as described in "Limiting Resources Available to Clients" on page 73.

As is evident from Step 3, indexes can reduce significantly the processing Directory Server must perform to respond to a search request from a client.

Costs: How Updates Affect Indexes

Updates change not only entries themselves, but also indexes referencing the entries. The more references to an entry in indexes, the higher the potential processing cost to modify the indexes during an update. Specifically, Directory Server modifies all impacted indexes as shown in Figure 3-3 on page 39 *before sending acknowledgement of the update to the client application*.

In addition to the processing costs incurred for index maintenance, indexes have a cost in terms of space on disk and potentially space in memory. When optimizing database cache size for searches, as described "Optimizing For Searches" on page 42, you may opt to provide enough memory to hold both entries and indexes in database cache. The larger the indexes, the more space required. 64-bit indexes require somewhat more space than 32-bit indexes, as well.

In general, tuning indexing for an instance of Directory Server means maintaining only those indexes for which the benefits from faster search processing offset the costs of more update processing and of more space needed. Maintaining useful indexes is good practice; maintaining unused indexes for attributes on which clients rarely search is a waste.

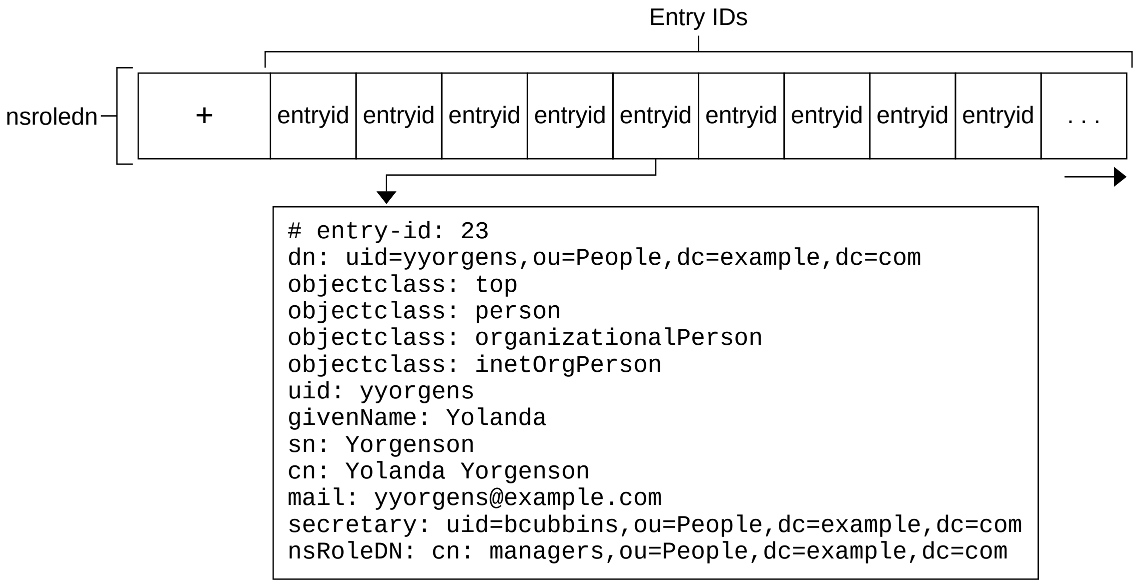
This section explains the costs of using each type of indexing:

- Presence Indexes
- Equality Indexes
- Substring Indexes
- Browsing (Virtual List View) Indexes
- Approximate Indexes
- International Indexes
- Example: Indexing an Entry

Presence Indexes

Figure 4-1 depicts a presence index for the `nsRoleDN` attribute, showing how this index is independent of the attribute value, but simply includes all entries in the database having an `nsRoleDN` attribute. Every value of the attribute matches `+`.

Figure 4-1 Representation of a Presence Index



```
# entry-id: 23
dn: uid=yyorgens,ou=People,dc=example,dc=com
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
uid: yyorgens
givenName: Yolanda
sn: Yorgenson
cn: Yolanda Yorgenson
mail: yyorgens@example.com
secretary: uid=bcubbins,ou=People,dc=example,dc=com
nsRoleDN: cn: managers,ou=People,dc=example,dc=com
```

As shown, the internal `entryid` attribute value allows Directory Server to store a reference to the entry that allows for quick retrieval. Directory Server actually retrieves the entry using the *dbinstance*`/id2entry.db3` index file, where *dbinstance* depends on the database identifier as implied in "About Indexes" on page 47.

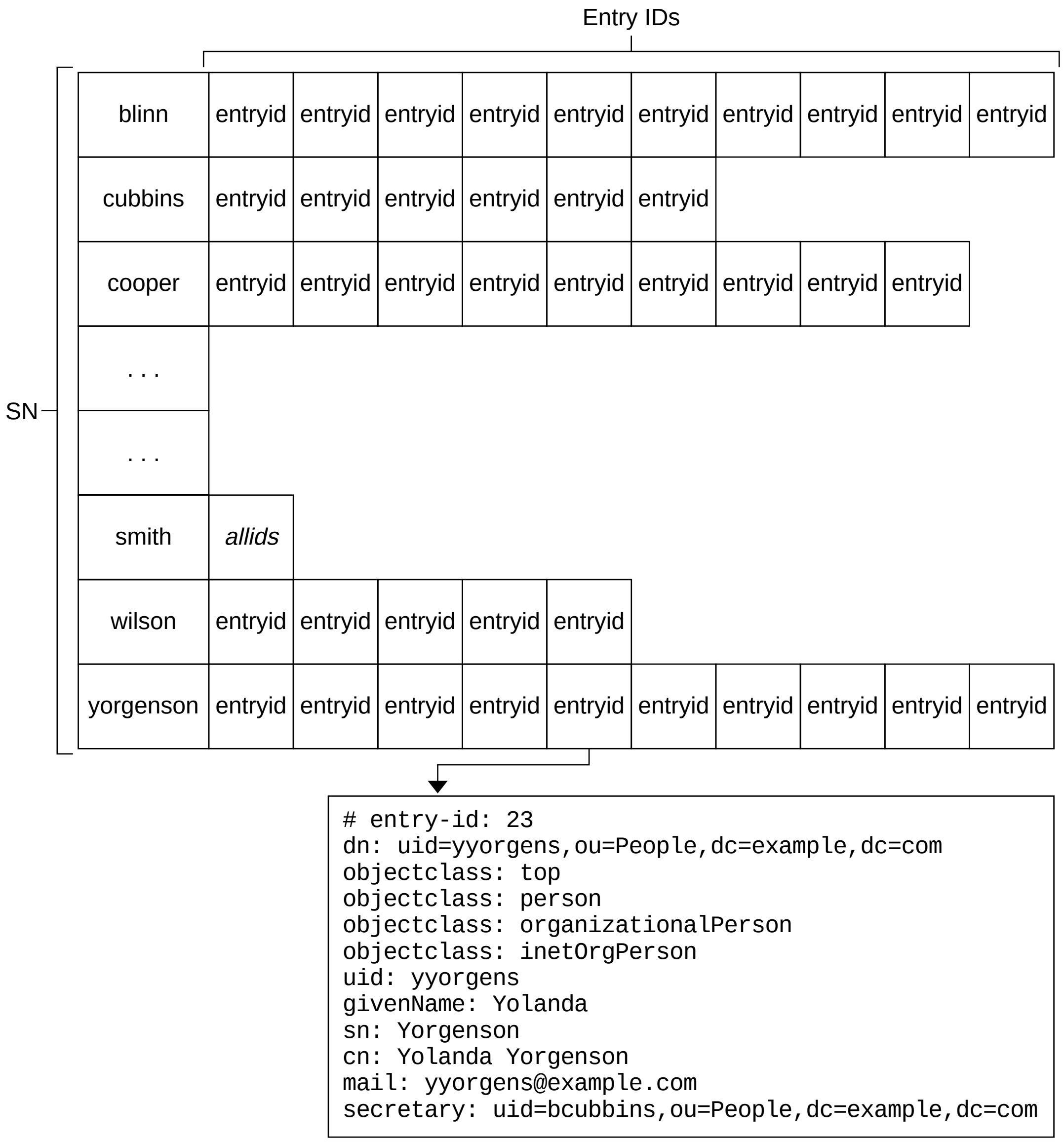
When Directory Server receives an update request to remove an attribute value indexed for presence, it must remove the entry from the presence index for that attribute before returning acknowledgement of the update to the client application.

The cost of presence indexes is generally lower than for other index types, although the list of entries maintained for a presence index may be long. When index list length is limited, presence indexes are useful primarily for attributes present in a relatively small percentage of directory entries. Refer to "Limiting Index List Length" on page 59 for further information.

Equality Indexes

Figure 4-2 depicts an equality index for the `SN` (surname) attribute. It shows how this index maintains a list per attribute value of entries having that attribute value for the `SN` attribute.

