

IBM Tivoli Directory Server



Performance Tuning and Capacity Planning Guide

Version 6.1

IBM Tivoli Directory Server



Performance Tuning and Capacity Planning Guide

Version 6.1

This edition applies to version 6, release 1, of IBM Tivoli Directory Server and to all subsequent releases and modifications until otherwise indicated in new editions.

© **Copyright International Business Machines Corporation 2007. All rights reserved.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

About this book	v
Intended audience for this book	v
Publications	v
IBM Tivoli Directory Server version 6.1 library	v
Related publications	vi
Accessing terminology online	vi
Accessing publications online	vi
Ordering publications	vii
Accessibility	vii
Tivoli technical training	vii
Support information	vii
Conventions used in this book.	viii
Typeface conventions	viii
Operating system-dependent variables and paths	viii

Chapter 1. IBM Tivoli Directory Server tuning general overview	1
IBM Tivoli Directory Server application components	1
LDAP caches and DB2 buffer pools.	2
Memory allocation between LDAP caches and buffer pools.	3
IBM Tivoli Directory Server tuning overview	3
DB2 tuning overview	3
Performance impact due to multiple password policy	4
Enforcing minimum ulimits	4
Generic LDAP application tips	4

Chapter 2. IBM Tivoli Directory Server tuning	7
LDAP caches	7
LDAP attribute cache	7
LDAP filter cache	11
Entry cache	13
ACL cache.	14
Measuring cache entry sizes.	15
LDAP cache configuration variables	15
Configuring attribute caching	16
Setting other LDAP cache configuration variables	17
Directory size.	19

Chapter 3. Tuning DB2 and LDAP caches	23
DB2 buffer pool tuning	24
Buffer pool sizes.	24
The Performance Tuning Tool (idsperftune).	26
Basic tuning	26
Advanced tuning	28
Optimization and organization (reorgchk and reorg)	29
Optimization	29
Database organization (reorgchk and reorg)	30
Data Row Compression	34

Indexes	35
Other DB2 configuration parameters	36
Database backup and restore considerations	37

Chapter 4. AIX operating system tuning	39
Enabling large files	39
Setting MALLOCTYPE	39
Setting other environment variables	40
Viewing ibmslapd environment variables (AIX operating system only)	40

Chapter 5. Hardware tuning	43
Disk speed improvements	43

Chapter 6. IBM Tivoli Directory Server features	45
Bulkload	45
Effects of using the -k option	45
Replication tuning	47
Number of replication threads	48
Replication context cache size	48
Replication ready size limit	49
Proxy server tuning	50
Monitoring performance	51
ldapsearch with "cn=monitor"	51
ldapsearch with "cn=workers,cn=monitor"	56
ldapsearch with "cn=connections,cn=monitor"	56
ldapsearch with "cn=changelog,cn=monitor"	56
When to configure the LDAP change log	57

Chapter 7. Capacity Planning	59
Disk requirements	60
Bulkload time and space information.	60
Memory requirements	65
CPU requirements	65
CPU scaling comparison for throughput (searches and updates)	65
Splitting the database across multiple disks.	67
Simultaneous multithreading	68
SMT on AIX FAQs	69

Appendix A. Workload description	71
---	-----------

Appendix B. Modifying TCP/IP settings	73
--	-----------

Appendix C. Platform configurations	75
--	-----------

Appendix D. Notices	77
Trademarks	78

Index	81
------------------------	-----------

About this book

IBM® Tivoli® Directory Server is the IBM implementation of Lightweight Directory Access Protocol for supported Windows®, AIX®, Linux® (xSeries®, zSeries®, pSeries®, and iSeries™), Solaris, and Hewlett-Packard UNIX® (HP-UX) operating systems.

IBM Tivoli Directory Server version 6.1 Performance Tuning and Capacity Planning Guide contains information about tuning the directory server for better performance.

Intended audience for this book

This book is for system administrators, network administrators, information technology architects, and application developers.

Readers need to know how to use the operating system on which IBM Tivoli Directory Server will be installed.

Publications

This section lists publications in the IBM Tivoli Directory Server version 6.1 library and related documents. The section also describes how to access Tivoli publications online and how to order Tivoli publications.

IBM Tivoli Directory Server version 6.1 library

The following documents are available in the IBM Tivoli Directory Server version 6.1 library:

- *IBM Tivoli Directory Server Version 6.1 What's New for This Release*, SC23-6539-00
Provides information about the new features in the IBM Tivoli Directory Server Version 6.1 release.
- *IBM Tivoli Directory Server Version 6.1 Quick Start Guide*, GI11-8172-00
Provides help for getting started with IBM Tivoli Directory Server 6.1. Includes a short product description and architecture diagram, as well as a pointer to the product Information Center and installation instructions.
- *IBM Tivoli Directory Server Version 6.1 System Requirements*, SC23-7835-00
Contains the minimum hardware and software requirements for installing and using IBM Tivoli Directory Server 6.1 and its related software. Also lists the supported versions of corequisite products such as DB2® and GSKit.
- *IBM Tivoli Directory Server Version 6.1 Installation and Configuration Guide*, GC32-1560-00
Contains complete information for installing, configuring, and uninstalling IBM Tivoli Directory Server. Includes information about upgrading from a previous version of IBM Tivoli Directory Server.
- *IBM Tivoli Directory Server Version 6.1 Administration Guide*, GC32-1564-00
Contains instructions for performing administrator tasks through the Web Administration Tool and the command line.
- *IBM Tivoli Directory Server Version 6.1 Command Reference*, SC23-7834-00

Describes the syntax and usage of the command-line utilities included with IBM Tivoli Directory Server.

- *IBM Tivoli Directory Server Version 6.1 Server Plug-ins Reference*, GC32-1565-00
Contains information about writing server plug-ins.
- *IBM Tivoli Directory Server Version 6.1 Programming Reference*, SC23-7836-00
Contains information about writing Lightweight Directory Access Protocol (LDAP) client applications in C and Java™.
- *IBM Tivoli Directory Server Version 6.1 Performance Tuning and Capacity Planning Guide*, SC23-6540-00
Contains information about tuning the directory server for better performance. Describes disk requirements and other hardware needs for directories of different sizes and with various read and write rates. Describes known working scenarios for each of these levels of directory and the disk and memory used; also suggests rough rules of thumb.
- *IBM Tivoli Directory Server Version 6.1 Problem Determination Guide*, GC32-1568-00
Contains information about possible problems and corrective actions that can be tried before contacting IBM Software Support.
- *IBM Tivoli Directory Server Version 6.1 Messages Guide*, GC32-1567-00
Contains a list of all informational, warning, and error messages associated with IBM Tivoli Directory Server 6.1.
- *IBM Tivoli Directory Server Version 6.1 White Pages*, SC23-7837-00
Describes the Directory White Pages application, which is provided with IBM Tivoli Directory Server 6.1. Contains information about installing, configuring, and using the application for both administrators and users.

Related publications

The following documents also provide useful information:

- *Java Naming and Directory Interface™ 1.2.1 Specification* on the Sun Microsystems Web site at <http://java.sun.com/products/jndi/1.2/javadoc/index.html>.
IBM Tivoli Directory Server Version 6.1 uses the Java Naming and Directory Interface (JNDI) client from Sun Microsystems. See this document for information about the JNDI client.

Accessing terminology online

The *Tivoli Software Glossary* includes definitions for many of the technical terms related to Tivoli software. The *Tivoli Software Glossary* is available at the following Tivoli software library Web site:

<http://publib.boulder.ibm.com/tividd/glossary/tivoliglossarymst.htm>

The IBM Terminology Web site consolidates the terminology from IBM product libraries in one convenient location. You can access the Terminology Web site at the following Web address:

<http://www.ibm.com/software/globalization/terminology>

Accessing publications online

IBM posts publications for this and all other Tivoli products, as they become available and whenever they are updated, to the Tivoli Information Center Web site at <http://publib.boulder.ibm.com/tividd/td/link/tdprodlist.html>.

In the Tivoli Information Center window, click **Tivoli product manuals**. Click the letter that matches the first letter of your product name to access your product library. For example, click **M** to access the IBM Tivoli Monitoring library or click **O** to access the IBM Tivoli OMEGAMON® library.

Note: If you print PDF documents on other than letter-sized paper, set the option in the **File → Print** window that allows Adobe® Reader to print letter-sized pages on your local paper.

