Tuning Guide for UNIX* Platforms

eDirectory™ 8.8 SP7

April 27, 2012

Novell.

Legal Notices

Any products or technical information provided under this Agreement may be subject to U.S. export controls and the trade laws of other countries. You agree to comply with all export control regulations and to obtain any required licenses or classification to export, re-export or import deliverables. You agree not to export or re-export to entities on the current U.S. export exclusion lists or to any embargoed or terrorist countries as specified in the U.S. export laws. You agree to not use deliverables for prohibited nuclear, missile, or chemical biological weaponry end uses. See the Novell International Trade Services Web page (http://www.novell.com/info/exports/) for more information on exporting Novell software. Novell assumes no responsibility for your failure to obtain any necessary export approvals.

Novell, Inc.
1800 South Novell Place
Provo, UT 84606
U.S.A.
www.novell.com

Online Documentation: To access the latest online documentation for this and other Novell products, see the Novell Documentation Web page (http://www.novell.com/documentation).

Novell Trademarks

For Novell trademarks, see the Novell Trademark and Service Mark list (http://www.novell.com/company/legal/trademarks/tmlist.html).

Third-Party Materials

Contents

About This Guide

This guide describes how to analyze and tune Novell eDirectory on UNIX platforms to yield superior performance in all deployments.

This guide introduces the following:

- Chapter 1, "Overview," on page 7
- Chapter 2, "eDirectory Subsystems," on page 9
- Chapter 3, "Analyzing System Bottlenecks," on page 13
- Chapter 4, "Tuning eDirectory Subsystems," on page 17
- Chapter 5, "eDirectory Configuration," on page 25

Audience

The guide is intended for network administrators.

Feedback

We want to hear your comments and suggestions about this manual and the other documentation included with this product. Please use the User Comments feature at the bottom of each page of the online documentation and enter your comments there.

Documentation Updates

For the most recent version of the *Novell eDirectory 8.8 SP7 Tuning Guide for UNIX Platforms*, see the Novell eDirectory 8.8 online documentation (http://www.novell.com/documentation/edir88/) Web site.

Additional Documentation

For more information about eDirectory 8.8 SP7, refer to the following:

- *Novell eDirectory 8.8 SP7 Installation Guide* (http://www.novell.com/documentation/edir88/edirin88/data/a2iii88.html)
- *Novell eDirectory 8.8 SP7 Administration Guide* (http://www.novell.com/documentation/edir88/edir88/data/a2iii88.html)
- *Novell eDirectory 8.8 SP7 What's New Guide* (http://www.novell.com/documentation/edir88/edir88new/data/front.html)
- *Novell eDirectory 8.8 SP7 Troubleshooting Guide* (http://www.novell.com/documentation/edir88/edir88tshoot/data/front.html)

These guides are available at Novell eDirectory 8.8 documentation Web site (http://www.novell.com/documentation/edir88/index.html).

For information about the eDirectory management utility, see the *Novell iManager 2.7 Administration Guide* (http://www.novell.com/documentation/imanager27/index.html).

1 Overview

Novell eDirectory 8.8 is a standards-compliant, cross-platform, highly scalable, fault-tolerant, and high-performance directory services solution. This guide provides information on tuning your eDirectory environment for improved performance.

Tuning for performance is a complex activity. It requires understanding of both the eDirectory and operating system's subsystems. It involves monitoring the system to identify bottlenecks and fixing them one at a time. Many a times resources are limited and tuning is confined to eDirectory and the operating system.

In this guide, read the Prerequisites section before attempting any kind of tuning, then proceed to the other sections. eDirectory Subsystems chapter describes primary subsystems that influence eDirectory performance. Analyzing System Bottlenecks chapter describes various system resources and their influence on eDirectory performance. Tuning eDirectory Subsystems chapter describes how to analyze and tune eDirectory under various conditions and environments. Finally, the eDirectory Configuration chapter describes how to configure various tunable parameters.

1.1 Prerequisites

Ensure that the following general prerequisites are met before attempting to tune the system for performance:

- A good eDirectory tree design (http://www.novell.com/documentation/edir88/edir88/data/a2iiido.html) can enhance eDirectory performance. The following considerations might apply:
  - Applications read all the information locally on the server without needing to chain the requests.
  - eDirectory efficiently handles object references automatically. If possible, objects on a server should not refer to objects that are not local on that server, because maintaining non-local object references can take more time. If such references exist, backlinks must be maintained. This becomes cumbersome in large deployments.
  - If you need a group with 10,000 members or more, dynamic groups are recommended. This allows you to avoid the overhead associated with maintaining references for so many people. Choose your dynamic group configuration carefully, because using multiple dynamic groups with improper search criteria might overload the server and reduce overall server performance. If a search operation takes a long time to complete, the chosen index might be inefficient. Minimize the use of regular (static) groups as this can increase tree walking on login.
  - Use ACLs efficiently. For example, use the [This] trustee and assign it at the container level instead of using an ACL template that assigns rights to itself. The fewer ACLs, the better the performance. For more information on ACLs, refer to "eDirectory Rights" (http://www.novell.com/documentation/edir88/edir88/data/fbachifb.html) in the *Novell eDirectory 8.8 SP7 Administration Guide*.
  - Distribute the load onto multiple replica servers.

  - Although a good tree design minimizes the need for tree walking, it is still sometimes necessary. Advanced Referral Costing (http://www.novell.com/documentation/edir88/edir88/data/b9u7705.html) can be considered.
  - If logins are slow, you can disable login updates. There are separate ways to disable login updates for both NDS® and Novell Modular Authentication Service™ (NMAS) logins. However, it is important to understand the security implications (http://www.novell.com/documentation/nmas33/admin/data/bg8dphs.html).
- Run health checks (http://www.novell.com/documentation/edir88/edir88/data/a5ziqam.html) through iMonitor. Ensure the following:
  - Time is in sync across all replica servers.
  - Replica synchronization and background processes are in a healthy state.