Figure 4-2 Representation of an Equality Index



```
# entry-id: 23
dn: uid=yyorgens,ou=People,dc=example,dc=com
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
uid: yyorgens
givenName: Yolanda
sn: Yorgenson
cn: Yolanda Yorgenson
mail: yyorgens@example.com
secretary: uid=bcubbins,ou=People,dc=example,dc=com
```

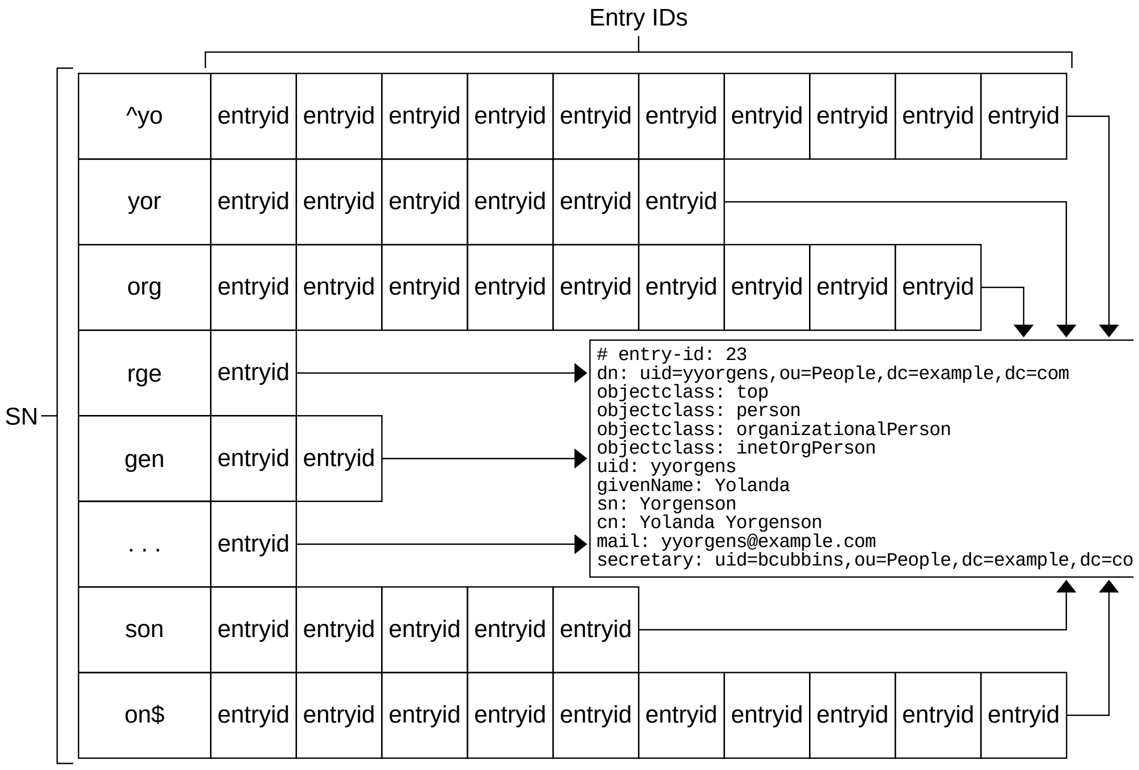
When Directory Server receives an update request for an entry having an attribute indexed for equality, it must determine whether the entry must be removed from the index or not, determine whether a list must be added to or removed from the index, and must then carry out any necessary modifications before returning acknowledgement of the update to the client application.

The cost of equality indexes is generally lower than for substring indexes, for example, but higher in terms of space than for presence. Some client applications such as messaging servers may, however, rely on equality indexes for top search performance. Avoid equality indexes for large binary attributes such as photos and hashed passwords.

Substring Indexes

Figure 4-3 depicts a substring index for the `SN` (surname) attribute. It shows an excerpt of how this index maintains a series of lists per attribute value.

Directory Server indexes substrings in three-character group. The search algorithm includes an optimization such that searches for two-character substrings may use the index. A search for `(sn=*ab*)` may therefore be accelerated using an index, for example, but a search for `(sn=*a*)` cannot. The optimization still is less efficient than using substring searches with at least three-character groups, as the three-character groups are actually stored in the indexes, as shown in Figure 4-3.



```
# entry-id: 23
dn: uid=yyorgens,ou=People,dc=example,dc=com
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
uid: yyorgens
givenName: Yolanda
sn: Yorgenson
cn: Yolanda Yorgenson
mail: yyorgens@example.com
secretary: uid=bcubbins,ou=People,dc=example,dc=co
```

Figure 4-3 Representation of a Substring Index

Furthermore, two-character substring search are more subject to reaching the index list length limit, after which the search no longer uses indexes. Refer to "Limiting Index List Length" on page 59 for further information.

Directory Server offers a further optimization allowing initial substring searches of only one character before the wildcard. Thus a search for `(sn=a*)`, but not `(sn=*a*)` or `(sn=*a)`, can also be accelerated when a substring index is available, for example. This optimization is subject to the same limitations as the two-character substring searches.

Notice that Directory Server builds an index of substrings according to its own built-in rules. These substrings are not configurable by the system administrator.

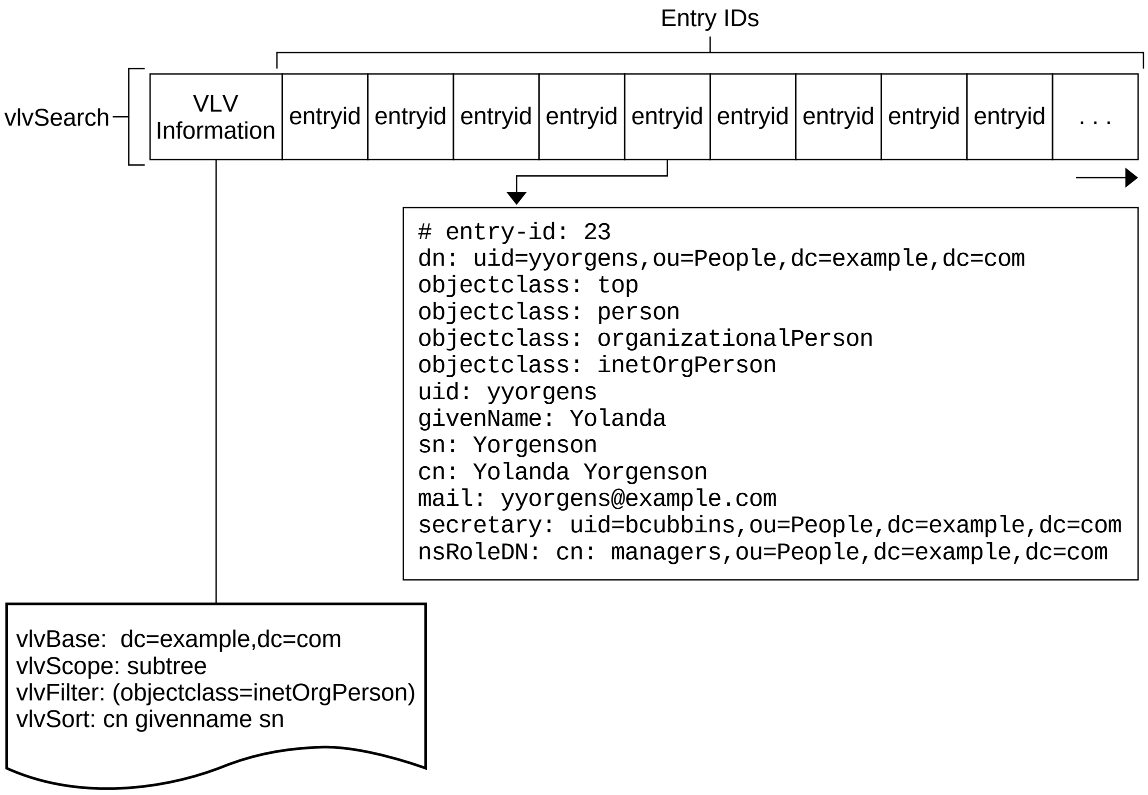
When Directory Server receives an update request for an entry having an attribute indexed for substrings, it must determine whether the entry must be removed from the index, determine whether and how modifications to the entry affect the index, determine whether entry IDs or lists of entry IDs must be added to or removed from the index, and must then carry out any necessary modifications before returning acknowledgement of the update to the client application. The number of updates depends on the length of the attribute value string.

Maintaining substring indexes is generally quite costly. As the cost is a function of the length of the string indexed, avoid unnecessary substring indexes, especially for attributes having potentially long string values such as `description`. Substring indexes cannot be applied to binary attributes such as photos.

Browsing (Virtual List View) Indexes

Figure 4-4 depicts a browsing index for a virtual lists view. It shows how this index depends on the virtual list view information. That is, the `vlvBase`, `vlvScope`, `vlvFilter`, and `vlvSort` attribute values for the browsing index. Entry IDs in this type of index are ordered according to the `vlvSort` criteria.

Figure 4-4 Representation of a Browsing Index



```
# entry-id: 23
dn: uid=yyorgens,ou=People,dc=example,dc=com
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
uid: yyorgens
givenName: Yolanda
sn: Yorgenson
cn: Yolanda Yorgenson
mail: yyorgens@example.com
secretary: uid=bcubbins,ou=People,dc=example,dc=com
nsRoleDN: cn: managers,ou=People,dc=example,dc=com
```

```
vlvBase:  dc=example,dc=com
vlvScope: subtree
vlvFilter: (objectclass=inetOrgPerson)
vlvSort: cn givenname sn
```

When Directory Server receives an update request for an entry matching a `vlvFilter` value, it must determine whether the entry must be removed from the index or not, determine the correct position of the entry in the list, and must then carry out any necessary modifications before returning acknowledgement of the update to the client application.

Approximate Indexes

Directory Server maintains approximate indexes using a variation of the *metaphone* phonetic algorithm. This algorithm breaks down an attribute string value into a rough approximation of its English phonetic pronunciation. Values to match in incoming search requests are handled using the same algorithm. As the algorithm is based loosely on syllables, it is not effective for attributes containing numbers such as telephone numbers.

The algorithm generates a target string for each attribute value string. Costs for this "sounds like" indexing of English-language strings are therefore similar to those for equality indexing.

International Indexes

International indexes use matching rules for particular locales to maintain indexes. Costs for such indexes therefore resemble costs for substring and equality indexes.

Using a custom matching rule server plug-in, you can extend standard support for international and other types of indexing. Refer to the *Directory Server Plug-in Developer's Guide* for more information on custom matching rule plug-ins.

Example: Indexing an Entry

Consider a user entry as shown in Code Example 4-1 being added to a suffix indexed for equality on `uid`, for equality, substring and approximate searches on Common Name (`cn`) and surname (`sn`) attributes, for equality searches on the `mail` attribute, for equality and substring searches on the `telephoneNumber` attribute, and for substring searches on the `description` attribute. This section examines why you might not want, for example, to create substring attributes on long string values, such as that in the `description` attribute.

Code Example 4-1 Sample User Entry

```
dn: uid=yyorgens,ou=People,dc=example,dc=com
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
uid: yyorgens
givenName: Yolanda
sn: Yorgenson
cn: Yolanda Yorgenson
mail: yolanda.yorgenson@example.com
telephoneNumber: 1-650-960-1300
description: Business Development Manager, Platinum Partners
```

In adding this entry, Directory Server must modify indexes for `cn`, `sn`, `mail`, `telephoneNumber`, and `description`. Table 4-2 illustrates the expected number of entries.