Ordering publications

You can order many Tivoli publications online at <http://www.elink.ibm.com/public/applications/publications/cgibin/pbi.cgi>.

You can also order by telephone by calling one of these numbers:

- In the United States: 800-879-2755
- In Canada: 800-426-4968

In other countries, contact your software account representative to order Tivoli publications. To locate the telephone number of your local representative, perform the following steps:

1. Go to <http://www.elink.ibm.com/public/applications/publications/cgibin/pbi.cgi>.
2. Select your country from the list and click **Go**.
3. Click **About this site** in the main panel to see an information page that includes the telephone number of your local representative.

Accessibility

Accessibility features help users with a physical disability, such as restricted mobility or limited vision, to use software products successfully. With this product, you can use assistive technologies to hear and navigate the interface. You can also use the keyboard instead of the mouse to operate all features of the graphical user interface.

For additional information, see the *IBM Tivoli Directory Server Version 6.1 Installation and Configuration Guide*.

Tivoli technical training

For Tivoli technical training information, refer to the following IBM Tivoli Education Web site at <http://www.ibm.com/software/tivoli/education>.

Support information

If you have a problem with your IBM software, you want to resolve it quickly. IBM provides the following ways for you to obtain the support you need:

- IBM Support Assistant: You can search across a large collection of known problems and workarounds, Technotes, and other information at <http://www.ibm.com/software/support/isa>.
- Obtaining fixes: You can locate the latest fixes that are already available for your product.

- **Contacting IBM Software Support:** If you still cannot solve your problem, and you need to work with someone from IBM, you can use a variety of ways to contact IBM Software Support.

For more information about resolving problems, see the *IBM Tivoli Directory Server Version 6.1 Problem Determination Guide*.

Conventions used in this book

This book uses several conventions for special terms and actions, operating system-dependent commands and paths, and margin graphics.

Typeface conventions

This book uses the following typeface conventions:

Bold

- Lowercase commands and mixed case commands that are otherwise difficult to distinguish from surrounding text
- Interface controls (check boxes, push buttons, radio buttons, spin buttons, fields, folders, icons, list boxes, items inside list boxes, multicolumn lists, containers, menu choices, menu names, tabs, property sheets), labels (such as **Tip:**, and **Operating system considerations:**)
- Keywords and parameters in text

Italic

- Citations (examples: titles of books, diskettes, and CDs)
- Words defined in text (example: a nonswitched line is called a *point-to-point line*)
- Emphasis of words and letters (words as words example: "Use the word *that* to introduce a restrictive clause."; letters as letters example: "The LUN address must start with the letter *L*.")
- New terms in text (except in a definition list): a *view* is a frame in a workspace that contains data.
- Variables and values you must provide: ... where *myname* represents....

Monospace

- Examples and code examples
- File names, programming keywords, and other elements that are difficult to distinguish from surrounding text
- Message text and prompts addressed to the user
- Text that the user must type
- Values for arguments or command options

Operating system-dependent variables and paths

This book uses the UNIX convention for specifying environment variables and for directory notation.

When using the Windows command line, replace *\$variable* with *% variable%* for environment variables and replace each forward slash (/) with a backslash (\) in directory paths. The names of environment variables are not always the same in the Windows and UNIX environments. For example, *%TEMP%* in Windows environments is equivalent to *\$TMPDIR* in UNIX environments.

Note: If you are using the bash shell on a Windows system, you can use the UNIX conventions.

Chapter 1. IBM Tivoli Directory Server tuning general overview

This guide provides tuning information for IBM Tivoli Directory Server and the related IBM Database 2™ (DB2) database. IBM Tivoli Directory Server is a Lightweight Directory Access Protocol (LDAP) directory that provides a layer on top of DB2, allowing users to efficiently organize, manipulate, and retrieve data stored in the DB2 database. Tuning for optimal performance is primarily a matter of adjusting the relationships between the LDAP server and DB2 according to the nature of your workload.

Because each workload is different, instead of providing exact values for tuning settings, guidelines are provided, where appropriate, for how to determine the best settings for your system.

Attention: Measurements in this guide were captured in a lab environment. The workload driving this test included a mixture of searches and binds, including wildcard searches which return multiple entries. Your results might differ from the lab results shown in this guide.

IBM Tivoli Directory Server application components

The following figure illustrates how IBM Tivoli Directory Server components interact with each other. Tuning these components can result in improved performance.

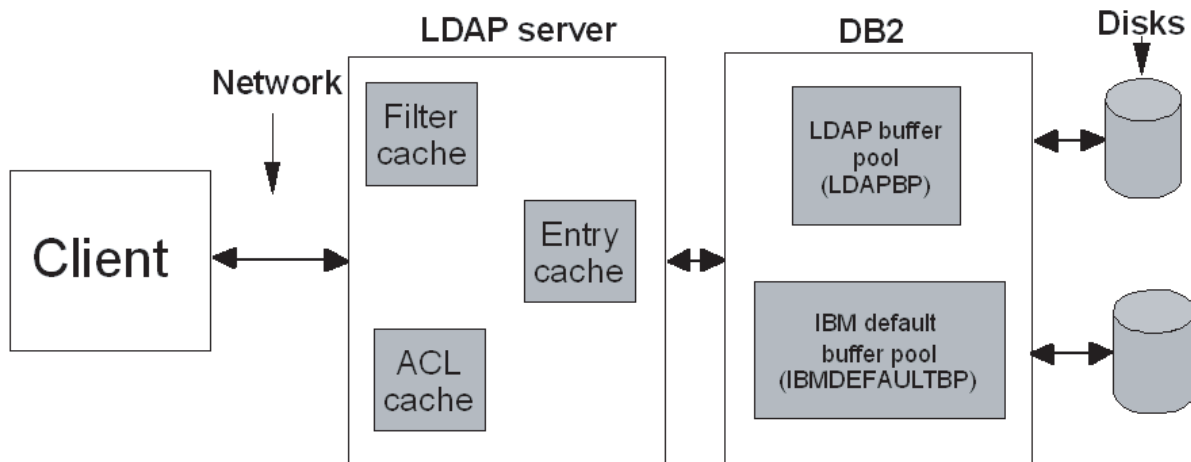


Figure 1. IBM Tivoli Directory Server

The arrows in Figure 1 represent the path of a query issued from a client computer. The query follows a path from the IBM Tivoli Directory Server client to the LDAP server, to DB2, to the physical disks in search of entries that match the query's search filter settings. The shorter the path to matching entries, the better overall performance you can expect from your system.

For example, if a query locates all the matching entries in the LDAP server, access to DB2 and the disks is not necessary. If the matching entries are not found in the

LDAP server, the query continues on to DB2 and, if necessary, to the physical disks as a last resort. Because of the time and resources it takes to retrieve data from disk, it is better from a performance standpoint to allocate a significant amount of memory to the LDAP server caches and DB2 buffer pools.

LDAP caches and DB2 buffer pools

Caches and buffer pools store previously retrieved data and can significantly improve performance by reducing disk access. When requested data is found within a cache or buffer pool, it is called a cache hit. A cache miss occurs when requested data is not located in a cache or buffer pool.

Because the type of information in each cache and buffer pool is different, it is useful to understand how and when each cache is accessed.

LDAP caches

Search operations attempt to use one or more caches when resolving the search filter as well as returning the individual matching entries. Most base-scoped searches can be resolved directly in memory by retrieving the base entry from the entry cache or the database bufferpool and performing the comparison of the entry with the filter.

If a base-scoped search cannot be resolved directly in memory or the search is not base-scoped, an attempt is made to use the attribute cache to resolve the filter in memory or to use the filter cache to retrieve the results of a previously run search operation. If LDAP caches cannot be used to resolve the filter, the filter will be resolved using DB2. When the individual entries are returned to the client, they are retrieved from memory using the entry cache, if possible. If the individual entries are not found in the entry cache, they are retrieved from DB2

The four LDAP caches are:

- LDAP attribute cache
- LDAP filter cache
- Entry cache
- ACL cache

For more information on these caches, see “LDAP caches” on page 7.

DB2 buffer pools

There are two DB2 buffer pools:

LDAPBP

LDAPBP contains cached entry data (`ldap_entry`) and all of the associated indexes. LDAPBP is similar to the entry cache, except that LDAPBP uses different algorithms in determining which entries are cached. It is possible that an entry that is not cached in the entry cache is located in LDAPBP. If the requested data is not found in the entry cache or LDAPBP, the query must access the physical disks.