2 eDirectory Subsystems

This section discusses the eDirectory Subsystems.

- Section 2.1, "FLAIM Database," on page 9
- Section 2.2, "Thread Pool," on page 11

2.1 FLAIM Database

eDirectory uses FLAIM as its database. FLAIM (Flexible Adaptable Information Manager) is used for traditional, volatile, and complex information. It is a very scalable database engine that supports multiple readers and a single-writer concurrency model. Readers do not block writers and writers do not block readers.

Physically, FLAIM organizes data in blocks. Some of the blocks are typically held in memory. They represent the block cache. The entry cache (sometimes called a record cache) caches logical entries from the database. Entries are constructed from the items in the block cache. FLAIM maintains hash tables for both caches. The hash bucket size is periodically adjusted based on the number of items.

By default eDirectory uses a block size of 4 KB. The block cache size for caching the complete DIB is equal to the DIB size, and the size required for the entry cache is about two to four times the DIB size.

While retrieving an entry, FLAIM first checks for the entry in the entry cache. If the entry exists, reading from the block cache isn't necessary. While retrieving a block from the disk, FLAIM first checks for the block in the cache. If the block exists, a disk read operation isn't necessary.

When an entry is added or modified, the corresponding blocks for that entry are not directly committed to the disk, so the disk and memory might not be in sync. However, the updates made to the entry are logged to the roll-forward log (RFL). An RFL is used to recover transactions after a system failure.

Least Recently Used (LRU) is the replacement algorithm used for replacing items in the cache.

- Section 2.1.1, "Checkpoint," on page 9
- Section 2.1.2, "Indexes," on page 10
- Section 2.1.3, "Roll-Forward Log," on page 10

2.1.1 Checkpoint

A checkpoint brings the on-disk version of the database to the same coherent state as the in-memory (cached) database. FLAIM can perform a checkpoint during the minimal update activity on the database. It runs every second and writes the dirty blocks (dirty cache) to the disk. Blocks that are modified in the cache but not yet written to the disk are called "dirty blocks". FLAIM acquires a lock on the database and performs the maximum amount of possible work until either the checkpoint completes or another thread is waiting to update the database. To prevent the on-disk database from becoming too far out of sync, there are conditions under which a checkpoint is forced even if threads are waiting to update the database:

- If the checkpoint thread cannot complete a checkpoint within a specified time interval (the default is 3 minutes), it is forced and the dirty cache is cleaned.
- If the size of the dirty cache is larger than the `maxdirtycache` (if set), a checkpoint is forced to bring down the dirty cache size to `mindirtycache` (if set) or to zero.

2.1.2 Indexes

An index is a set of keys arranged in a way that significantly speeds up the task of finding any particular key within the index. Index keys are constructed by extracting the contents of one or more fields (attributes) from the entries. Indexes are maintained in the block cache. Any changes to the indexed attributes requires changes in the index blocks.

eDirectory defines a default set of indexes for system attributes (fields). System attributes such as `parentID` and `ancestorID` are used for one-level and subtree searches. These indexes cannot be suspended or deleted. The directory internally uses them. Default indexes are defined for attributes such as `CN`, `Surname`, `Given Name`, and so on. Indexes can be of type presence, value, and substring indexes. These indexes can be suspended. On deletion they are automatically re-created.

You can use iManager or the ndsindex Lightweight Directory Access Protocol (LDAP) utility to create indexes. Indexes (http://www.novell.com/documentation/edir88/edir88/data/a5tuuu5.html) are server-specific.

By enabling the RECM tag in DSTrace (ndstrace), you can view the index chosen for the search queries.

The following example is for a DSTrace log for a subtree search using `“cn=admin”`, CN.

```
3019918240 RECM: Iter #b239c18 query ((Flags&1)==1) && ((CN$217A$.Flags&8=="admin")
&& (AncestorID==32821))
```

```
3019918240 RECM: Iter #b239c18 index = CN$IX$220
```

The following example is for an DSTrace log for a subtree search using `“Description= This is for testing”`, `AncestorID`.

```
2902035360 RECM: Iter #83075b0 query ((Flags&1)==1) &&
((Description$225A$.Flags&8=="This is for testing") && (AncestorID==32821))
```

```
2902035360 RECM: Iter #83075b0 index = AncestorID_IX
```

2.1.3 Roll-Forward Log

FLAIM logs operations for each update transaction in a roll-forward log (RFL) file. An RFL is used to recover transactions from a system failure or when restoring from a backup. The RFL file is truncated after every checkpoint is completed unless it is turned on (`rflkeepfiles`) by using a hot continuous backup (http://www.novell.com/documentation/edir88/edir88/data/a2n4mb7.html).

2.2 Thread Pool

eDirectory is multi-threaded for performance reasons. In multi-threading, when the system is busy, more threads are created to handle the load and some threads are terminated to avoid extra overhead. It is inefficient and costly to frequently create and destroy threads. Instead of spawning new threads and destroying them for every task, a number of threads are started and placed in a pool. The system allocates the threads from the thread pool to several tasks as needed. Tasks are held in two types of queues:

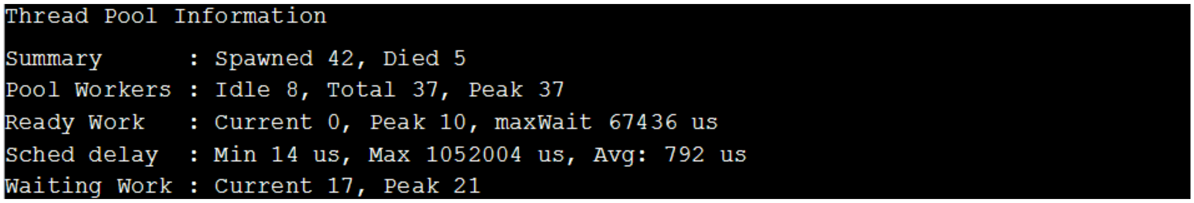
- Tasks that need immediate scheduling are held in the Ready queue.
- Tasks that need scheduling at a later time are held in the Waiting queue.