Table 4-2 Index Updates for Sample User Entry

Attribute	Approximate	Equality	Substring[1]	Total Index Updates
`uid`		1		1
`cn`	1	1	17	19
`sn`	1	1	9	11
`mail`		1		1
`telephoneNumber`		1	11	12
`description`			47	47

1. Substring indexing on strings as long as the `description` string here is not recommended for most deployments.

Notice that the number of substring index updates for the `description` string is larger (47) than the number of updates (44) for all other attributes combined. Also, further modifications to the `description` string may again imply a maximum number of updates or more depending on the new string. In most cases, avoid substring indexing of this volume by not applying substring indexing to long strings such as `description` values.

Tuning Indexing for Performance

In many cases, tuning indexing for performance implies activating indexes to speed up frequent searches, and deactivating indexes that are expensive to maintain and not frequently used.

NOTE Database backups include indexes, and so should match the Directory Server configuration.

After changing how indexes are configured, back up both the configuration and the data.

Allowing Only Indexed Searches

Directory Server makes it possible to prevent costly unindexed searches, returning `LDAP_UNWILLING_TO_PERFORM` to clients requesting an unindexed search.

To prevent unindexed searches against a particular database, set the `nsslapd-require-index` attribute value to `on` for the database:

```
$ ldapmodify -h host -p port -D "cn=Directory Manager" -w password
dn: cn=example, cn=ldbm database, cn=plugins, cn=config
changetype: modify
replace: nsslapd-require-index
nsslapd-require-index: on
^D (^Z on Windows systems)
```

The change takes effect immediately. No need to restart Directory Server.

Limiting Index List Length

In large and fast growing directory deployments, indexing may reach the point of diminishing returns for a particular index key. At the point of diminishing returns, the list associated with a particular key becomes so long that maintaining the list costs more than performing an occasional unindexed search on that particular key for candidate entries.

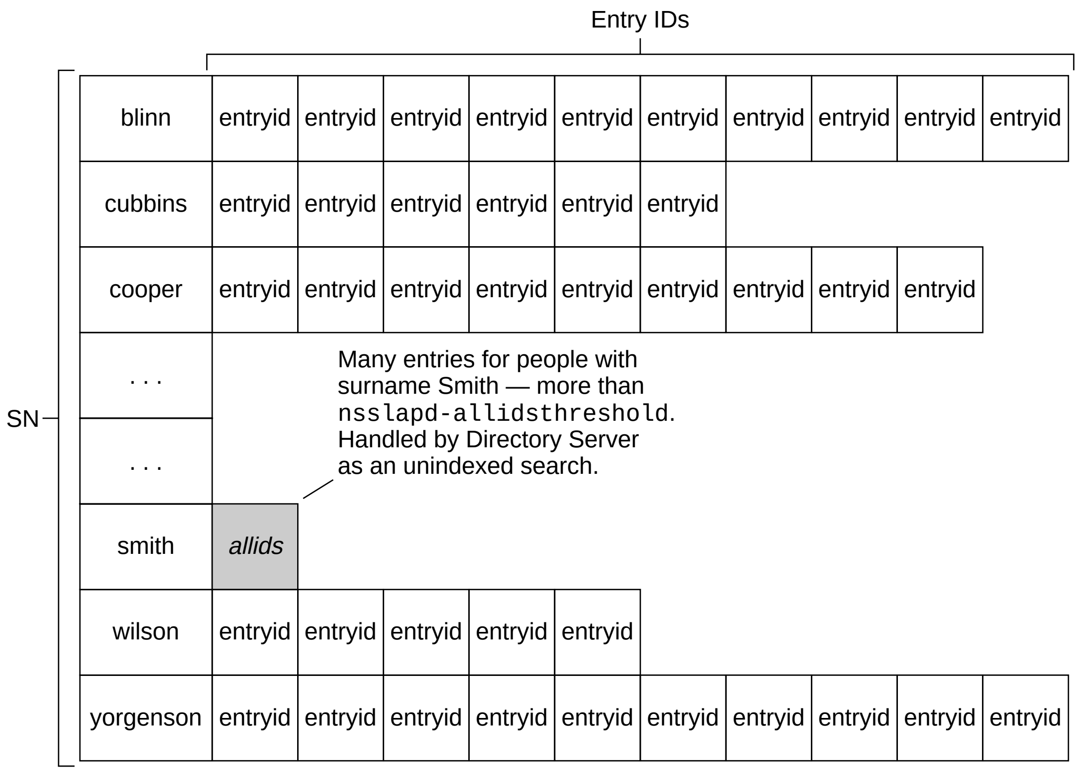
Imagine a library card catalog proposal for indexing by topic. Imagine one of the topics is fiction. Yet, the library has so many works of fiction that looking them up in the card catalog and then going to the shelves to retrieve the books takes longer than simply browsing through the fiction section. So the library does not maintain a catalog for fiction, but still maintains card catalogs for other topics.

Directory Server has a mechanism for handling this situation, using a configuration attribute holding a threshold value. If the number of entries in the list for a particular key reaches the threshold, Directory Server replaces the list for the key with a token specifying that an unindexed search should be performed to find candidate entries for that particular key. The value is somewhere near but less than the value for the maximum number of candidate entries checked for a search, set using `nsslapd-lookthroughlimit`, as described in Table 6-2 on page 76.

The mechanism is referred to as the *all IDs threshold*, named after the configuration attribute used to set the global threshold value, `nsslapd-allidsthreshold` on `cn=config,cn=ldbm database,cn=plugins,cn=config`. Notice this value is currently global to the Directory Server instance. It cannot be set differently for different indexes.

Figure 4-5 illustrates the example of indexing on surname with a number of Smiths greater than `nsslapd-allidsthreshold`.

Figure 4-5 Reaching the All IDs Threshold for an Index Key


Entry IDs
SN
blinn
cubbins
cooper
. . .
. . .
smith
wilson
yorgenson
entryid
allids
Many entries for people with surname Smith — more than nsslapd-allidsthreshold. Handled by Directory Server as an unindexed search.

Notice that the threshold affects only one list in the index table. Lists for other keys are not affected.

Symptoms of Inappropriate Index List Size

If clients perform primarily indexed searches and cache sizes are correctly tuned as described in Chapter 3, "Tuning Cache Sizes," yet you still observe poor search performance, an inappropriate threshold value may be the cause. When you observe poor search performance for indexed searches, ensure cache sizes are appropriately tuned first. Next, examine the access log to determine whether Directory Server is reaching the all IDs threshold often.

The `notes=U` flag at the end of an `access` log `RESULT` message indicates Directory Server performed an unindexed search. A previous `SRCH` message for the same connection and operation specifies the search filter used. The following two-line example traces an unindexed search for `(cn=Smith)` returning 10000 entries. Time stamps have been removed from the messages.

```
conn=2 op=1 SRCH base="o=example.com" scope=0 filter="(cn=Smith)"
conn=2 op=1 RESULT err=0 tag=101 nentries=10000 notes=U
```

If you observe many such pairs for searches that should be indexed, you may be able to improve search performance by increasing the threshold.

Changing the Index List Threshold Size

Good values for `nsslapd-allidsthreshold` typically fall in a range around 5 percent of the total number of entries in the directory. For example, the default value of 4000 is generally right for Directory Server instances handling 80,000 entries or less. You may decide to set the value significantly higher than 5 percent of the total if you expect to add large numbers of entries to the directory in the near term, or if you expect the directory to grow considerably. You may also decide to set the threshold differently on consumer replicas supporting many searches than on masters supporting almost only writes. If you plan to initialize a large directory from LDIF in the near term, you may even choose to adjust the value for `nsslapd-allidsthreshold` just before initialization, as each change to the value of this attribute requires that all indexes be rebuilt. Finally, you may choose to set this value quite high in directories with deeply hierarchical DITs, so searches for all entries below a given branch are indexed. In any case, avoid setting the all IDs threshold very high (above 50,000) even for very large deployments unless you have a good, specific reason for doing so.

Change the all IDs threshold as follows. Note that service is interrupted on the Directory Server instance undergoing the change.

1. Adjust the value of the `nsslapd-allidsthreshold` attribute on `cn=config, cn=ldbm database, cn=plugins, cn=config` using `ldapmodify`.
2. Stop the Directory Server instance.
3. Export all directory databases to LDIF.
4. Initialize all directory databases from LDIF.

Refer to the *Directory Server Administration Guide* for specific instructions.

5. If database cache size was tuned for the old all IDs threshold value and the server has adequate physical memory, consider increasing database cache size by 25 percent of the magnitude of the increase to the threshold.

   In other words, if you increase the all IDs threshold from 4000 to 6000, you may choose to increase database cache size by about 12.5 percent to account for increased index list size. Find the optimum size empirically before applying changes to production servers. Refer to Chapter 3, "Tuning Cache Sizes," for details on database cache tuning.

6. Restart the Directory Server instance.

Chapter 5

Tuning Logging

Directory Server provides several log types, summarized in Table 5-1. This chapter discusses how to handle the different types of logs.

Table 5-1 Types of Logs Used by Directory Server

Log	**Type**	**Use**
Access	Flat file	Evaluating directory use patterns, verifying configuration settings, diagnosing access problems. Refer to "Access Logging" on page 64 for details.
Audit	Flat file	Providing audit trails for security and data integrity. Refer to "Audit Logging" on page 65 for details.
Changelog	Database	Enables synchronization between replicas. Refer to "Multi-Master Replication Change Logging" on page 68 for details.
Error	Flat file	Debugging directory deployments. Refer to "Error Logging" on page 66 for details.
Retro changelog	Database	Permitting backward compatibility with previous versions. Refer to "Retro Change Logging" on page 69 for details.
Transaction	Database	Maintaining database integrity. Refer to "Transaction Logging" on page 69 for details.

In high-volume deployments, writing to logs can be disk intensive, resulting in noticeable negative performance impact. Given the potential for I/O bottlenecks inherent with heavy logging in high volume systems, consider putting log files on a lesser used disk.