See “LDAPBP buffer pool size” on page 25 for more information.

IBMDEFAULTBP

DB2 system information, including system tables and other LDAP information, is cached in the IBMDEFAULTBP. You might need to adjust the IBMDEFAULTBP cache settings for better performance. See “IBMDEFAULTBP buffer pool size” on page 25 for more information.

Memory allocation between LDAP caches and buffer pools

The LDAP caches are generally more efficient as a means of caching LDAP searches; however, parts of the LDAP cache get invalidated on updates and must be reloaded before performance benefits return. Some experimentation between the two caching schemes is required to find the best memory allocation for your workload.

IBM Tivoli Directory Server tuning overview

Tuning the LDAP server can significantly improve performance by storing useful data in the caches. It is important to remember, however, that tuning the LDAP server alone is insufficient. Some tuning of DB2 is also required for optimal performance.

The most significant performance tuning related to the IBM Tivoli Directory Server involves the LDAP caches. LDAP caches are fast storage buffers in memory used to store LDAP information such as queries, answers, and user authentication for future use. While LDAP caches are useful mostly for applications that frequently retrieve repeated cached information, they can greatly improve performance by avoiding calls to the database. See “LDAP caches” on page 7 for information about how to tune the LDAP caches.

DB2 tuning overview

DB2 serves as the data storage component of the IBM Tivoli Directory Server. Tuning DB2 results in overall improved performance.

This guide contains several recommendations for tuning DB2, but the most commonly tuned items are:

- **DB2 buffer pools** – Buffer pools are DB2 data caches. Each buffer pool is a data cache between the applications and the physical database files. Adjusting the size of the DB2 buffer pools can result in improved performance. See “DB2 buffer pool tuning” on page 24 for information about buffer pool tuning.
- **Optimization and organization** – After initially loading a directory, or after a number of updates have been performed, it is very important to update database statistics and organization for DB2 to perform optimally. See “Optimization and organization (reorgchk and reorg)” on page 29 for more information.
- **Indexes** – Indexes can make locating data on disk very fast, providing a significant boost to performance. For information about how to create indexes, see “Indexes” on page 35.

Attention: You should place the DB2 log on a physical disk drive separate from the data. For improved data-integrity, have the DB2 log and the data on separate drives. Use the following command to set the path to the DB2 log file directory:

```
DB2 UPDATE DATABASE CONFIGURATION FOR database_alias USING NEWLOGPATH path
```

Be sure the database instance owner has write access to the specified path or the command fails. For more information on using DB2 commands, see Chapter 3, “Tuning DB2 and LDAP caches,” on page 23.

Performance impact due to multiple password policy

To evaluate a user's effective password policy, the directory server takes into consideration all the password policies associated with a user. This means that the directory server evaluates the individual, group, and global password policies to determine a user's effective password policy.

However, in order to authenticate a user during bind, the bind performance could be degraded. This is because during bind time a user's group password policy must be determined and to determine a user's group policy, group membership must be resolved.

To improve performance, during server startup time, a flag is set if password policy entries (except for the global password policy) exist in the DIT and references to group and individual password policy entries are defined. If the flag is set, then a user's effective password policy is evaluated by searching the user's individual and group policies. If the flag is not set, then there is no attempt to search for a user's individual or group policy and the global policy, if it is turned on, is used as the user's effective policy.

After server startup, this flag is set when attribute `ibm-pwdGroupPolicyDN` or `ibm-pwdIndividualPolicy Dn` is added to the DIT. After the flag is set, it will not be reset by any delete or replace operations. It can be reset only when the server is restarted and the flag is reevaluated.

Enforcing minimum ulimits

The directory server tries to enforce minimum ulimit option values that are considered important for the smooth running of the server. To accomplish this, the directory server first checks if the ulimit option values for the current process are greater than or equal to the prescribed ulimit option values specified in the configuration file. In case the ulimit option values for the current process are lesser than the prescribed values, the server attempts to set the ulimit option values of the current process to the prescribed values.

Note: The directory server tries to enforce minimum ulimit option values that are considered important for the smooth running of the server. However, an administrator can modify the minimum ulimit option values using the web administration tool or through command line.

Generic LDAP application tips

The following are some tips that can help improve performance:

- Perform searches on indexed attributes only. See "Indexes" on page 35 for instructions for defining and verifying indexes for IBM Tivoli Directory Server.
- Open a connection only once and reuse it for many operations if possible.
- Minimize the number of searches by retrieving multiple attribute values at one time.
- Retrieve only the attributes you need. Do not use ALL by default. For example, when you search for the groups a user belongs to, ask for only the Distinguished Names (DNs), and not the entire group. Do not request the member or uniquemember attributes if possible.
- Minimize and batch updates (**add**, **modify**, **modrdn**, **delete**) when possible.

- Use base-scoped searches whenever possible rather than one-level or subtree searches. A base-scoped search is a search done using the ldapsearch utility, where the scope of the search is defined as a base object.
- Avoid using wildcard searches where the wildcard is in any position other than the leading character in a term, or a trailing character. Use wildcard searches that are similar to the following (leading character):

sn=*term

or the following (trailing character):

sn=term*

Note: A filter such as sn=*term* is less efficient than the examples given.

- When using nested groups, keep the depth of nesting to 50 groups or less. Greater nesting depths can result in greater processing times when performing add or delete operations that involve updates to the nested group hierarchy.
- Set server search limits to prevent accidental long-running searches.
- Use the ldap_modify interface to add members to or delete members from a group. Do not do a search to retrieve all members, edit the returned list, then send the updated list as a modify-replace operation. This modify-replace scenario will not perform well with large groups.
- For the Proxy server, do not set the value in the **Connection pool size** field to be less than 5.

Chapter 2. IBM Tivoli Directory Server tuning

This chapter discusses the following performance tuning tasks for the IBM Tivoli Directory Server:

- Tuning LDAP caches
- Determining how directory size affects performance

LDAP caches

LDAP caches are fast storage buffers in memory used to store LDAP information such as queries, answers, and user authentication for future use. Tuning the LDAP caches is crucial to improving performance.

An LDAP search that accesses the LDAP cache can be faster than one that requires a connection to DB2, even if the information is cached in DB2. For this reason, tuning LDAP caches can improve performance by avoiding calls to the database. The LDAP caches are especially useful for applications that frequently retrieve repeated cached information. See Figure 1 on page 1 for an illustration of the LDAP caches.

The following sections discuss each of the LDAP caches and demonstrate how to determine and set the best cache settings for your system. Keep in mind that every workload is different, and some experimentation will likely be required in order to find the best settings for your workload.

Note: Cache sizes for the filter cache, ACL cache, and entry cache are measured in numbers of entries.

LDAP attribute cache

The attribute cache stores configured attributes and their values in memory. When a one-level or sub-tree search is performed, or a base-scoped search is performed that cannot be resolved directly in memory, the attribute cache manager resolves the search operation in memory if all attributes used in the filter are cached and the filter is a type supported by the attribute cache manager. Resolving filters in memory leads to improved search performance over resolving filters using DB2.

There are two things that can happen when a query arrives at the attribute cache:

- **All attributes used in the search filter are cached and the filter is of a type that can be resolved by the attribute cache manager.** If this is the case, the list of matching entry IDs is resolved in memory using the attribute cache manager. This list of matching IDs is then sent to the entry cache. For this reason, the attribute cache is most efficient when used in combination with the entry cache. The attribute cache manager can resolve simple filters of the following types:

- exact match filters
- presence filters

The attribute cache manager can also resolve complex filters that are conjunctive or disjunctive. Additionally, the subfilters within complex filters must be exact match, presence, conjunctive, or disjunctive.

- exact match filters
- presence filters

- conjunctive filters
- disjunctive filters

Filters containing attributes with language tags are not resolved by the attribute cache manager.

For example, if the attributes `objectclass`, `uid`, and `cn` are all cached, the following filters can be resolved in memory within the attribute cache manager:

- `(cn=Karl)`
- `(cn=*)`
- `(&(objectclass=eperson)(cn=Karl))`
- `(&(objectclass=eperson)(cn=*)(uid=1234567))`
- `(&(&(objectclass=eperson)(cn=*)) (uid=1234567))`
- `(&(uid=1234567) (&(objectclass=eperson)(cn=*))`

- **Some or all of the attributes used in the search filter are not cached or the filter is of a type that cannot be resolved by the attribute cache manager.** If this is the case, the query is sent to the filter cache for further processing.