Not every module uses the thread pool. The actual number of threads for the process is more than the number that exists in the thread pool. For example, FLAIM manages its background threads separately.

Running the `ndstrace -c threads` command returns the following thread pool statistics:

- The total number of threads that are spawned, terminated, and idle.
- The total number of worker threads currently and the peak number of worker threads.
- The number of tasks and peak number of tasks in the Ready queue.
- The minimum, maximum and average number of microseconds spent in the Ready queue.
- The current and maximum number of tasks in the Waiting queue.

An example of a sample thread pool:



```
Thread Pool Information

Summary      : Spawned 42, Died 5
Pool Workers : Idle 8, Total 37, Peak 37
Ready Work   : Current 0, Peak 10, maxWait 67436 us
Sched delay  : Min 14 us, Max 1052004 us, Avg: 792 us
Waiting Work : Current 17, Peak 21
```

There are certain thread pool parameters:

- **n4u.server.max-threads:** Maximum number of threads that can be available in the pool.
- **n4u.server.idle-threads:** Maximum number of idle threads that can be available in the pool.
- **n4u.server.start-threads:** Number of threads started.

Run the `ndsconfig get` and `ndsconfig set` commands to get and set the thread pool size.

3 Analyzing System Bottlenecks

There are several system resources that influence eDirectory performance. In addition, upgrading to the latest version of operating system improves performance.

- Section 3.1, "Disk I/O Subsystem," on page 13
- Section 3.2, "CPU Subsystem," on page 14
- Section 3.3, "Memory Subsystem," on page 14
- Section 3.4, "Network Subsystem," on page 15

3.1 Disk I/O Subsystem

The disk subsystem is the most common bottleneck. The I/O takes a relatively long time with longer queues, resulting in high disk utilization and idle CPU cycles. Use the iostat tool during expected peak loads to determine the average response time indicators.

Disk read, write, and update operations can be sequential or random. Random reads and updates is the most common access pattern in eDirectory deployments.

Some solutions for random workloads:

- Increase the RAM. This allows caching frequently used data or read-ahead data at the filesystem layer. It also allows caching the DIB within the FLAIM subsystem.
- Use dedicated volumes for the DIB. Filesystem performance improves for volumes created closer to the spindle. Use dedicated volumes for RFL and other logs.
- As disks develop increasing latency over a period of time because of fragmentation, they should be defragmented.
- Add separate disk drives for FLAIM RFL. This type of logging can be performed on high-speed disks.
- Use a RAID 10(1+0) environment with more disk drives.

Files created by eDirectory can grow to 4 GB. Filesystems that are optimized to handle large files work efficiently with eDirectory.

- For Solaris™, the Veritas* VxFS filesystem is an extent-based file system where the file system metadata is optimized for large files. The UFS filesystem is indirectly block-based, where the filesystem metadata is stored in larger number of blocks. It can even be scattered for large files, which makes UFS slower for larger files.
- For Linux™, the Reiser filesystem is a fast journaling file system and performs better than the ext3 filesystem on large DIB sets. However, the write back journaling mode of ext3 is known to match the performance of the Reiser filesystem although the default ordered mode provides

better data consistency. XFS is a high-performance journaling file system, capable of handling large files and offering smooth data transfers. eDirectory 8.8 SP7 is supported on SLES 11 32 and 64-bit platforms having XFS file system.

FLAIM supports a block size of 4 KB and 8 KB. By default, it is 4 KB. This is same as the default block size on Linux (`tune2fs -l device`). However, on Solaris, the UFS filesystem is created with a default block size of 8 KB (`df -g mountpoint`). If the FLAIM block size is smaller than the filesystem block size, partial block writes can happen. If the database block size is larger than the filesystem block size, individual block reads and writes are split into a series of distinct physical I/O operations. Therefore, you should always keep the FLAIM block size the same as the filesystem block size.

Block sizes can be controlled only during the creation of the DIB. Add a line “blocksize=8192” to `_ndsdb.ini` to create the DIB with 8K block size.

Choosing the right block size depends on the average size of the FLAIM record on your deployments. Empirical testing is required on the right set of test data to determine which block size is better for your deployment.

3.2 CPU Subsystem

eDirectory is built on a highly scalable architecture. The performance increases with the increase in the number of processors. Increased throughput is observed until at least the 12th processor under heavy load. However, this increase is subject to the performance of other resources during the increasing load on the system. Servers are often under-configured with disks and memory. You should add more processors only under the following circumstances:

- If the average load on currently used processors is beyond 75% percent utilization. If the current CPU utilization is below 75%, adding more CPUs might not improve performance.
- If there is a satisfying increase in performance.

If eDirectory is configured with too many threads, considerable amount of CPU time is spent in context switching. In this case, a decrease in threads can result in better throughput.

3.3 Memory Subsystem

Server applications can perform significantly better when RAM is increased. Caching the eDirectory database in the filesystem or in the FLAIM cache can lead to improved performances of search and modify operations. However, you cannot cache the complete DIB in large deployments. Avoid page swapping even if it means reducing the FLAIM entry and block cache sizes. Use the vmstat tool to find more information on the memory subsystem.

As eDirectory uses memory, each thread from the thread pool uses 1 MB of RAM for its stack. By default, the FLAIM cache size is set to 200 MB.

Several loadable modules are started when eDirectory starts, but the loadable module architecture of eDirectory allows you to reduce the memory footprint of the process by not loading the unused modules (for example, SecretStore, LDAP, or eMBox). In addition, products like IDM have some modules that run inside eDirectory.

The memory used by eDirectory might appear to be growing. Although memory is freed by an eDirectory process, it might not be released to the system free pool because the memory manager used internally by eDirectory tries to optimize the memory allocations for future. This is one of the reasons for not recommending FLAIM dynamic configuration. Use the Top tool to find the approximate virtual memory size of the ndsd process in your deployment.