Access Logging

The access log contains detailed information about client connections and operations performed. The access log can be indispensable when diagnosing access problems, verifying server configuration settings, and evaluating server usage patterns.

Although the access log provides beneficial troubleshooting information, it may become an I/O bottleneck. Set access logging levels to the minimum required level. Table 5-2 provides further recommendations for specific attributes.

Table 5-2 Tuning Recommendations for Access Logging

Configuration Attribute	**Short Description and Tuning Recommendations**

`dn: cn=config`
`nsslapd-accesslog`
	Specifies the path and filename of the access log file.
	In most deployments, the access log may share a disk with the audit and error logs, and the replication changelog.

`dn: cn=config`
`nsslapd-accesslog-level`
	Specifies the level of informational logging used.
	Leave at default (256) unless a higher level is required.

`dn: cn=config`
`nsslapd-accesslog-logbuffering`
	Determines whether the access log is buffered.
	Leave `on` (default) unless you must disable buffering to see access log messages as they are triggered. Disabling buffering can result in a drop in overall performance.

`dn: cn=config`
`nsslapd-accesslog-logging-enabled`
	Enables and disables access logging.
	Set `nsslapd-accesslog-level` to the lowest acceptable setting. Rotate the access log frequently (each day or week) and use `nsslapd-accesslog-logmaxdiskspace` and `nsslapd-accesslog-logminfreediskspace` to manage disk space use.

`dn: cn=config`
`nsslapd-accesslog-logmaxdiskspace`
	Specifies maximum disk space in MB that all access logs (current and rotated logs) may consume.
	Set this value below the total amount of disk space dedicated to access logging, leaving space for other logs on the disk.

Table 5-2 Tuning Recommendations for Access Logging *(Continued)*

Configuration Attribute	Short Description and Tuning Recommendations
`dn: cn=config` `nsslapd-accesslog-logminfreediskspace`	Specifies minimum free disk space in MB allowed before old logs are purged. When the amount of free disk space falls below the value specified on this attribute, the oldest access logs are deleted until enough disk space is freed to correspond to the setting for this attribute. If the access logs cannot be written because the disk is full, the server shuts down.

Refer to the *Directory Server Administration Reference* for information about individual configuration attributes.

The *Directory Server Resource Kit Tools Reference* covers extracting information from the access log.

Audit Logging

The audit log contains detailed information about all changes made to each database as well as to server configuration. Audit logging is disabled by default.

When enabled in deployments having high modify volume, enabling audit logging causes a very noticeable overall drop in performance. Unless the deployment requires it, leave audit logging disabled. For large or high volume deployments that require audit logging, consider allocating a separate disk on a separate controller to the audit log. Table 5-3 provides further recommendations for specific attributes.

Table 5-3 Tuning Recommendations for Audit Logging

Configuration Attribute	Short Description and Tuning Recommendations
`dn: cn=config` `nsslapd-auditlog`	Specifies the path and filename of the audit log file. In most deployments, the audit log may share a disk with the access and error logs, and the replication changelog.
`dn: cn=config` `nsslapd-auditlog-logging-enabled`	Enables and disables audit logging. Leave `off` (default setting) unless audit logging is required.

Table 5-3 Tuning Recommendations for Audit Logging *(Continued)*

Configuration Attribute	Short Description and Tuning Recommendations
`dn: cn=config` `nsslapd-auditlog-logmaxdiskspace`	Specifies maximum disk space in MB that all audit logs (current and rotated logs) may consume. Set this value below the total amount of disk space dedicated to audit logging, leaving space for other logs on the disk.
`dn: cn=config` `nsslapd-auditlog-logminfreediskspace`	Specifies minimum free disk space in MB allowed before old logs are purged. When the amount of free disk space falls below the value specified on this attribute, the oldest audit logs are deleted until enough disk space is freed to correspond to the setting for this attribute. If the audit logs cannot be written because the disk is full, the server shuts down.

Error Logging

The error log for a Directory Server instance contains detailed error, warning, and informational messages encountered during normal server operation. The low default logging level produces relatively little disk activity.

When log level is set higher to generate debugging information, however, Directory Server may begin writing large numbers of messages to disk. The write load can result in a very noticeable overall drop in performance. To avoid a drop in performance, increase log levels progressively, component by component, instead of activating log levels for all components at once.

The error log does not support log buffering. All messages are flushed to disk immediately. Table 5-4 provides recommendations for specific attributes.

Table 5-4 Tuning Recommendations for Error Logging

Configuration Attribute	Short Description and Tuning Recommendations
`dn: cn=config` `nsslapd-errorlog`	Specifies the path and filename of the error log file. In most deployments, the error log may share a disk with the access and audit logs, and the replication changelog.
`dn: cn=config` `nsslapd-errorlog-logging-enabled`	Enables and disables error logging. Leave `on` (default setting).

Table 5-4 Tuning Recommendations for Error Logging *(Continued)*

Configuration Attribute	Short Description and Tuning Recommendations

`dn: cn=config`
`nsslapd-errorlog-logmaxdiskspace`
	Specifies maximum disk space in MB that all error logs (current and rotated logs) may consume.
	Set this value below the total amount of disk space dedicated to error logging, leaving space for other logs on the disk.

`dn: cn=config`
`nsslapd-errorlog-logminfreediskspace`
	Specifies minimum free disk space in MB allowed before old logs are purged.
	When the amount of free disk space falls below the value specified on this attribute, the oldest error logs are deleted until enough disk space is freed to correspond to the setting for this attribute. If the error logs cannot be written because the disk is full, the server shuts down.

`dn: cn=config`
`nsslapd-infolog-area`
	Specifies the components for which informational messages are logged.
	Leave at `0` (default) unless debugging a component. Avoid setting for more than one component at a time on production servers.

`dn: cn=config`
`nsslapd-infolog-level`
	Specifies the level of informational logging used.
	Leave at `0` (default) unless debugging a component for which setting `nsslapd-infolog-area` alone fails to generate sufficient detail.

Refer to the *Directory Server Administration Reference* for information about individual configuration attributes.

Multi-Master Replication Change Logging

Directory Server uses a replication changelog to enable synchronization between replicas. Refer to the *Directory Server Deployment Planning Guide* for an discussion of the changelog and to the *Directory Server Administration Reference* for configuration details. Table 5-5 provides further recommendations for specific attributes.

Table 5-5 Tuning Recommendations for Multi-Master Change Logging

Configuration Attribute	Short Description and Tuning Recommendations

`dn: cn=changelog5,cn=config`
`nsslapd-cachememsize`
	Specifies the changelog database cache size.
	Consider changing this from the default of 10 MB for high volume deployments.

`dn: cn=changelog5,cn=config`
`nsslapd-changelogdir`
	Specifies the path of the changelog database.
	In most deployments, the replication changelog may share a disk with the access, audit, and error logs.

`dn: cn=changelog5,cn=config`
`nsslapd-changelogmaxage`
	Specifies the maximum age for entries in the changelog. Refer to the *Directory Server Administration Reference* for details on the syntax.
	Change this from `0` (default, indicating no maximum) to an interval after which replicated servers are fully synchronized and the changelog may be trimmed.

`dn: cn=changelog5,cn=config`
`nsslapd-changelogmaxentries`
	Specifies the maximum number of entries in the changelog.
	Change this from `0` (default, indicating no maximum) to a number sufficient to allow replicated servers to become fully synchronized before the changelog is trimmed.

Refer to the *Directory Server Administration Reference* for information about individual configuration attributes.

Retro Change Logging

Directory Server ships with a retro changelog plug-in that you may enable to record changes on a supplier server in a format compatible with Directory Server 4.x releases and accessible through LDAP. The retro changelog plug-in is disabled by default and should not be enabled unless required for compatibility reasons. Refer to the *Directory Server Administration Reference* for details. Table 5-6 provides further recommendations for specific attributes.

Table 5-6 Tuning Recommendations for Retro Change Logging

Configuration Attribute	Short Description and Tuning Recommendations

`dn: cn=Retro Changelog Plugin,cn=plugins,cn=config`
`nsslapd-changelogdir`
	Specifies the path of the retro changelog.
	In most deployments, the retro changelog may share a disk with the access, audit, and error logs.

`dn: cn=Retro Changelog Plugin,cn=plugins,cn=config`
`nsslapd-changelogmaxage`
	Specifies the maximum age for entries in the retro changelog. Refer to the *Directory Server Administration Reference* for details on the syntax.
	Change this from `0` (default, indicating no maximum) to an interval after which clients using the retro changelog have processed the log entries generated.

`dn: cn=Retro Changelog Plugin,cn=plugins,cn=config`
`nsslapd-changelogmaxentries`
	Specifies the maximum number of entries in the retro changelog.
	Change this from `0` (default, indicating no maximum) to a maximum number of entries retained in the retro changelog before trimming.

Refer to the *Directory Server Administration Reference* for information about individual configuration attributes.

Transaction Logging

Directory Server maintains database integrity through transaction logging. Upon accepting an update operation — `add`, `modify`, `delete`, or `modrdn` — Directory Server writes a log message about the operation to the transaction log. Durable transaction logging, enabled by default, ensures data integrity. It does so by ensuring each update operation is committed to the transaction log on disk before the result code for the update operation is returned to the client application. In the event of a system crash, Directory Server uses the transaction log to recover the database. As the transaction log aids in the recovery of a database shut down abnormally, consider storing the transaction log and directory database on separate disk subsystems.

Table 5-7 provides recommendations for specific attributes.

Table 5-7 Tuning Recommendations for Transaction Logging

Configuration Entry DN and Configuration Attribute	Short Description and Tuning Recommendations

`dn: cn=config,cn=ldbm database,cn=plugins,cn=config`
 	
`nsslapd-db-checkpoint-interval`
	Specifies how often Directory Server checkpoints the transaction log, ensures the entire database system is synchronized to disk, and cleans up transaction logs.
	
	Leave at `60` (default interval in seconds) unless database performance optimization based on empirical testing calls for a different value. Increasing the value of this attribute may result in a performance boost for update operations, but also means that recovery after disorderly shutdown takes longer, and that the transaction log uses more disk space.