Note: If there are no attributes in the attribute cache, the attribute cache manager determines this quickly, and the query is sent to the filter cache.

For example, if the attributes `objectclass`, `uid`, and `cn` are the only cached attributes, the following filters will not be able to be resolved in memory by the attribute cache manager:

- `(sn=Smith)`
- `(cn=K*)`
- `(|(objectclass=eperson)(cn~=Karl))`
- `(&(objectclass=eperson)(cn=K*)(uid=1234567))`
- `(&(&(objectclass=eperson)(cn<=Karl)) (uid=1234567))`
- `(&(uid=1234567) (&(objectclass=eperson)(sn=*))`

Note: Choosing to cache `member`, `uniquemember`, or `ibm-membergroup` can lead to slower performance of `delete` and `modrdn` operations. If the entry being deleted or renamed is a member of many groups, or large groups, then the attribute caches need to be updated to reflect this change for every group in which the entry was a member.

Determining which attributes to cache

To determine which attributes to cache, experiment with adding some or all of the attributes listed in the `cached_attribute_candidate_hit` attribute to the attribute cache. Then run your workload and measure the differences in operations per second. For information about the `cached_attribute_candidate_hit` attribute, see "ldapsearch with "cn=monitor"" on page 51.

Note: Choosing to cache `member`, `uniquemember`, or `ibm-membergroup` can lead to slower performance of `delete` and `modrdn` operations. If the entry being deleted or renamed is a member of many groups or large groups, the attribute caches are updated to reflect this change for every group in which the entry was a member. This additional processing can lead to slower performance of these types of operations.

Examples: Information about attributes that are cached, their individual sizes in kilobytes, and their hit counts can be retrieved during `cn=monitor` searches. Also, up to ten attributes that are most often used in search filters that can be processed by the attribute cache manager, but are not yet cached, can be retrieved during

cn=monitor searches. Use a combination of the output from cn=monitor searches and knowledge of the types of searches your applications use to determine which attributes to cache.

Example 1: The following results are for a cn=monitor search for a server that had no attributes configured for attribute caching:

```
ldapsearch -h ldaphost -s base -b cn=monitor objectclass=*
  cached_attribute_total_size
cached_attribute_configured_size cached_attribute_hit cached_attribute_size
cached_attribute_candidate_hit
cn=monitor
cached_attribute_total_size=0
cached_attribute_configured_size=1200
cached_attribute_candidate_hit=mail:50000
cached_attribute_candidate_hit=uid:45000
cached_attribute_candidate_hit=givenname:500
cached_attribute_candidate_hit=sn:200
```

If this cn=monitor search produced these results, you can assume that the attributes to cache must be uid and mail. Even though givenname and sn were used in search filters that have been resolved by the attribute cache manager had those attributes been cached, their hit counts are very low in comparison to the attributes uid and mail, and using memory to store givenname and sn is not realistic.

After the attributes uid and mail are cached and the application or performance test is rerun, the cn=monitor search should be performed again to determine if there is enough memory configured to cache both attributes. If there is not enough memory, then additional memory must be configured, or the least-used attribute must be removed from the list of attributes to cache.

Example 2: In this example, givenname and sn are already cached. The hit count for objectclass is very high. Also, the hit rates for uid and mail are very high:

```
ldapsearch -h ldaphost -s base -b cn=monitor objectclass=*
  cached_attribute_total_size
cached_attribute_configured_size cached_attribute_hit cached_attribute_size
cached_attribute_candidate_hit
cn=monitor
cached_attribute_total_size=1000
cached_attribute_configured_size=1200
cached_attribute_hit=givenname:500
cached_attribute_size=givenname:300
cached_attribute_hit=sn:200
cached_attribute_size=sn:400
cached_attribute_candidate_hit=objectclass:110000
cached_attribute_candidate_hit=mail:90000
cached_attribute_candidate_hit=uid:85000
cached_attribute_candidate_hit=workloc:25000
```

Note: cached_attribute_total_size is the amount of memory used by the directory attribute cache, in kilobytes. This number includes additional memory used to manage the cache that is not charged to the individual attribute caches. Consequently, this total is larger than the sum of the memory used by all the individual attribute caches.

As in the previous example, givenname and sn are not good choices for caching because of their relatively low hit count, in comparison to the other attributes listed. You can assume that objectclass is the best choice and that uid and mail are also excellent choices. If attribute caching is reconfigured to cache objectclass, uid and mail, you might discover after caching is complete and after rerunning

your performance tests under the same conditions, that your performance isn't what you expect. Also, the `cn=monitor` search yields the following unexpected results which show that only `objectclass` is cached, and its hit count is much lower than when it was a candidate:

```
ldapsearch -h ldaphost -s base -b cn=monitor objectclass=*
  cached_attribute_total_size
  cached_attribute_configured_size  cached_attribute_hit  cached_attribute_size
  cached_attribute_candidate_hit
  cn=monitor
  cached_attribute_total_size=1000
  cached_attribute_configured_size=1200
  cached_attribute_hit=objectclass:10000
  cached_attribute_size=objectclass:750
  cached_attribute_candidate_hit=mail:90000
  cached_attribute_candidate_hit=uid:85000
  cached_attribute_candidate_hit=workloc:25000
  cached_attribute_candidate_hit=givenname:300
  cached_attribute_candidate_hit=sn:200
```

Two things occurred to cause these results:

1. The `objectclass` attribute table was large in comparison to the other attribute tables. Even though `objectclass`, `uid` and `mail` were all configured to be cached, `objectclass` was the only attribute that fit within the maximum memory configured for attribute caching.
2. Further analysis of the search filters used by your application reveals that `objectclass` was not used in search filters by itself very often. The attribute cache manager could not resolve many filters because not all attributes in the filter were cached. A combination of the `cn=monitor` output and analysis of the filters used by your application is necessary to determine which attributes to cache. The following search filters were used in this example:

<code>(objectclass=*)</code>	10000 hits
<code>(givenname=*)</code>	300 hits
<code>(sn=*)</code>	200 hits
<code>(mail=*)</code>	50000 hits
<code>(uid=*)</code>	45000 hits
<code>(workloc=*)</code>	5000 hits
<code>(&(objectclass=person)(mail=*))</code>	40000 hits
<code>(&(objectclass=person)(uid=*))</code>	40000 hits
<code>(&(objectclass=person)(workloc=*))</code>	20000 hits

You can see from the above filter analysis that `objectclass`, when used alone, had only 10000 hits. Therefore, if the only attribute cached is `objectclass`, the attribute cache manager can only resolve 10000 out of the 210500 total search filters. If the server is reconfigured to have enough memory to hold both the `objectclass` and `mail` attributes, 100000 of the search filters can be resolved in the attribute cache manager. If `objectclass`, `uid` and `mail` were all configured and enough memory was available, 185000 of the search filters can be resolved by the attribute cache manager. However, if memory is constrained and only one attribute can be cached, the best choice is `mail` with 50000 hits. If both `uid` and `mail` can be cached, 95000 filters can be resolved in the attribute cache manager, which is almost as many hits as caching `objectclass` and `mail` instead.

Because caching `uid` and `mail` likely consumes less memory than caching `objectclass` and `mail`, caching `uid` and `mail` instead of `objectclass` and `mail` might be a better choice if not enough memory is available on your server. Therefore, it is necessary to understand and consider the types of search filters used by your application in order to determine the appropriate attributes to cache as well as to consider the amount of memory that you want the attribute cache to be able to use.

LDAP filter cache

The filter cache contains cached entry IDs that match a search filter that was previously resolved in DB2. When the client issues a query for some data and that query is not a base-scoped search that can be resolved in memory nor is it a filter that can be resolved in memory by the attribute cache manager, the query goes to the filter cache. There are two things that can happen when a query arrives at the filter cache:

- **The IDs that match the filter settings used in the query are located in the filter cache.** If this is the case, the list of the matching entry IDs is sent to the entry cache.
- **The matching entry IDs are not cached in the filter cache.** In this case, the query must access DB2 in search of the desired data.