The maximum memory that can be allocated to a process is limited in several ways. A certain amount of RAM is used by the operating system and other processes on the system. The operating system can impose limitations on physical RAM that a process uses. For example, a 32-bit eDirectory limits the FLAIM cache to 2.5 GB, but with 64-bit eDirectory, FLAIM caches can grow beyond 2.5 GB. However, the 64-bit eDirectory needs an increased memory footprint (approximately 20%) than the 32-bit eDirectory.

3.4 Network Subsystem

Typical deployments have sufficient bandwidth to handle peak network load. Adequate bandwidth reduces errors, collisions, and dropped packets. Use the netstat tool to determine the network statistics.

Several operating systems provide TCP/IP tunable parameters for tuning network intensive servers. For information, refer to the documentation for the operating systems.

If the network is the bottleneck, you should increase the bandwidth. Configuring a dedicated private network between the application servers and the eDirectory server might also help in reducing the network congestion.

4 Tuning eDirectory Subsystems

This section includes the following information:

4.1 FLAIM Database

Cache sizing is arguably the most important factor affecting the overall performance of eDirectory. The greater the number of items (blocks and entries) that can be cached, the better the overall performance is. The percentage of times that the blocks or entries are found in the cache is called the hit ratio. A higher ratio results in better performance. iMonitor can be used to view the hit ratio.

The block cache is most useful for update operations. The entry cache is most useful for operations that performs a base-scoped search for an entry. However, both one-level and sub-tree scoped searches use the entry cache as well as the block cache. The block cache is used to retrieve indexes. Create the right type of indexes as necessary, for more information see "Choosing Indexes" on page 18.

A fault in the block cache can result in a disk read operation. Disk reads are always expensive, but they can be avoided if a block is retrieved from the filesystem cache.

The amount of memory required to cache the complete database in the block cache is nearly the size of the database on the disk, and the amount of memory required to cache the complete database in the entry cache is nearly two to four times the database size on the disk. When you have less memory on a system, try a smaller entry cache and a much larger block or filesystem cache.

If reads are localized to a set of entries in the directory, you should increase the entry cache as long as it results in an improved entry cache hit ratio.

If the read pattern is completely random and the DIB is much larger than the available RAM, you should have a larger block cache or a filesystem cache than the entry cache.

Any method you use to tune eDirectory for an improved performance needs empirical testing. A good ratio of entry to block cache for search-intensive environments is 2:1 ratio. Ensure that sufficient memory is left for other processes. Avoid page swapping even if it means reducing the FLAIM cache sizes.

Because FLAIM provides preallocated caching, memory allocated to the eDirectory cache is never fragmented by the native operating system memory manager.

4.1.1 Choosing Indexes

Indexes are meant to improve the one-level or sub-tree scoped search performance. Dynamic groups also use one-level or sub-tree scoped searches. Indexes are not used for base-scoped searches.

Because a Presence index does not differentiate between present and not present (deleted) values, it is mainly used for internal purpose. If applications run a Presence type search query, this index is never used, so applications should not have Presence indexes created for them.

Applications can create a Value index for an attribute, which is sufficient for most of the searches. FLAIM can use a Value index for performing both Presence as well as Substring searches on the attributes.

A Substring index can significantly decelerate the updates performed on an attribute. The number of index blocks required to support a Substring index is quite large compared to the Value index. This means more block cache is required to cache them. Create a Substring index only when necessary. A Value index should suffice for most searches. However, if Substring searches do not yield acceptable performance with a Value index, you can create a Substring index on those attributes.

If a search operation takes a long time to complete despite the chosen index, you might introduce a newer value index on one of the attributes of the search filter. Pick the attribute that yields best results when indexed.

4.1.2 Tuning for Updates

The block cache is most useful for update operations. Indexes also reside in the block cache. Although indexes help in faster searches, having too many indexes keeps the server busy maintaining them. Indexes are modified if attribute values are modified, added, or deleted. During large upload operations, indexes can be disabled for faster upload.

Having the RFL directory on a different disk than the DIB directory improves performance.

An acceptable limit for response time for an update operation can be controlled by using the maxdirtycache. For example, if an acceptable limit for the server response is 5 seconds and random disk write speed is 20 MB per second, then the maxdirtycache should be set as 20x5 = 100 MB. Ensure that the block cache can hold these dirty blocks in memory. See Section 5.2.2, “Modifying FLAIM Cache Settings through _ndsdb.ini,” on page 27 for more information.

4.2 Thread Pool

By default, the maximum number of threads that can be available in the thread pool is 128. This number should suffice for most deployments. It can be increased to 512 threads in larger deployments. You should increase the number of threads in the pool in the following cases:

- If the number of idle threads is often zero.
- If the average amount of time spent by a task in the Ready queue is high and increasing.
- If the number of tasks in the Ready queue is high and increasing.

Keep increasing the max threads if the performance of the server increases. It should also result in increased CPU utilization.

4.3 ACLs

- Section 4.3.1, “Improving eDirectory Searches and Reads,” on page 19
- Section 4.3.2, “Disabling ACL Templates,” on page 19

4.3.1 Improving eDirectory Searches and Reads

An LDAP search in eDirectory returns results depending on the number of attributes returned for a user (inetOrgPerson).

When an object is created in eDirectory, default ACLs might be added on the object. This depends on ACL templates in the schema definition for the objectClass to which this object belongs. For example, in the default configuration for inetOrgPerson, there can be up to six ACLs added on the user object. When an LDAP search request is made to return this user object with all attributes, it takes slightly longer to return this object to the client than returning this user object without ACL attributes.

Though default ACLs can be turned off, administrators may not want to turn them off because they are required for better access control. However, you can improve the search performance by not requesting them or by marking them as read filtered attributes. These changes do not break any applications because most applications use effective privileges and do not rely on specific ACLs.

Not requesting ACLs: An ACL attribute is not needed by several applications, so the applications can be modified to request specific attributes in which the application is interested. This results in better performance of the LDAP search.