`dn: cn=config,cn=ldbm database,cn=plugins,cn=config`
 	
`nsslapd-db-durable-transaction`
	Specifies whether update operations are committed to the transaction log on disk before result codes are sent to clients.
	
	Leave `on` (default) for deployments requiring a high level of data integrity. Rather than disabling durable transaction logging to boost performance, first consider batching transactions using `nsslapd-db-transaction-batch-val`.
	
	When durability is disabled, log messages flushed to the file system but not yet to disk may be lost in the event of a system crash. This means that with durable transaction logging `off`, some updates may be unrecoverable even after the client receives a successful update result code.

Table 5-7 Tuning Recommendations for Transaction Logging *(Continued)*

Configuration Entry DN and Configuration Attribute	**Short Description and Tuning Recommendations**

`dn: cn=config,cn=ldbm database,cn=plugins,cn=config`
`nsslapd-db-logbuf-size`
	Specifies the buffer size for log information stored in memory until the buffer fills or the transaction commit forces the buffer to be written to disk.
	Leave at `524288` (512K, default). If you must change the value, do so before loading much data into the directory, then follow these steps:
	1. Reduce the load on Directory Server.
	2. Export all databases to LDIF.
	3. Change the value of `nsslapd-db-logbuf-size`.
	4. Stop Directory Server.
	5. Delete files with names of the form `__db.`*xxx* and `guardian` in `nsslapd-db-home-directory`.
	6. Import all databases from LDIF.
	7. Start Directory Server.
	The value of this attribute must not exceed 25% of the transaction log file size, which by default is 10 MB. For a default configuration, therefore, this attribute should not exceed 2.5 MB in size.

`dn: cn=config,cn=ldbm database,cn=plugins,cn=config`
`nsslapd-db-logdirectory`
	Specifies the path of the transaction log.
	Consider storing the transaction log and directory database on separate disk subsystems.

`dn: cn=config,cn=ldbm database,cn=plugins,cn=config`
`nsslapd-db-transaction-batch-val`
	Specifies how many updates are batched before being committed to the directory database.
	Only change from `0` (no batching, default) if you can afford to lose updates in the event of a crash.
	If you can afford to lose updates in a crash, then setting this to attribute to a value such as 5 can potentially increase write performance significantly. In order for batching to work correctly, the maximum size of a batch of transactions must fit in the transaction log buffer. You may therefore need to increase the value of `nsslapd-db-logbuf-size` when changing the value of this attribute.

Refer to the *Directory Server Administration Reference* for information about individual configuration attributes.

Chapter 6

Managing Use of Other Resources

After optimizing cache size, attribute value indexing, and log management, it may prove useful to tune how Directory Server limits resources made available to client applications, and how Directory Server makes use of system resources. It may also prove useful to reconfigure and even disable some features offered as Directory Server plug-ins. This chapter includes the following sections:

- Limiting Resources Available to Clients
- Using Available System Resources

Limiting Resources Available to Clients

Default configuration may allow client applications to use more Directory Server resources than are actually required. This may leave the door open to accidentally or intentionally abusive client applications negatively impacting server performance, by opening many connections then leaving them idle or unused, launching costly and unnecessary unindexed searches, or storing enormous and unplanned for binary attribute values in the directory.

In some deployment situations, it is not advisable to modify the default configuration. For deployments in which you opt not to change the configuration attribute values mentioned in this section, consider using Sun Java System Directory Proxy Server software to set limits externally, and to help protect against denial of service attacks.

In some deployment situations, one instance of Directory Server must support both directory-intensive client applications such as messaging servers and occasional 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 recommendations in Table 6-1 address settings for limiting resources available to all client applications. These limits do not apply to the Directory Manager user, so ensure client applications do not connect as the Directory Manager user.

Table 6-1 Tuning Recommendations for Limiting Resources Available to Clients

Configuration Entry DN and Attribute	Short Description and Tuning Recommendations

`dn: cn=config`
 	
`nsslapd-idletimeout`
	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.
	
	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 limits.
	
	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 (2 hours), for example.

dn: `cn=config` `nsslapd-ioblocktimeout`	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. For Directory Server instances particularly exposed to denial of service attacks, consider lowering this value from the default of 1,800,000 milliseconds (30 minutes).
dn: `cn=config,cn=ldbm database,cn=plugins,cn=config` `nsslapd-lookthroughlimit`	Sets the maximum number of candidate entries checked for matches during a search. 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 limits. In any case, consider lowering this value from the default of 5000 entries, but not below the threshold value of `nsslapd-sizelimit`.
dn: `cn=config` `nsslapd-maxbersize`	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. If you are confident you can accurately anticipate maximum entry size for your directory data, consider changing this value from the default of 2097152 (2 MB) to the size of the largest expected directory entry. 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.

Table 6-1 Tuning Recommendations for Limiting Resources Available to Clients *(Continued)*

Configuration Entry DN and Attribute	Short Description and Tuning Recommendations
dn: `cn=config` `nsslapd-maxthreadsperconn`	Sets the maximum number of threads per client connection. 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 limits. 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. It is typically not advisable to specify more than 10 threads per connection.
dn: `cn=config` `nsslapd-sizelimit`	Sets the maximum number of entries Directory Server returns in response to a search request. 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 limits. In any case, consider lowering this value from the default of 2000 entries.
dn: `cn=config` `nsslapd-timelimit`	Sets the maximum number of seconds Directory Server allows for handling a search request. 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 limits. In any case, set this value as low as you can and still meet deployment requirements. The default value of 3600 seconds (1 hour) is larger than necessary for many deployments. Consider using 600 seconds (10 minutes) as a starting point for optimization tests.

Refer to the *Directory Server Administration Reference* for information about individual configuration attributes.

Using Available System Resources

Depending on deployment requirements, you may choose to tune how a Directory Server instance uses system and network resources, how access control is managed, and how server plug-ins are configured. The recommendations in Table 6-2 address settings for system resources.

Table 6-2 Tuning Recommendations for Configuring Use of System Resources

Attribute (on dn: cn=config)	Short Description and Tuning Recommendations
`nsslapd-listenhost`	Sets the hostname for the IP interface on which Directory Server listens. This attribute is multi-valued. 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. 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.
`nsslapd-maxdescriptors`	Sets the maximum number of file descriptors Directory Server attempts to use. 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. 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. If you have increased the number of file descriptors available on the system as described in "File Descriptors" on page 28, then set the value of this attribute accordingly. The value of this attribute should be less than or equal to the maximum number of file descriptors available on the system.
`nsslapd-nagle`	Sets whether to delay sending of TCP packets at the socket-level. Consider setting this to `on` if you need to reduce network traffic.

Table 6-2 Tuning Recommendations for Configuring Use of System Resources *(Continued)*

Attribute (on dn: cn=config)	Short Description and Tuning Recommendations
`nsslapd-reservedescriptors`	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*. Consider increasing the value of this attribute from the default of 64 if all of the following are true. • Directory Server replicates to more than 10 consumers or Directory Server maintains more than 30 index files. • Directory Server handles a large number of client connections. • Messages in the error log suggest Directory Server is running out of file descriptors for operations *not* related to client connections. 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 `nsslapd-maxdescriptors`. 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` to: `20 + 4 * (number of databases) + (total number of indexes) + (value of nsoperationconnectionslimit) * (number of chaining backends) + ReplDescriptors + PTADescriptors + SSLDescriptors` Where *ReplDescriptors* = number of supplier replica + 8 if replication is used, *PTADescriptors* is 3 if the Pass Through Authentication (PTA) plug-in is enabled (0 otherwise), and *SSLDescriptors* is 5 if SSL is used (0 otherwise). 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.
`nsslapd-securelistenhost`	Sets the hostname for the IP interface on which Directory Server listens for SSL connections. This attribute is multi-valued. Default behavior is to listen on all interfaces. Consider this attribute in the same way as `nsslapd-listenhost`.

Table 6-2 Tuning Recommendations for Configuring Use of System Resources *(Continued)*

Attribute (on dn: cn=config)	Short Description and Tuning Recommendations
`nsslapd-threadnumber`	Sets the number of threads Directory Server uses. Consider adjusting the value of this attribute if any of the following are true: • Client applications perform many simultaneous, time-consuming operations such as updates or complex searches. • Directory Server supports many simultaneous client connections. 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 + number of simultaneous updates. Consider also adjusting the maximum number of threads per client connection, `nsslapd-maxthreadsperconn`, as discussed in Table 6-1. 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. Verify estimates through empirical testing. Results depend not only on the particular deployment situation but also on the underlying system.

Refer to the *Directory Server Administration Reference* for information about individual configuration attributes.

Managing Access Control

Directory Server now offers performance and scalability improvements for Access Control Instructions (ACIs) such as better memory management and support for macro ACIs. Improvements notwithstanding, Directory Server uses significant system resources to evaluate complex ACIs. Extensive use of complex ACIs can therefore negatively impact performance.

Macro ACIs help you limit the number of ACIs used. By limiting the number of ACIs, you render access control easier to manage and reduce the load on the system. Macros are placeholders that represent a DN, or a portion of a DN, in an ACI. A macro can be used in an ACI target, in an ACI bind rule, or in both. When Directory Server receives a request, it checks which ACI macros match against the resource targeted for the resulting operation. If a macro matches, Directory Server replaces it with the value of the actual DN. Directory Server then evaluates the ACI normally. For more information on ACIs, refer to the *Directory Server Administration Guide*.

Testing has demonstrated that Directory Server can support more than 50,000 ACIs. The impact on performance for various deployment scenarios is currently under analysis. Keep the number of ACIs as small as possible to limit negative impact on performance, and to reduce the complexity of managing access controls. For deployments involving complex ACI environments, consider using Sun Java System Directory Proxy Server to provide some access control features.

Configuring Server Plug-Ins

Directory Server implements many key features such as access control, replication, syntax checking, and attribute uniqueness using plug-ins. In the context of a particular deployment, you may find it useful to reconfigure some plug-ins. The recommendations in Table 6-3 address settings for some standard plug-ins.