Filter cache size

To determine how big your filter cache should be, run your workload with the filter cache set to different values and measure the differences in operations per second. For example, Figure 2 shows varying operations per second based on different filter cache sizes for one installation:

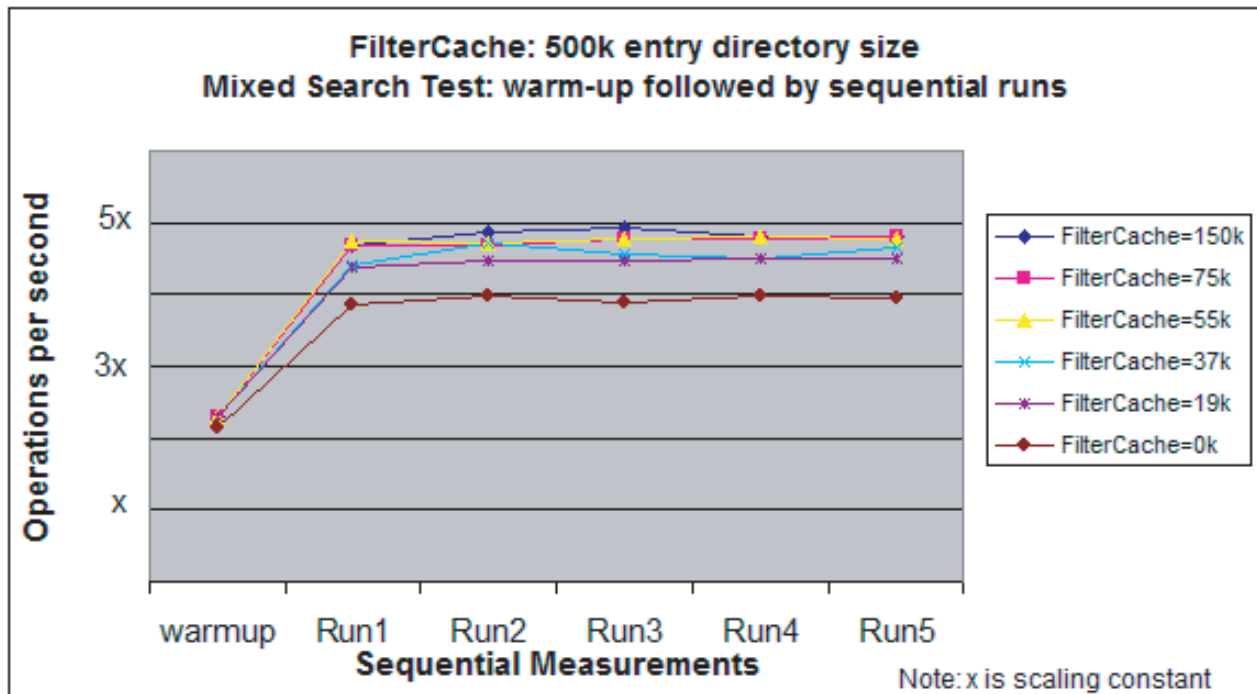


Figure 2. Varying the size of the filter cache

For this workload it appears that a filter cache large enough to hold 55,000 entries results in the best performance. There is no benefit in making the filter cache any larger than this. See “LDAP cache configuration variables” on page 15 to set the filter cache size.

Filter cache size with updates

Figure 3 on page 12 shows that, for the test installation, there is no performance benefit in allocating any memory to the filter cache if even a small fraction of the operations in the workload are updates.

If this proves to be the case for your workload, the only way to retain the performance advantage of a filter cache when updates are involved is to batch your updates. This allows long intervals during which there are only searches. If you cannot batch updates, specify a filter cache size of zero and allocate more memory to other caches and buffer pools. See “LDAP cache configuration variables” on page 15 for instructions on how to set configuration variables such as filter cache size.

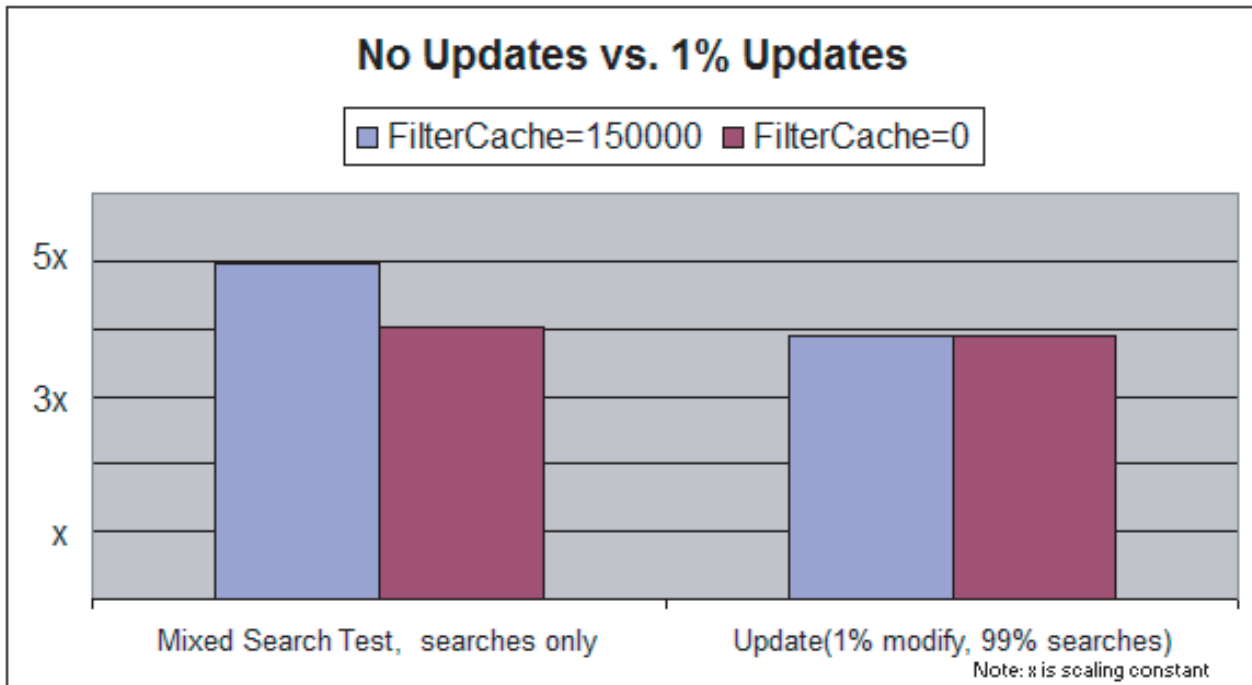


Figure 3. Effect of updates on the performance of the filter cache

Filter cache bypass limits

The filter cache bypass limit configuration variable limits the size of entries that can be added to the filter cache. For example, if the bypass limit variable is set to 1,000, search filters that match more than 1,000 entries are not added to the filter cache. This prevents large, uncommon searches from overwriting useful cache entries. To determine the best filter cache bypass limit for your workload, repeatedly run your workload with the filter cache bypass limits set to different values and measure the operations per second.

For example, Figure 4 on page 13 shows operations per second based on varying cache bypass limit sizes:

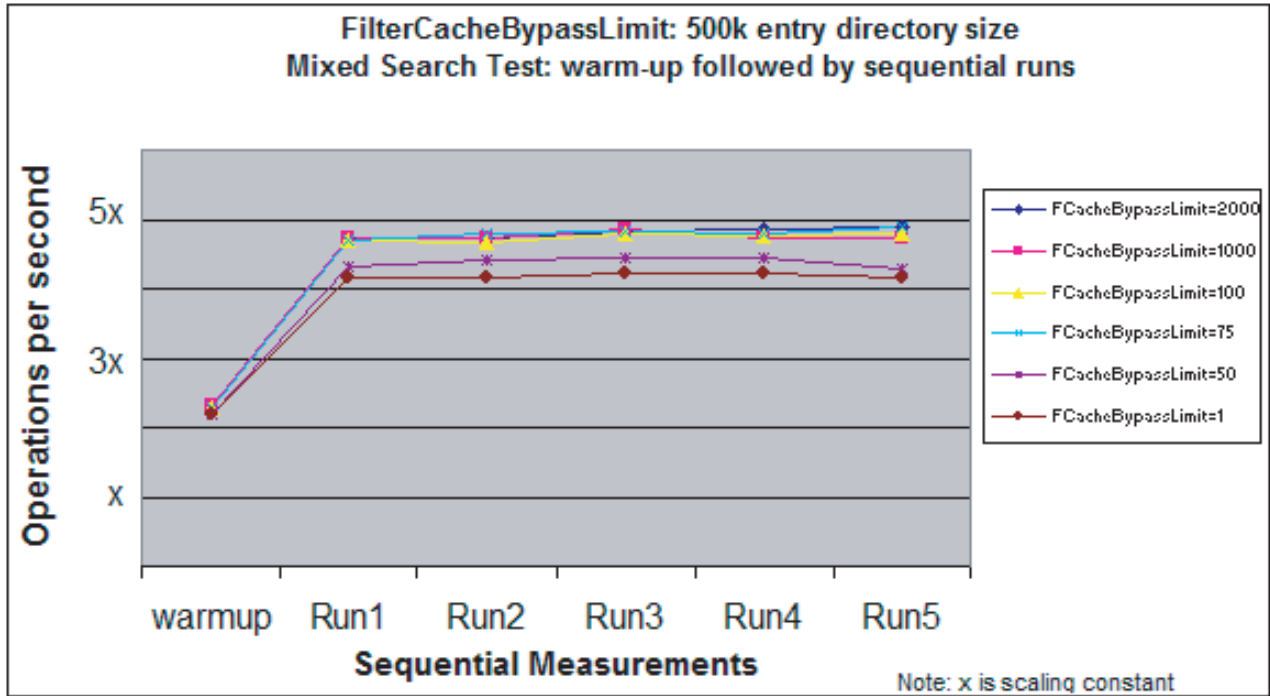


Figure 4. Varying the filter cache bypass limit

For the workload in Figure 4, setting the limit too low downgrades performance by preventing valuable filters from being cached. Setting the filter bypass limit to approximately 100 appears to be the best size for this workload. Setting it any larger benefits performance only slightly.

See “LDAP cache configuration variables” on page 15 to set the filter cache bypass limit.

Entry cache

The entry cache contains cached entry data. Entry IDs are sent to the entry cache. If the entries that match the entry IDs are in the entry cache, then the results are returned to the client. If the entry cache does not contain the entries that correspond to the entry IDs, the query goes to DB2 in search of the matching entries.

Entry cache size

To determine how big your entry cache should be, run your workload with the entry cache set to different sizes and measure the differences in operations per second. For example, Figure 5 on page 14 shows varying operations per second based on different entry cache sizes:

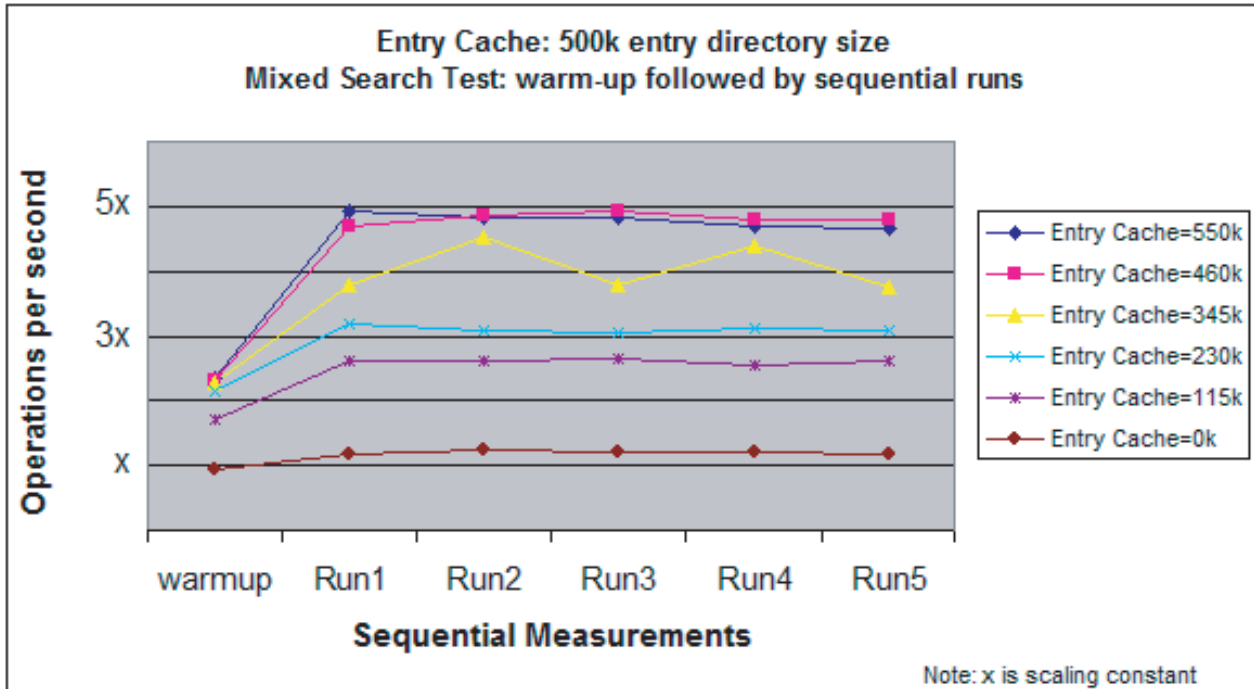


Figure 5. Varying the size of the entry cache

From the results in Figure 5, it appears that an entry cache large enough to hold 460,000 entries results in the best performance. There is no benefit to making the entry cache any larger than this. Setting the entry cache at 460,000 results in 4 times as many operations per second than if entry cache was set to zero. To find the best cache size for your workload, you must run your workload with different cache sizes. See “LDAP cache configuration variables” on page 15 to set the filter cache size.

Note: The test with Entry Cache size at 345k resulted in unpredictable performance due to the nature of the test case and the relationship to the chosen cache size. Certain parts of the workload were in cache while others not, resulting in a harmonics effect.

Group members cache

The group members cache is an extension of the Entry cache. This cache stores member and uniquemember attribute values with their entries. The group entries will only be a part of the group members cache if the entry structures actually have members and uniquemembers. Otherwise, they will be a part of the regular entry cache. Group member caching can be controlled using two new configuration options:

- **ibm-slapdGroupMembersCacheSize:** This defines the number of groups whose members will be cached. The default value for this configuration option is 25.
- **ibm-slapdGroupMembersCacheBypassLimit:** This defines the maximum number of members a group can have in order for it to be cached in the group members cache. The default value of this configuration option is 25000.

ACL cache

The Access Control List (ACL) cache contains information about the access permissions of recently queried entries, such as the entry owner and whether the entry’s permissions are explicit or inherited. Having this information cached in

memory can speed up the process of determining whether the user who submitted the query is authorized to see all, some, or none of its results.

Measuring cache entry sizes

Filter cache and entry cache sizes are measured in numbers of entries. When determining how many entries to allow in your LDAP caches, it can be useful to know how big the entries in your cache are.

The following example shows how to measure the size of cached entries:

Note: This example calculates the average size of an entry in a sample entry cache, but the average filter cache entry size can be calculated similarly.

1. From the LDAP server:
 - a. Set the filter cache size to zero.
 - b. Set the entry cache size to a small value; for example, 200.
 - c. Start **ibmslapd**.
2. From the client:
 - a. Run your application.
 - b. Find the entry cache population (call this *population1*) using the following command:

```
ldapsearch -h servername -s base -b cn=monitor objectclass=* | grep entry_cache_current
```
3. From the LDAP Server:
 - a. Find the memory used by **ibmslapd** (call this *ibmslapd1*):
 - On AIX operating systems, use the following command:

```
ps -e -o vsz -o command | grep ibmslapd
```
 - On Windows operating systems, use the **VM size** column in the **Task Manager**.
 - b. Stop **ibmslapd**.
 - c. Increase the size of the entry cache but keep it smaller than your working set.
 - d. Start **ibmslapd**.
4. Run your application again and find the entry cache population (call this *population2*). See step 2b for the command syntax.
5. Find the memory used by **ibmslapd** (call this *ibmslapd2*). See step 3a for the command syntax.
6. Calculate the size of an entry cache entry using the following formula:
$$\frac{(\text{ibmslapd size2} - \text{ibmslapd size1})}{(\text{entry cache population2} - \text{entry cache population1})}$$

For example, using this formula with a 500,000-entry database results in the following measurement:

$$(192084 \text{ KB} - 51736 \text{ KB}) / (48485 - 10003) = 3.65 \text{ KB per entry}$$

LDAP cache configuration variables

LDAP cache configuration variables allow you to set the LDAP cache sizes, bypass limits, and other variables that affect performance.

Configuring attribute caching

The attribute cache size is measured by the amount of memory the attribute cache requires. You can configure the maximum amount of memory allowed to be used for attribute caching.