Marking an ACL as read filtered: If an application cannot be modified, the *arf_acl.ldif* can be used by an administrator to mark the ACL attribute as a read filtered attribute. When the ACL is marked as a read filtered attribute, the server does not return the attribute on the entry if all attributes are requested. However, the if the LDAP search is done to return operational attributes or if the request specifically asks for ACL attributes, the marked attribute is returned. *rrf_acl.ldif* can be used to turn off the read filtered flag on an ACL attribute. These LDIFs affect the ACL attribute on the schema, so only a user with Supervisor rights on tree root can extend them.

By default, an ACL is not marked as read filtered, so the performance benefit for requests to return all attributes is not seen.

The following table depicts the location of *arf_acl.ldif* and *rrf_acl.ldif* files in different platforms.

Platform	Location
UNIX	◆ `/opt/novell/eDirectory/lib/nds-schema/`
NetWare	◆ `<unzipped_location>\nw\sys\system\schema`
Windows	◆ `<unzipped_location>\nt\I386\NDSonNT\ndsnt\nds`

4.3.2 Disabling ACL Templates

You can disable the Access Control List (ACL) templates to increase the bulkload performance. The implication of this is that some of the ACLs will be missing; however, you can resolve this by adding the required ACLs to the LDIF file or applying them later.

1. Run the following command:

```
ldapsearch -D cn_of_admin -w password -b cn=schema -s base
objectclasses=inetorgperson
```

The output of this command would be as follows:

```
dn: cn=schema

objectClasses: (2.16.840.1.113730.3.2.2 NAME 'inetOrgPerson' SUP
organizationalPerson STRUCTURAL MAY (groupMembership $ ndsHomeDirectory
$ loginAllowedTimeMap $ loginDisabled $ loginExpirationTime $
loginGraceLimit $ loginGraceRemaining $ loginIntruderAddress $
loginIntruderAttempts $ loginIntruderResetTime $
loginMaximumSimultaneous $ loginScript $ loginTime $
networkAddressRestriction $ networkAddress $ passwordsUsed $
passwordAllowChange $ passwordExpirationInterval $
passwordExpirationTime $ passwordMinimumLength $ passwordRequired $
passwordUniqueRequired $ printJobConfiguration $ privateKey $ Profile $
publicKey $ securityEquals $ accountBalance $ allowUnlimitedCredit $
minimumAccountBalance $ messageServer $ Language $ UID $
lockedByIntruder $ serverHolds $ lastLoginTime $ typeCreatorMap $
higherPrivileges $ printerControl $ securityFlags $ profileMembership $
Timezone $ sASServiceDN $ sASSecretStore $ sASSecretStoreKey $
sASSecretStoreData $ sASPKIStoreKeys $ userCertificate
$ nDSPKIUserCertificateInfo $ nDSPKIKeystore $ rADIUSActiveConnections $
rADIUSAttributeLists $ rADIUSConcurrentLimit $ rADIUSConnectionHistory
$ rADIUSDefaultProfile $ rADIUSDialAccessGroup $ rADIUSEnableDialAccess
$ rADIUSPassword $ rADIUSServiceList $ audio $ businessCategory $
carLicense $ departmentNumber $ employeeNumber $ employeeType $
givenName $ homePhone $ homePostalAddress  $ initials $ jpegPhoto $
labeledUri $ mail $ manager $ mobile $ pager $ ldapPhoto $
preferredLanguage $ roomNumber $ secretary $ uid $ userSMIMECertificate
$ x500UniqueIdentifier $ displayName $ userPKCS12) X-NDS_NAME 'User' X
-NDS_NOT_CONTAINER '1' X-NDS_NONREMOVABLE '1' X-NDS_ACL_TEMPLATES
('2#subtree#[Self]#[All Attributes Rights]' '6#entry#[Self]#loginScript'
'1#subtree#[Root Template]#[Entry Rights]' '2#entry#[Public]#messageServer'
'2#entry#[Root Template]#groupMembership'
'6#entry#[Self]#printJobConfiguration' '2#entry#[Root
Template]#networkAddress'))
```

2. In the output noted in the previous step, delete the information marked in bold.
3. Save the revised output as an LDIF file.
4. Add the following information to the newly saved LDIF file:

```
dn: cn=schema

changetype: modify

delete: objectclasses

objectclasses: (2.16.840.1.113730.3.2.2)

-

add:objectclasses
```

Therefore, your LDIF should now be similar to the following:

```
dn: cn=schema

changetype: modify

delete: objectclasses

objectclasses: (2.16.840.1.113730.3.2.2)

-

add:objectclasses
```

```
objectClasses: (2.16.840.1.113730.3.2.2 NAME ’inetOrgPerson’ SUP
organizationalPerson STRUCTURAL MAY (groupMembership $ ndsHomeDirectory
$ loginAllowedTimeMap $ loginDisabled $ loginExpirationTime $
loginGraceLimit $ loginGraceRemaining $ loginIntruderAddress $
loginIntruderAttempts $ loginIntruderResetTime $
loginMaximumSimultaneous $ loginScript $ loginTime $
networkAddressRestriction $ networkAddress $ passwordsUsed $
passwordAllowChange $ passwordExpirationInterval $
passwordExpirationTime $ passwordMinimumLength $ passwordRequired
$ passwordUniqueRequired $ printJobConfiguration $ privateKey $ Profile $
publicKey $ securityEquals $ accountBalance $ allowUnlimitedCredit $
minimumAccountBalance $ messageServer $ Language $ UID $
lockedByIntruder $ serverHolds $ lastLoginTime $ typeCreatorMap $
higherPrivileges $ printerControl $ securityFlags $ profileMembership $
Timezone $ sASServiceDN $ sASSecretStore $ sASSecretStoreKey $
sASSecretStoreData $ sASPKIStoreKeys $ userCertificate $
nDSPKIUserCertificateInfo $ nDSPKIKeystore $ rADIUSActiveConnections $
rADIUSAttributeLists $ rADIUSConcurrentLimit $ rADIUSConnectionHistory $
rADIUSDefaultProfile $ rADIUSDialAccessGroup $ rADIUSEnableDialAccess
$ rADIUSPassword $ rADIUSServiceList $ audio $ businessCategory $
carLicense $ departmentNumber $ employeeNumber $ employeeType $ givenName $
homePhone $ homePostalAddress  $ initials $ jpegPhoto $ labeledUri $ mail
$ manager $ mobile $ pager $ ldapPhoto $ preferredLanguage $ roomNumber
$ secretary $ uid $ userSMIMECertificate $ x500UniqueIdentifier $
displayName $ userPKCS12) X-NDS_NAME ’User’ X-ND S_NOT_CONTAINER ’1’ X
-NDS_NONREMOVABLE ’1’)
```