Table 6-3 Tuning Recommendations for Some Standard Plug-Ins

Name and DN	Short Description and Tuning Recommendations
7-Bit Check Plug-In `dn: cn=7-bit check,cn=plugins,cn=config`	Allows Directory Server to check that attribute values are 7-bit clean. That is, that attribute values provided contain only those characters that fit in 7-bit encoding. You may choose to disable this plug-in (default `on`) if the infrastructure is designed to support wider encodings such as Japanese characters, for example.
Legacy Replication Plug-In dn: `cn=Legacy Replication Plugin,cn=plugins,cn=config`	Allows Directory Server to function as a consumer of a 4.x supplier. Unless you intend to use Directory Server as a consumer of a 4.x supplier during an upgrade for example, turn this plug-in off (`on` by default in case 4.x replication capabilities are required).

Table 6-3 Tuning Recommendations for Some Standard Plug-Ins *(Continued)*

Name and DN	Short Description and Tuning Recommendations
Referential Integrity Plug-In dn: `cn=referential integrity postoperation,cn=plugins,cn=config`	Allows Directory Server to ensure relationships between related entries are maintained. For example, when a user entry is removed from the directory or renamed, the groups to which the user belonged are updated as needed without manual intervention. Enable and configure this plug-in on all masters. Set the `nsslapd-pluginarg0` to a positive value, such as `10` (seconds) to ensure that work performed by this plug-in happens asynchronously, rather than synchronously. When enabling the plug-in, also create equality indexes for all attributes configured for use with the plug-in. The plug-in uses such indexes when searching for entries to update. Without equality indexes for the attributes it uses, the plug-in must perform costly unindexed searches that have negative impact on performance. Refer to the *Directory Server Administration Guide* for instructions on configuring and enabling the plug-in.

Refer to the *Directory Server Administration Reference* for information about individual configuration attributes.

Chapter 7

Tuning Class of Service

Directory Server provides Class of Service (CoS) functionality to facilitate centralized management of common attribute values. For example, many users may share the same fax number as an attribute of their entries; CoS lets you store that number in one entry, then generate it for the other entries. As a result, you only have to keep track of the fax number in one place, but client applications can search for it on any entry to which it applies.

The disadvantage of CoS is the performance penalty paid for the convenience of having Directory Server generate attribute values for you. In many cases, the advantages of centralized management of common attribute values clearly outweigh the cost disadvantages. Yet it helps to have some idea how CoS works in order to make that comparison.

This chapter covers how Directory Server implements CoS to help you gauge whether a particular use of CoS fits your performance requirements. It includes the following sections:

- How Class of Service Works
- Implementing CoS for Best Performance

How Class of Service Works

This section takes a look at how CoS works from the inside. To gain a clear understanding of the functionality CoS provides and to find information on using CoS in your deployment, refer to the *Directory Server Deployment Planning Guide*. For step by step instructions on how to set up CoS, refer to the *Directory Server Administration Guide*.

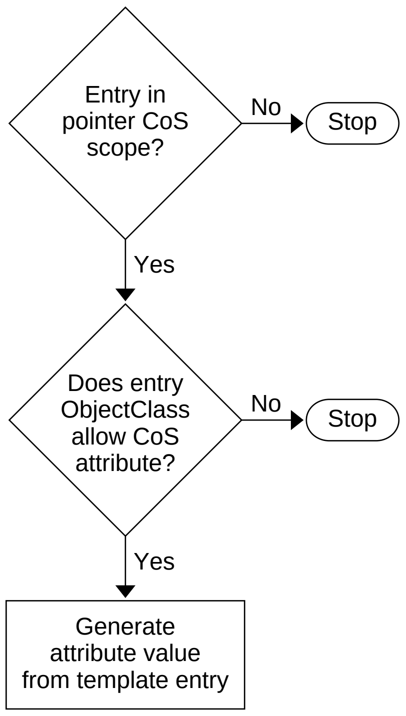
Directory Server supports three types of CoS functionality: pointer CoS, indirect CoS, and classic CoS. For each type of CoS, template entries provide the actual attribute values, and definition entries specify the relationship between the template entries and the target entries on which Directory Server generates attribute values. The following sections cover CoS operation:

- Pointer CoS
- Indirect CoS
- Classic CoS
- CoS Ambiguity

Pointer CoS

Pointer CoS definition entries point to a template entry holding the attribute value used. Recall also that a CoS definition entry targets all entries in the subtree where the definition is located. For pointer CoS, Directory Server therefore generates the same attribute value for all entries in the subtree. Figure 7-1 shows how Directory Server generates each CoS attribute value.

Figure 7-1 Pointer CoS Operation


Entry in pointer CoS scope?
No
Stop
Yes
Does entry ObjectClass allow CoS attribute?
No
Stop
Yes
Generate attribute value from template entry

As shown, pointer CoS attribute generation involves:

1. Checking whether the entry is in the scope of the pointer CoS definition.
2. Checking whether the object class(es) for the target entry allow that entry to hold the attribute specified by the pointer CoS definition.

   In other words, if the pointer CoS definition provides for fax numbers, can the entry in question hold a fax number attribute?

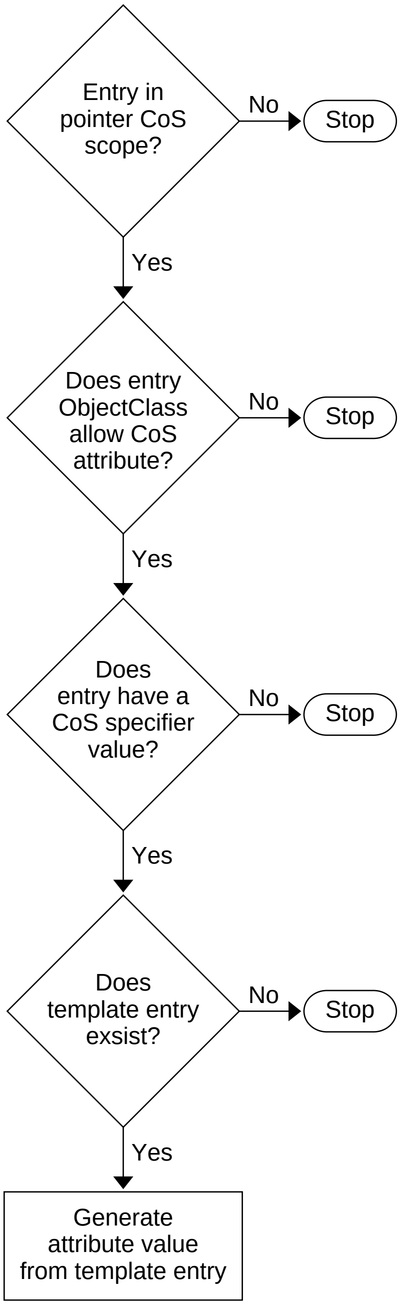
   If you turn `nsslapd-schemacheck` on `cn=config` to `off`, Directory Server skips this step.
3. Generating the attribute value from the template entry onto the target entry.

As pointer CoS definitions directly identify template entries, Directory Server can cache attribute values to generate. Pointer CoS therefore typically has lower performance cost than indirect or classic CoS configurations involving the same number of CoS definitions.

Indirect CoS

Indirect CoS definition entries specify both the attribute to generate and the attribute in the target entry identifying the template entry. You might use Indirect CoS, for example, to generate fax numbers for employees in `ou=People,dc=example,dc=com` where fax number depends on building code. Figure 7-2 shows how Directory Server generates each CoS attribute value.

Figure 7-2 Indirect CoS Operation


Entry in pointer CoS scope?
No
Stop
Yes
Does entry ObjectClass allow CoS attribute?
No
Stop
Yes
Does entry have a CoS specifier value?
No
Stop
Yes
Does template entry exsist?
No
Stop
Yes
Generate attribute value from template entry

As shown, indirect CoS attribute generation involves:

1. Checking whether the entry is in the scope of the indirect CoS definition.
2. Checking whether the object class(es) for the target entry allow that entry to hold the attribute specified by the indirect CoS definition.

   If you turn `nsslapd-schemacheck` on `cn=config` to `off`, Directory Server skips this step.
3. Retrieving the value of the target entry attribute identified by the CoS indirect specifier attribute of the definition entry.

   This attribute value is the DN of the template entry.
4. Looking up the template entry using the attribute value retrieved in Step 3.
5. Generating the attribute value from the template entry onto the target entry.

Directory Server can cache indirect CoS definition entries efficiently. It does not cache indirect CoS template entries in any special way. As template indirect CoS template entries depend entirely on target entry attribute values, the cost of maintaining such a cache could be prohibitively expensive. Looking up template entries can of course work faster if Directory Server retrieves them from entry cache in RAM, rather than from disk storage.

Indirect CoS typically has higher performance cost than both pointer CoS and classic CoS for configurations involving the same number of CoS definitions.

Classic CoS

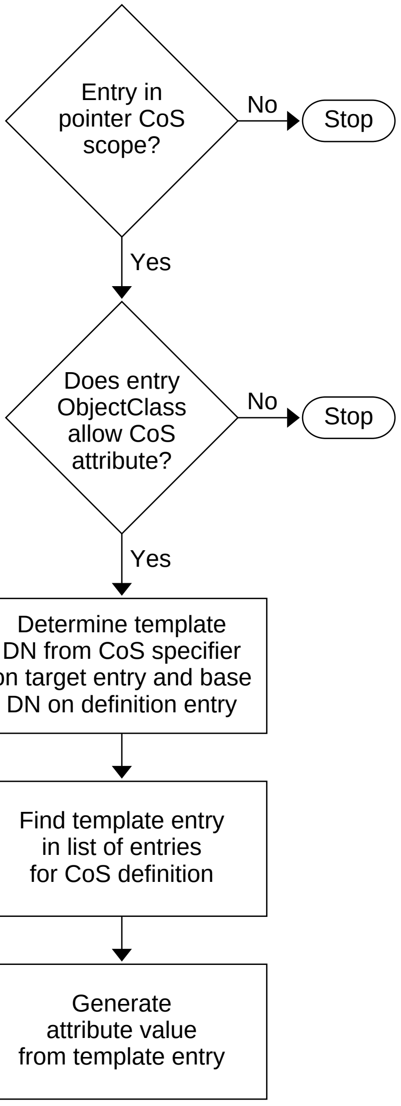
Classic CoS definition entries provide similar functionality to indirect CoS definition entries. Instead of specifying a target entry attribute that then fully identifies the template entry by DN, however, classic CoS definition entries specify the base DN of the template entry and the CoS specifier attribute on the target entry that further identifies the template entry under the base DN that holds the CoS attribute value to generate. You might use classic CoS, for example, to generate one set of fax numbers for managers, fax machines used for confidential information perhaps, and another set of fax numbers for other employees, where fax number depends on the building code.