You can configure attribute caching for the directory database, the changelog database, or both. Typically, there is no benefit from configuring attribute caching for the changelog database unless you perform very frequent searches of the changelog.

Using the Web Administration Tool

To configure the attribute cache using the Web Administration Tool:

Expand the **Manage server properties** category in the navigation area of the Web Administration Tool, select the **Attribute cache** tab.

1. You can change the amount of memory in kilobytes available to the directory cache. The default is 16384 kilobytes (16 MB).
2. You can change the amount of memory in kilobytes available to the changelog cache. The default is 16384 kilobytes (16 MB).

Note: This selection is disabled if a changelog has not been configured.

To enable directory automatic attribute caching, perform the following steps:

1. Select the **Enable directory automatic attribute cache** check box. This enables other elements within this group.
2. Enter the start time for directory automatic attribute caching in the **Start Time** text box.
3. From the **Interval** combo box, select the interval at which the directory automatic attribute caching is to be performed again.

To enable change log automatic attribute caching, perform the following steps:

1. Select the **Enable change log automatic attribute cache** check box. This enables other elements within this group.
2. Enter the start time for change log automatic attribute caching in the **Start Time** text box.
3. From the **Interval** combo box, select the interval at which the change log automatic attribute caching is to be performed again.

Note: Automatic attribute caching for change log should not be enabled unless frequent searches within the change log are required and the performance of these searches are critical.

To add an attribute:

1. Select the attribute that you want to cache from the **Available attributes** drop-down menu. Only those attributes that can be designated as cached attributes are displayed in this menu. For example, sn.

Note: An attribute remains in the list of available attributes until it has been placed in both the Directory and the Changelog containers.

2. Click either **Add to Database** or **Add to Change log** button. The attribute is displayed in the appropriate list box. You can list the same attribute in both containers.
3. Repeat this process for each attribute you want to cache.

Note: An attribute is removed from the drop-down list when it is added to both the **Cached attributes under Database** and **Cached attributes under Change log** listboxes. If changelog is not enabled, then the **Add to Change log** button is disabled and the entry cannot be added to **Cached attributes under Change log** list box. The attribute is removed from the available attributes list when it is added to **Cached attributes under Database** list box.

4. When you are finished, click **Apply** to save your changes without exiting, or click **OK** to apply your changes and exit, or click **Cancel** to exit this panel without making any changes.

Using the command line

To configure the attribute cache through the command line, issue the following command:

```
ldapmodify -D <adminDN> -w<adminPW> -i<filename>
```

where <filename> contains the following, for example.

- For the directory database:

```
dn: cn=Directory, cn=RDBM Backends, cn=IBM Directory,
    cn=Schemas, cn=Configuration
changetype: modify
add: ibm-slapdCachedAttribute
ibm-slapdCachedAttribute: sn
-
add: ibm-slapdCachedAttribute
ibm-slapdCachedAttribute: cn
-
replace: ibm-SlapdCachedAttributeSize
ibm-SlapdCachedAttributeSize: 262144
```
- For the changelog database:

```
dn: cn=change log, cn=RDBM Backends, cn=IBM Directory,
    cn=Schemas, cn=Configuration
changetype: modify
add: ibm-slapdCachedAttribute
ibm-slapdCachedAttribute: changetype
-
replace: ibm-SlapdCachedAttributeSize
ibm-SlapdCachedAttributeSize: 32768
```

See the *IBM Tivoli Directory Server Version 6.1 Administration Guide* for more information.

Setting other LDAP cache configuration variables

You can set LDAP configuration variables using the Web Administration Tool or the command line.

Using the Web Administration Tool

To set LDAP configuration variables using the Web Administration Tool:

1. Expand the **Manage server properties** category in the navigation area of the Web Administration tool.
2. Click **Performance**.
3. You can modify any of the following configuration variables:
 - **Cache ACL information** — This option must be selected for the **Maximum number of elements in ACL cache** settings to take effect.
 - **Maximum number of elements in ACL cache (ACL cache size)** — The default is 25,000.

- **Maximum number of elements in entry cache** (entry cache size) — Specify the maximum number of elements in the entry cache. The default is 25,000. See “Entry cache” on page 13 for more information about the entry cache.
 - **Maximum number of elements in search filter cache** (filter cache size) — The search filter cache consists of the requested search filters and resulting entry identifiers that matched. On an update operation, all filter cache entries are invalidated. The default is 25,000. See “LDAP filter cache” on page 11 for more information about the filter cache.
 - **Maximum number of elements from a single search added to search filter cache** (filter cache bypass limit) — If you select **Elements**, you must enter a number. The default is 100. Otherwise select **Unlimited**. Search filters that match more entries than the number specified here are not added to the search filter cache. See “Filter cache bypass limits” on page 12 for more information about bypass limits.
4. When you are finished, click **OK** to apply your changes, or click **Cancel** to exit the panel without making any changes.

Using the command line

To set LDAP configuration variables using the command line, issue the following command:

```
ldapmodify -DAdminDN -wAdminpassword -ifilename
```

where the file *filename* contains:

```
dn: cn=Directory,cn=RDBM Backends,cn=IBM Directory,
    cn=Schemas,cn=Configuration
changetype: modify
replace: ibm-slapdDbConnections
ibm-slapdDbConnections: 15
```

```
dn: cn=Front End, cn=Configuration
changetype: modify
replace: ibm-slapdACLCache
ibm-slapdACLCache: TRUE
```

```
-
replace: ibm-slapdACLCacheSize
ibm-slapdACLCacheSize: 25000
```

```
-
replace: ibm-slapdEntryCacheSize
ibm-slapdEntryCacheSize: 25000
```

```
-
replace: ibm-slapdFilterCacheSize
ibm-slapdFilterCacheSize: 25000
```

```
-
replace: ibm-slapdFilterCacheBypassLimit
ibm-slapdFilterCacheBypassLimit: 100
```

Additional settings

There are several additional settings that affect performance by putting limits on client activity, minimizing the impact to server throughput and resource usage, such as:

- `ibm-slapdSizeLimit: 500`
- `ibm-slapdTimeLimit: 900`
- `ibm-slapdIdleTimeOut: 300`

- `ibm-slapdMaxEventsPerConnection`: 100
- `ibm-slapdMaxEventsTotal`: 0
- `ibm-slapdMaxNumOfTransactions`: 20
- `ibm-slapdMaxOpPerTransaction`: 5
- `ibm-slapdMaxTimeLimitOfTransactions`: 300
- `ibm-slapdPagedResAllowNonAdmin`: TRUE
- `ibm-slapdPagedResLmt`: 3
- `ibm-slapdSortKeyLimit`: 3
- `ibm-slapdSortSrchAllowNonAdmin`: TRUE

For more information about these settings, see "Appendix R. IBM Tivoli Directory Server configuration schema" in *IBM Tivoli Directory Server Version 6.1 Installation and Configuration Guide*.

Note: Default values are shown.

The IBM Tivoli Directory Server response time for searches with alias dereferencing option set to **always** or **searching** is significantly greater than that of searches with the dereferencing option set to **never**. A server-side configuration option `ibm-slapdDerefAliases` under `dn: cn=Configuration` can be used to override the dereference option specified in the client search requests. The allowed values are:

- **never**
- **find**
- **search**
- **always**

By setting the value to **never**, the server does not attempt to dereference possible aliases, and the response time for searches improves.

Directory size

It is important when you run your workload that you consider several measurements. For example, measuring the number of operations per second as shown in Figure 6 on page 20, it appears that performance degrades significantly as the database size grows.