5 Enter the following command:

`ldapmodify -D cn_of_admin -w password -f LDIF_file_name`

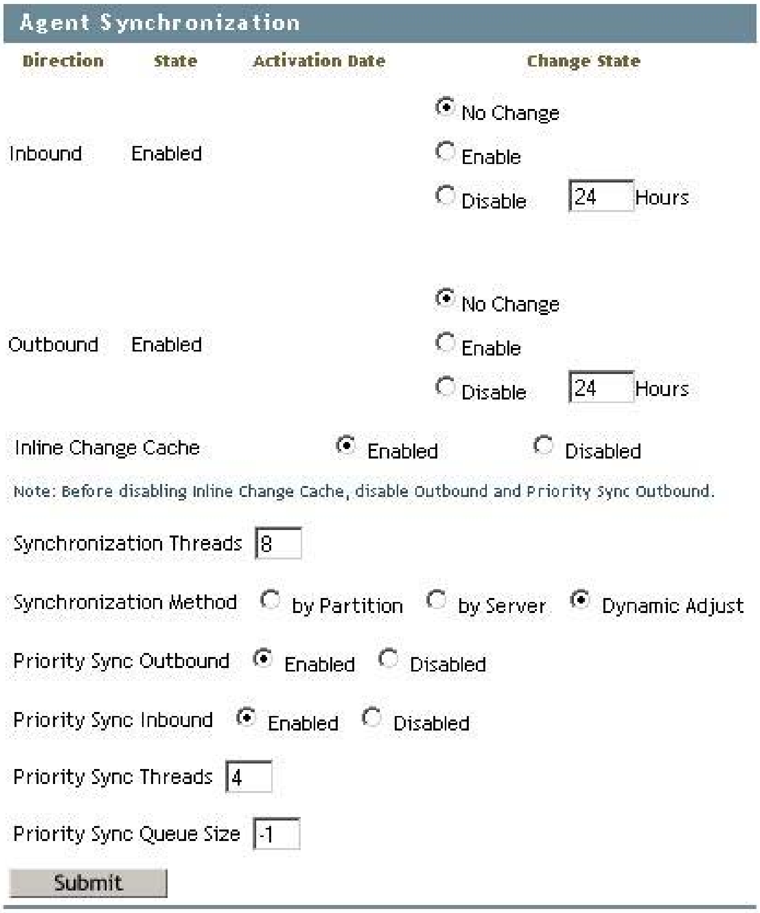
4.4 Replication

When a new server is added to a tree, it is synchronized with its replica. This synchronization can take considerable time, depending on the size of the partition. Synchronization can be avoided by using DIBClone (http://www.novell.com/documentation/edir88/edir88/data/acavuil.html#aky13ak), which allows cloning a server with the same partitions.

By default, in eDirectory, the maximum number of synchronization threads is set to 8. The maximum allowed value is 16. This value can be increased if a large number of partitions are held by the server or when a large number of replica servers exist for the partitions held by the server.

Go to the *Agent Configuration* in the iMonitor, then click *Agent Synchronization* from the left pane to view or change the replication settings.

Figure 4-1 *Agent Synchronization*


Agent Synchronization

Direction	State	Activation Date	Change State
Inbound	Enabled		● No Change ○ Enable ○ Disable 24 Hours
Outbound	Enabled		● No Change ○ Enable ○ Disable 24 Hours

Inline Change Cache ● Enabled ○ Disabled

Note: Before disabling Inline Change Cache, disable Outbound and Priority Sync Outbound.

Synchronization Threads 8

Synchronization Method ○ by Partition ○ by Server ● Dynamic Adjust

Priority Sync Outbound ● Enabled ○ Disabled

Priority Sync Inbound ● Enabled ○ Disabled

Priority Sync Threads 4

Priority Sync Queue Size -1

A dynamically adjusted configuration uses “by server” synchronization if the number of replica servers of all partitions held by the server is more than the number of partitions held by the server. Otherwise it uses the “by partition” synchronization method.

- The number of threads used by the “by server” synchronization method is half the number of servers.
- The number of threads used by the “by partition” synchronization method is same as the number of partitions.

For example, if the synchronization thread count is set to 16 and if there are 18 servers, 9 threads are used for the “by server” synchronization method.

4.5 SSL Overhead

LDAP over SSL adds an additional load on the CPU because of its encryption requirements. A lab performance study shows greater than a 10% performance hit because of encryption overhead.

4.6 64-Bit Versus 32-Bit

Because of the increased size of some of the basic data types in 64-bit eDirectory environments, it needs an increased memory (approximately 20%) over the same instance of 32-bit eDirectory.

eDirectory running as a 32-bit application can only allocate 4 GB memory to the process which limits the amount of usable cache. The maximum size of the FLAIM cache in a 32-bit eDirectory instance is limited to 2.5 GB. This is not a limitation with 64-bit eDirectory, where process memory can grow beyond 4 GB.

The most significant setting that affects eDirectory performance is the cache. Therefore, 64-bit eDirectory performs better for DIBs larger than 1 GB. It is observed through tests that increasing the hit ratio on larger DIB by increasing the cache results in better performance.