As each classic CoS definition entry specifies a base DN to identify template entries, each classic CoS definition potentially relates to a large number of templates that Directory Server can identify in advance. Directory Server caches the list of templates for each classic CoS definition entry.

To optimize the cache structure for fast lookup, Directory Server maintains a hash when more than 10 templates correspond to a definition. Directory Server builds hash keys from the values of the attributes identified as `cosSpecifier` attribute values (the RDNs of the template entries). To avoid clashes, where several templates correspond to the same hash key, ensure the RDNs of the template entries are significantly different. You can monitor CoS through CoS monitoring attributes described in the *Directory Server Administration Reference*.

Figure 7-3 shows how Directory Server generates each CoS attribute value.

Figure 7-3 Classic CoS Operation


Entry in pointer CoS scope?
No
Stop
Yes
Does entry ObjectClass allow CoS attribute?
No
Stop
Yes
Determine template DN from CoS specifier on target entry and base DN on definition entry
Find template entry in list of entries for CoS definition
Generate attribute value from template entry

As shown, classic CoS attribute generation involves:

1. Checking whether the entry is in the scope of the classic CoS definition.

2. Checking whether the object class(es) for the target entry allow that entry to hold the attribute specified by the classic CoS definition.

   If you turn `nsslapd-schemacheck` on `cn=config` to `off`, Directory Server skips this step.

3. Determining the template entry DN using the base DN specified in the definition entry and the specifier value in the target entry.
4. Looking up the template entry in the list of template entries under the base DN specified in the definition entry.
5. Generating the attribute value from the template entry onto the target entry.

Each classic CoS attribute value generation therefore requires several lookups. The performance cost of classic CoS typically costs less than indirect CoS and more than pointer CoS configurations involving the same number of CoS definitions.

CoS Ambiguity

Nothing in Directory Server prevents you from creating multiple CoS definition entries that each generate a value for the same single-valued attribute on a given target entry. The *Directory Server Administration Guide* explains that when multiple definitions of identical CoS priority can apply Directory Server picks one arbitrarily. By default, unless you specify otherwise, all CoS definitions have the same priority.

Directory Server logs warning messages when forced to make an arbitrary distinction among multiple applicable definition entries. This logging capability was not provided in earlier versions of Directory Server. Such warning messages takes the form:

```
Definition defDN1 and definition defDN2 compete to provide attribute 'type' at priority level
```

You can also configure Directory Server to log informational messages when it is forced to make an arbitrary distinction among multiple, potentially applicable definition entries. To do so, set the log level high enough to include informational messages from plug-ins. Note that this can result in a heavy logging load, so you might not want to set logging that high on a production server instance. The content of informational messages takes the following form:

```
Definition defDN1 and definition defDN2 potentially compete to provide attribute 'type' at
priority level
```

You can then choose whether to resolve such cases of CoS ambiguity by setting CoS priorities appropriately on the definition entries.

Implementing CoS for Best Performance

This section examines some implications of how CoS works on the ways you might naturally use CoS functionality as intended, and avoid performance pitfalls.

- When Many Entries Share the Same Value
- When Entries Have Natural Relationships
- Avoid Thousands of CoS Definitions

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

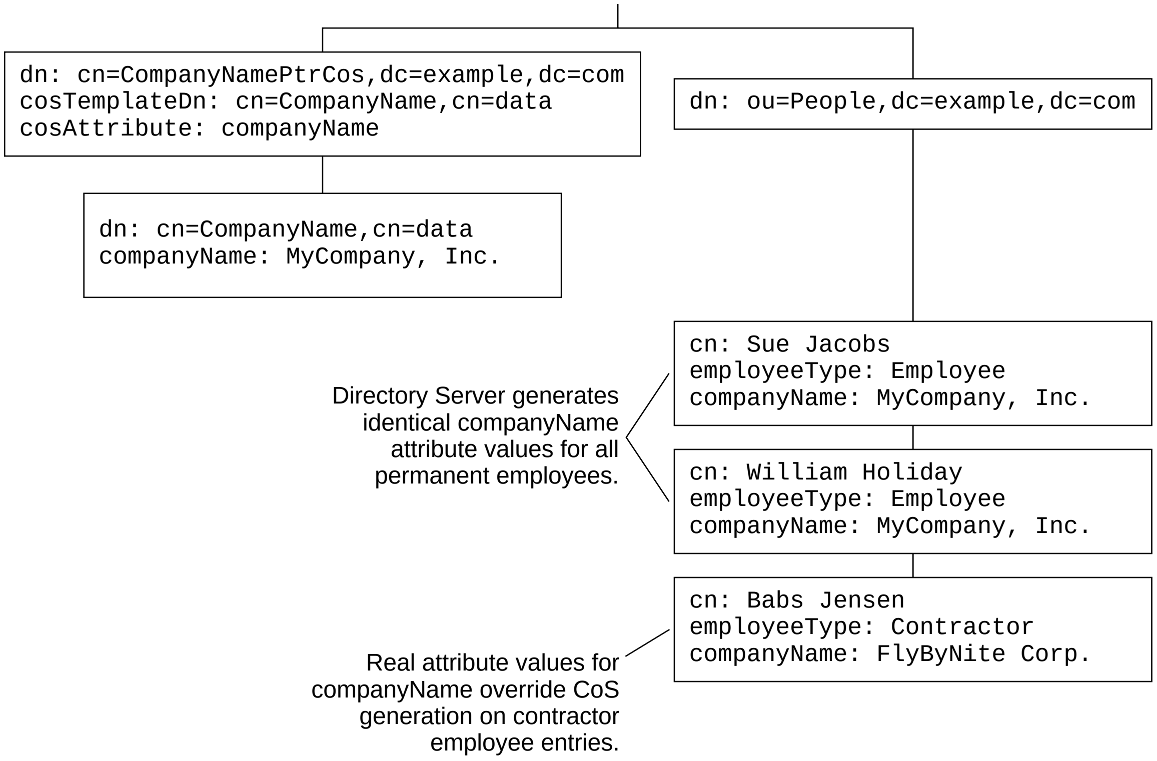
Sometimes you cannot prevent developers from creating clients that search for `objectclass=*`, and then examine a big result set on the client when they actually need only an email address, for example. If you can influence how client applications are written, however, work to convince directory client developers that their applications will perform much better when looking up only those attribute values they actually need.

When Many Entries Share the Same Value

CoS provides big benefits for relatively low cost when you need the same attribute value to appear on a large number of 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`. Figure 7-4 demonstrates this with pointer CoS. Notice here that CoS generates `companyName` values for all permanent employees without overriding real, not CoS generated, `companyName` values stored for contractor employees.

Figure 7-4 Generating CompanyName With Pointer CoS


dn: cn=CompanyNamePtrCos,dc=example,dc=com
cosTemplateDn: cn=CompanyName,cn=data
cosAttribute: companyName
dn: ou=People,dc=example,dc=com
dn: cn=CompanyName,cn=data
companyName: MyCompany, Inc.
cn: Sue Jacobs
employeeType: Employee
companyName: MyCompany, Inc.
Directory Server generates identical companyName attribute values for all permanent employees.
cn: William Holiday
employeeType: Employee
companyName: MyCompany, Inc.
cn: Babs Jensen
employeeType: Contractor
companyName: FlyByNite Corp.
Real attribute values for companyName override CoS generation on contractor employee entries.

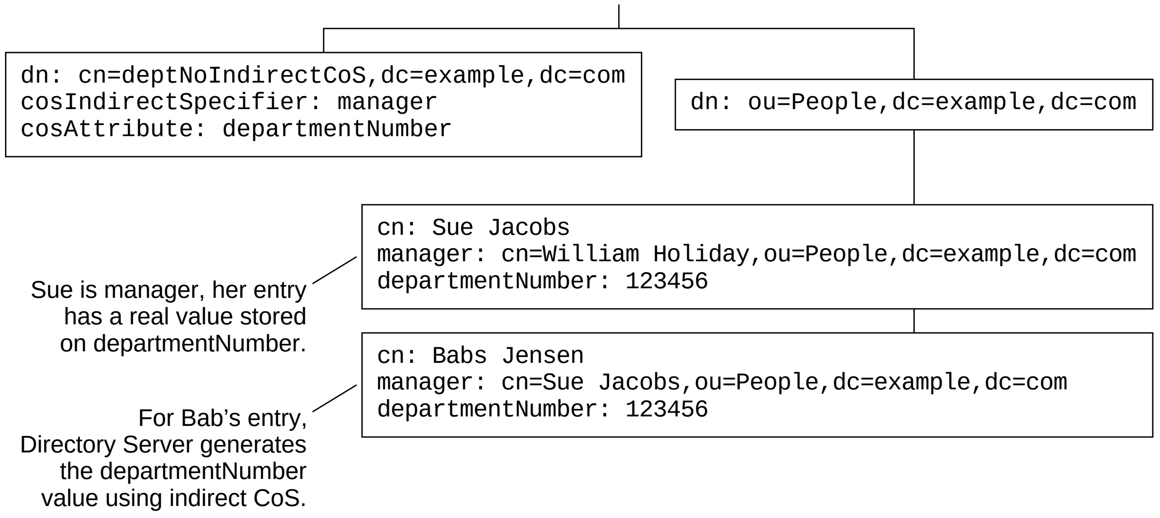
In cases where many, many entries share the same value, pointer CoS works particularly well. The ease of maintaining companyName for permanent employees clearly offsets the additional processing cost of generating attribute values. Deep directory information trees (DIT) tend to bring entries sharing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

When Entries Have Natural Relationships

CoS also provides big data administration benefits where directory data has natural relationships.

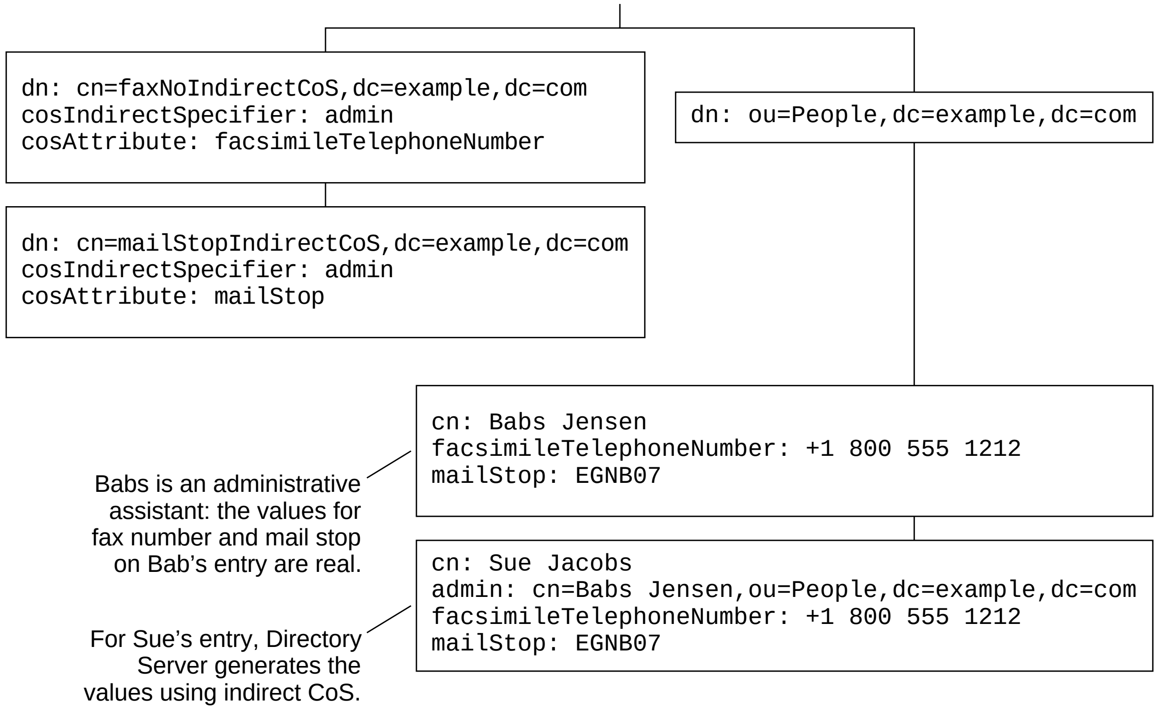
Consider an enterprise directory for example in which every employee has a manager, and every employee shares a mail stop and fax number with the nearest administrative assistant. Figure 7-5 and Figure 7-6 demonstrate use of indirect CoS to retrieve department number from the manager entry and mail stop and fax number from the administrative assistant entry, respectively.

Figure 7-5 Generating `DepartmentNumber` With Indirect CoS


dn: cn=deptNoIndirectCoS,dc=example,dc=com
cosIndirectSpecifier: manager
cosAttribute: departmentNumber
dn: ou=People,dc=example,dc=com
cn: Sue Jacobs
manager: cn=William Holiday,ou=People,dc=example,dc=com
departmentNumber: 123456
cn: Babs Jensen
manager: cn=Sue Jacobs,ou=People,dc=example,dc=com
departmentNumber: 123456
Sue is manager, her entry has a real value stored on departmentNumber.
For Bab's entry, Directory Server generates the departmentNumber value using indirect CoS.

In this implementation, manager's entries have real values for `departmentNumber`, and these real values override generated values. Directory Server does not generate attribute values from CoS generated attribute values. As a result, for the example shown in Figure 7-5 department number attribute values need to be managed only on manager entries. Likewise, for the example shown in Figure 7-6 mail stop and fax number attributes need to be managed only on administrative assistants' entries.

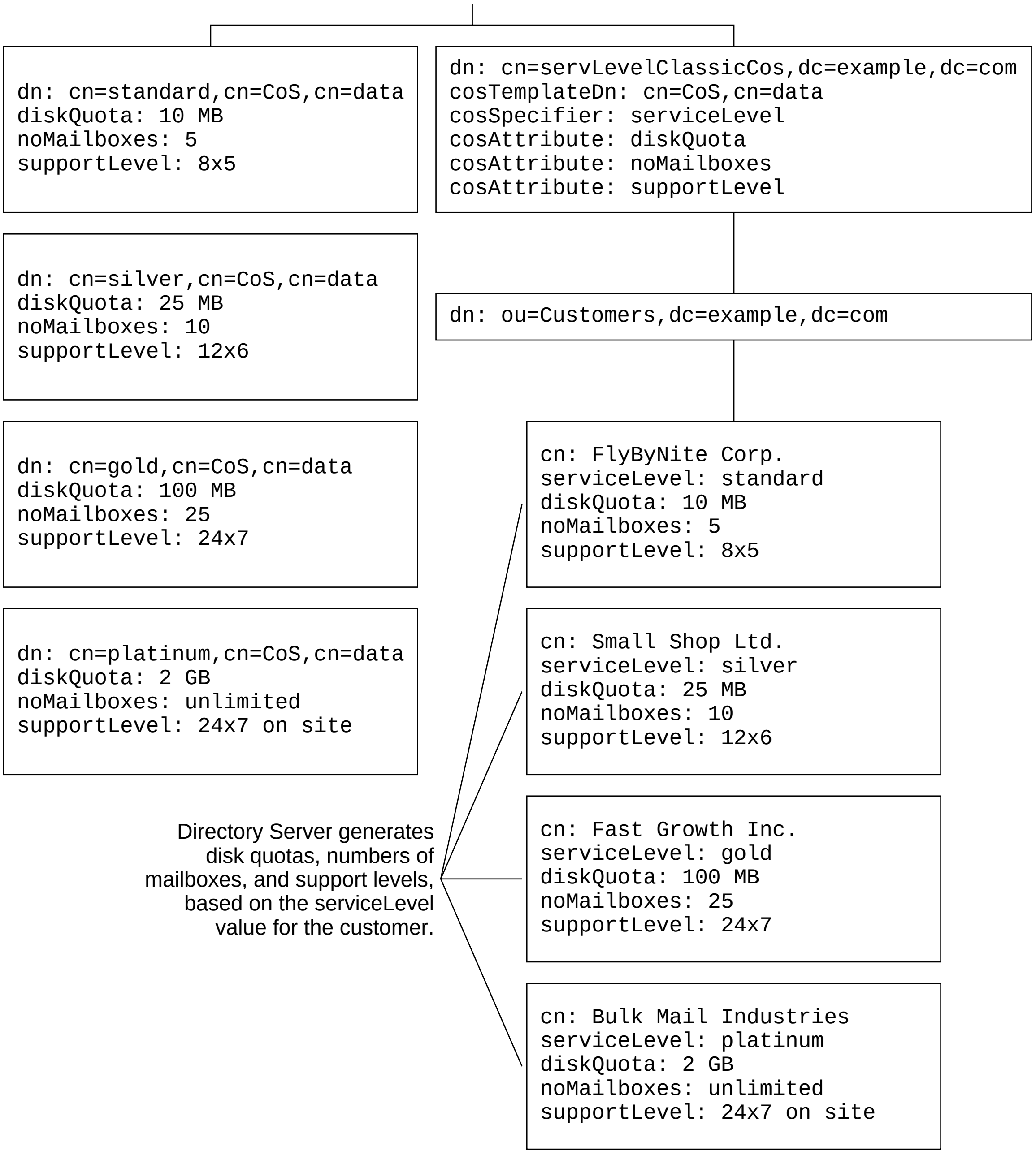
Figure 7-6 Generating Mail Stop and Fax Number With Indirect CoS


dn: cn=faxNoIndirectCoS,dc=example,dc=com
cosIndirectSpecifier: admin
cosAttribute: facsimileTelephoneNumber
dn: cn=mailStopIndirectCoS,dc=example,dc=com
cosIndirectSpecifier: admin
cosAttribute: mailStop
dn: ou=People,dc=example,dc=com
cn: Babs Jensen
facsimileTelephoneNumber: +1 800 555 1212
mailStop: EGNB07
cn: Sue Jacobs
admin: cn=Babs Jensen,ou=People,dc=example,dc=com
facsimileTelephoneNumber: +1 800 555 1212
mailStop: EGNB07
Babs is an administrative assistant: the values for fax number and mail stop on Bab's entry are real.
For Sue's entry, Directory Server generates the values using indirect CoS.

Notice that 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 offering customers standard, silver, gold, platinum packages. A customer's disk quota, number of mailboxes, and rights to prepaid support levels depends on the service level purchased. Figure 7-7 demonstrates how a classic CoS scheme enables this.

Figure 7-7 Generating Service Level Data With Classic CoS



Notice one CoS definition may be associated with multiple CoS template entries.

Avoid Thousands of CoS Definitions

As described in “Classic CoS” on page 86, Directory Server optimizes for the case where one classic CoS definition entry is associated with multiple CoS template entries. Directory Server does not, however, optimize for the case where many, many CoS definitions potentially apply. Instead, Directory Server checks each CoS definition to determine whether it applies. This behavior leads to performance problems when you define thousands of CoS definitions.

This situation can arise in a modified version of the example shown in Figure 7-7. Consider an Internet service provider offering 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 therefore means creating 1000 classic CoS definitions, with the consequent performance hit as Directory Server 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, where customers’ customers’ entries identify the entries defining their class of service allotments.

With or without thousands of CoS definitions, once you start approaching the limit of having different CoS schemes for every target entry or two, you are better off paying the management price of updating the real values, thereby gaining the performance price of reading real, rather than CoS-generated values.

Glossary

Refer to the *Java Enterprise System Glossary* (`http://docs.sun.com/doc/816-6873`) for a complete list of terms that are used in this documentation set.

Index