When 32-bit eDirectory runs on a 64-bit operating system, a larger filesystem cache can result in a higher throughput even though the FLAIM cache is limited to 2.5 GB. The operating system automatically uses the available RAM for a filesystem cache.

64-bit eDirectory on a Linux x86_64 bit system performs better than the 32-bit eDirectory.

64-bit eDirectory on Solaris sparcv9 systems did not show significant difference in performance in comparison with the same 32-bit instance of eDirectory. With larger DIB sets, it shows increased performance due to a bigger FLAIM cache.

4.7 Import Convert and Export (ICE)

The Novell Import Convert and Export (ICE) utility uses an optimized bulk update protocol called LBURP to upload data into eDirectory. This protocol is significantly faster than uploading data by using a simple `ldapmodify` command. For more information, refer to the Improving Bulkload Performance (http://www.novell.com/documentation/edir88/edir88/data/bqu6wcq.html) in the *Novell eDirectory 8.8 Administration Guide.*

4.8 ldif2dib

For tuning eDirectory performance during offline bulk upload by using the ldif2dib utility, refer to the Tuning ldif2dib (http://www.novell.com/documentation/edir88/edir88/data/b4f3qw0.html) in the *Novell eDirectory 8.8 Administration Guide.*

5 eDirectory Configuration

This section includes the following information:

- Section 5.1, “Configuring the FLAIM Subsystem,” on page 25
- Section 5.2, “Modifying FLAIM Cache Settings,” on page 25

5.1 Configuring the FLAIM Subsystem

In order to address a wide range of deployments and configurations, two mechanisms for controlling the cache memory consumption are provided in the eDirectory. These mechanisms are mutually exclusive.

- Section 5.1.1, “Hard Cache Limit,” on page 25
- Section 5.1.2, “Dynamically Adjusting the Limit,” on page 25

5.1.1 Hard Cache Limit

You can specify a hard memory limit in one of the following ways:

- As a fixed number of bytes.
- As a percentage of physical memory.
- As a percentage of available physical memory.

When a hard limit is specified by using the second or third method, it is always translated to a fixed number of bytes. This means that for the second method, the number of bytes is the percentage of physical memory detected when eDirectory is started. For the third method, the number of bytes is the percentage of available physical memory detected when eDirectory is started.

5.1.2 Dynamically Adjusting the Limit

A dynamic adjustment causes eDirectory to periodically adjust its memory consumption in response to the variable memory consumption by other processes. Although adjusting memory dynamically works well in typical scenarios, this mechanism is not recommended for optimal performance of eDirectory on UNIX platforms because of large differences in memory usage patterns and memory allocators on UNIX platforms.

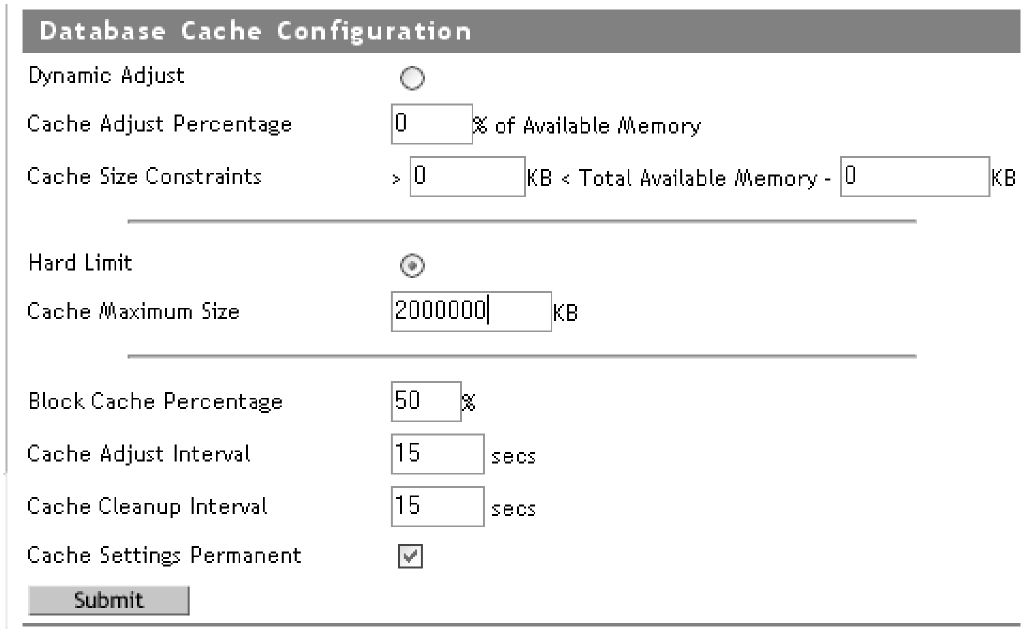
5.2 Modifying FLAIM Cache Settings

- Section 5.2.1, “Modifying FLAIM Cache Settings through iMonitor,” on page 26
- Section 5.2.2, “Modifying FLAIM Cache Settings through _ndsdb.ini,” on page 27

5.2.1 Modifying FLAIM Cache Settings through iMonitor

You can use iMonitor to do the following:

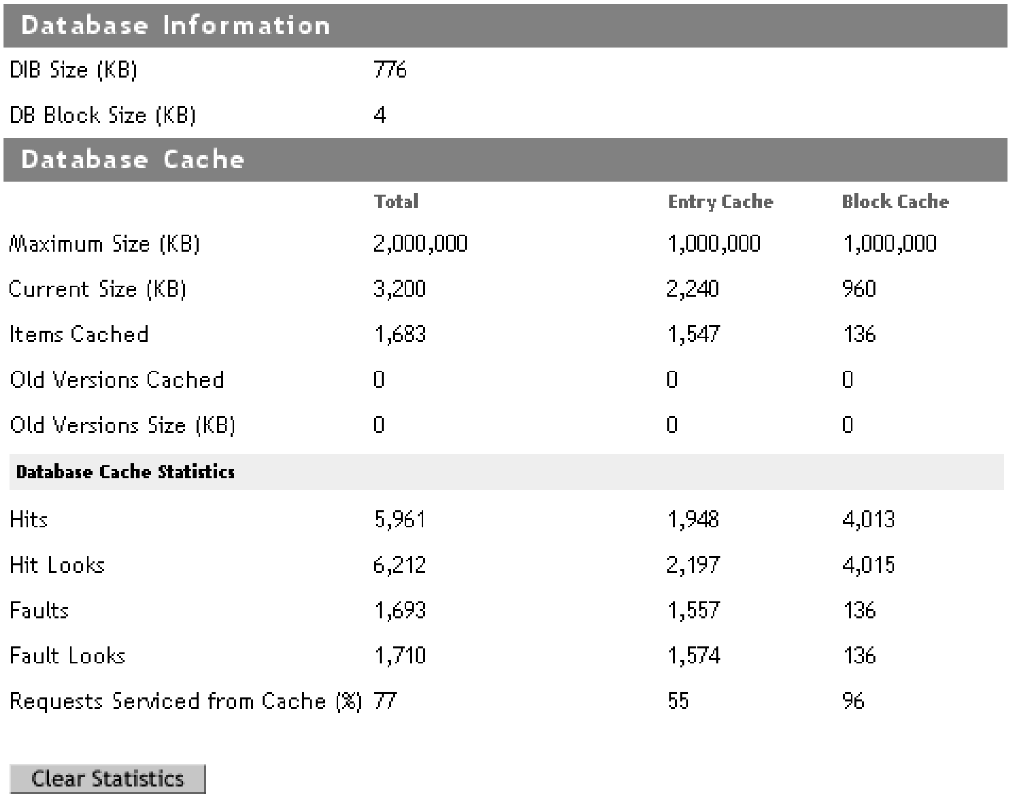
- View or change the cache settings.



Database Cache Configuration

Dynamic Adjust	○
Cache Adjust Percentage	0 % of Available Memory
Cache Size Constraints	> 0 KB < Total Available Memory - 0 KB
Hard Limit	●
Cache Maximum Size	2000000 KB
Block Cache Percentage	50 %
Cache Adjust Interval	15 secs
Cache Cleanup Interval	15 secs
Cache Settings Permanent	☑

- Monitor the cache statistics.



Database Information

DIB Size (KB)	776
DB Block Size (KB)	4

Database Cache

	Total	Entry Cache	Block Cache
Maximum Size (KB)	2,000,000	1,000,000	1,000,000
Current Size (KB)	3,200	2,240	960
Items Cached	1,683	1,547	136
Old Versions Cached	0	0	0
Old Versions Size (KB)	0	0	0
Database Cache Statistics			
Hits	5,961	1,948	4,013
Hit Looks	6,212	2,197	4,015
Faults	1,693	1,557	136
Fault Looks	1,710	1,574	136
Requests Serviced from Cache (%)	77	55	96

Refer to the Database cache under Agent Configuration of iMonitor for the above information.

Database Cache Information	Description
Maximum Size	The maximum size (in KB) that the specified cache is allowed to grow to.
Current Size	The current size (in KB) of the specified cache.
Items Cached	The number of items in the specified cache.
Old Versions Cached	The number of old versions in the specified cache. Old versions of cache items are kept to maintain the consistency of read transactions in the database. In other words, if one thread is in a read transaction and another is in a write transaction, old versions of blocks modified by the writer are maintained on behalf of the reader. This is done so that the reader's results are guaranteed to produce a consistent view during the life of its transaction even though modifications are taking place during that time.
Old Versions Size	The size (in KB) of the old version items cached.
Hits	The number of times an item was successfully accessed from the specified cache.
Hit Looks	The number of items looked at in the cache before an item was successfully accessed from the specified cache. The hit-look-to-hit ratio is a measure of cache lookup efficiency. Normally, the ratio should be close to 1:1.
Faults	The number of times an item was not found in the specified cache and had to be obtained in a lower level cache or from the disk.
Fault Looks	The number of items looked at in the cache before it was determined that the desired item was not in the specified cache. The fault-look-to-fault ratio is a measure of cache lookup efficiency. Normally, the ratio should be close to 1:1.

5.2.2 Modifying FLAIM Cache Settings through _ndsdb.ini

The FLAIM cache settings and other FLAIM configurations can be performed by modifying the `_ndsdb.ini` file that resides in the DIB directory. Restart eDirectory when `_ndsdb.ini` file is changed.

You can set the dynamically adjusting limit or the hard cache limit. The cache options are listed below. Multiple options can be specified, in any order, separated by commas. All are optional.

- **DYN or HARD** - Dynamically adjusting a limit or hard limit.
- **% : percentage** - Percentage of available or physical memory to use.
- **AVAIL or TOTAL** - The percentage specifies available memory or total physical memory. It is applicable only for the hard limit and ignored for the dynamically adjusting limit, because dynamically adjusting limits are always calculated based on the available physical memory. By default, it is AVAIL.
- **MIN: bytes** - Minimum number of bytes.
- **MAX: bytes** - Maximum number of bytes.
- **LEAVE: bytes** - Minimum number of bytes to leave.

For example:

```
cache=HARD,%:75, MIN:200000000
```

```
cache=500000000
```

- **preallocatecache: true/false** - This setting causes eDirectory to preallocate the amount of memory specified by the hard cache limit.
- **rfldirectory -** A different path can be specified for RFL files.
- **cpinterval** - Number of seconds after which FLAIM forces a checkpoint. The default is 3 minutes.
- **maxdirtycache** - Maximum dirty cache bytes.
- **lowdirtycache** - Minimum dirty cache bytes.
- **blockcachepercent** - Percentage of the FLAIM cache used for block cache.
- **cacheadjustinterval** - Interval in seconds for dynamically adjusting the cache.
- **cachecleanupinterval** - Interval in seconds for cleaning up older versions of entries and blocks from the cache.