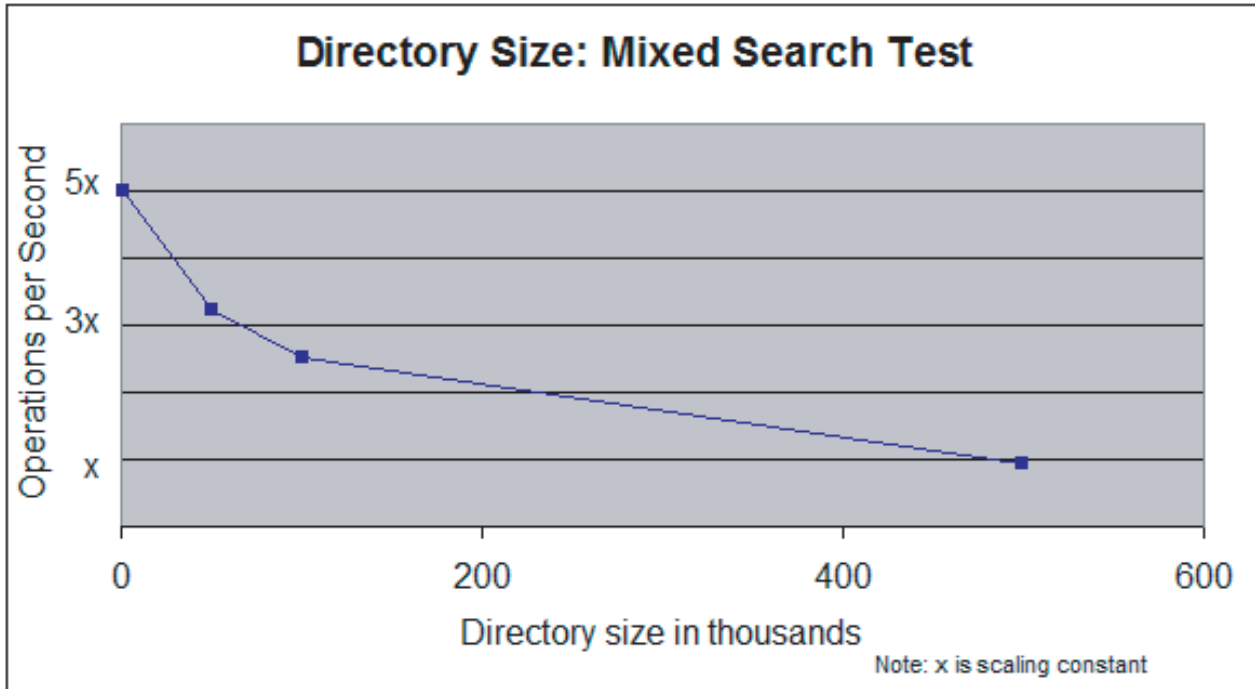


Figure 6. Operations per second

However, the benchmark tool test includes a large fraction of wildcard searches and exact-match searches, such as "(sn=Smith)" that return all entries where the sn value is "Smith". Both of these types of searches typically return multiple entries in response to a single search request. As Figure 7 on page 21 shows, as the size of the directory grows, so does the number of entries returned in response to wildcard and exact-match search requests.

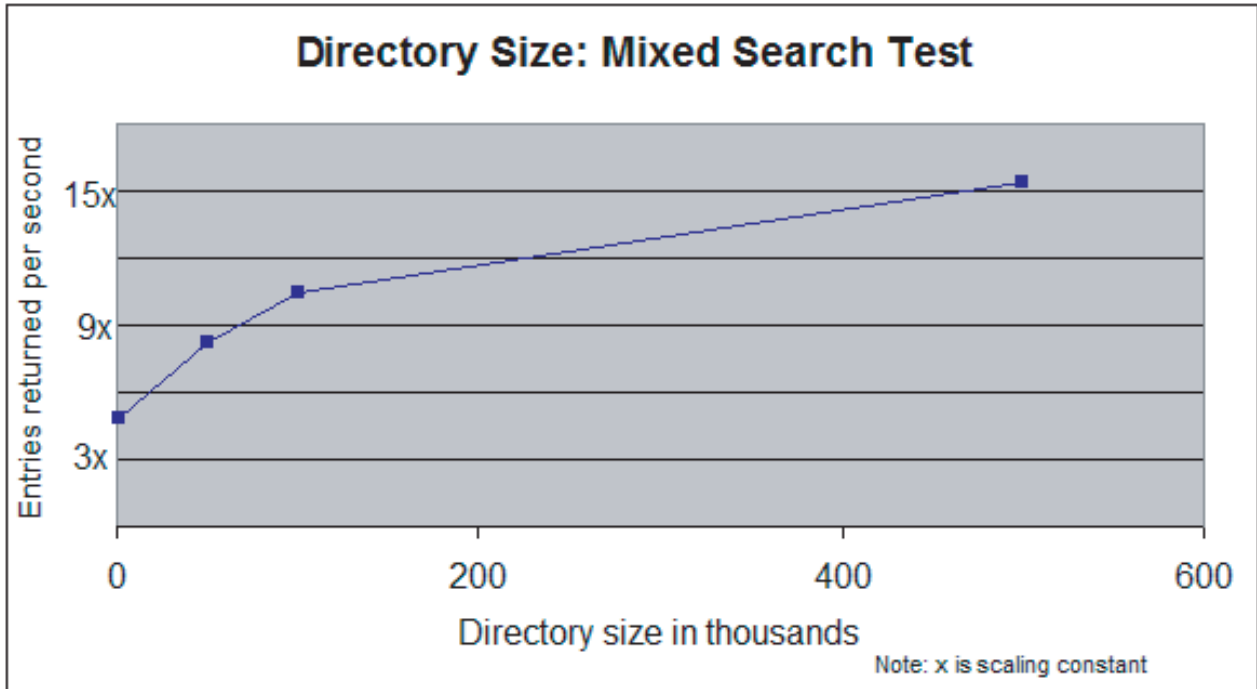


Figure 7. Entries returned per second

In this situation, the number of entries returned per second is a truer measure of throughput than operations per second, because each operation requires more work to be performed as the size of the database grows.

Note: As your directory grows, it might become necessary to readjust the sizes of the LDAP caches and DB2 buffer pools. You can determine the optimal sizes for your caches and buffer pools using the guidelines in “LDAP caches” on page 7 and “DB2 buffer pool tuning” on page 24.

Chapter 3. Tuning DB2 and LDAP caches

IBM Tivoli Directory Server uses DB2 as the data store and Structured Query Language (SQL) as the query retrieval mechanism. While the LDAP server caches LDAP queries, answers, and authentication information, DB2 caches tables, indexes, and statements.

Many DB2 configuration parameters affect either the memory (buffer pools) or disk resources. Since disk access is usually much slower than memory access, the key database performance tuning objective is to decrease the amount of disk activity.

This chapter covers the following areas:

- DB2 buffer pool tuning
- Tuning DB2 and LDAP caches using the **idsperftune** tool
- Optimization and organization (**reorgchk** and **reorg**)
- Other DB2 configuration parameters
- Backing up and restoring the database (**backup** and **restore**)
- Data row compression feature

For detailed information about DB2 commands, see the DB2 documentation at the following Web site: <http://www.ibm.com/software/data/db2/library/>

Attention: Only users listed as database administrators can run the DB2 commands. Be sure the user ID running the DB2 commands is a user in the `db2sysadm` group (UNIX operating systems) or a member of the Administrator group (Windows operating systems.) This includes the DB2 instance owner and root.

If you have any trouble running the DB2 commands, check to ensure that the DB2 environment variables have been established by running **db2profile** (if not, the **db2 get** and **db2 update** commands will not work). The script file **db2profile** is located in the `sqllib` subdirectory under the instance owner's home directory. If you need to tailor this file, follow the comments inside the file to set your instance name, user paths, and default database name (the default path is `/home/ldapdb2/sqllib/db2profile`.) It is assumed that the user is logged in as **ibm-slapdDbUserId**. If logged in as the root user on a UNIX operating system, it is possible to switch to the instance owner as follows:

```
su - instance_owner
```

where *instance_owner* is the defined owner of the LDAP database.

To log on as the database administrator on a Windows 2000 operating system, run the following command:

```
runas /user:instance_owner db2cmd
```

where *instance_owner* is the defined owner of the LDAP database.

Notes:

1. If you have problems connecting to the database on Windows systems, check the `DB2INSTANCE` environment variable. By default this variable is set to `DB2`. However, to connect to the database, the environment variable must be set to

the database instance name. For additional stability and performance enhancements, upgrade to the latest version of DB2.

2. In Version 9.1, the `self_tuning_mem` database configuration parameter is automatically set to ON when you create a single-partition database and sets the following values to AUTOMATIC.
 - `pckcachesz`
 - `locklist`
 - `maxlocks`
 - `sortheap`
 - `sheapthres_shr`
 - `database_memory` (You can set `database_memory` to AUTOMATIC only on Windows and AIX platforms.)

DB2 buffer pool tuning

DB2 buffer pool tuning is one of the most significant types of DB2 performance tuning. A buffer pool is a data cache between LDAP and the physical DB2 database files for both tables and indexes. DB2 buffer pools are searched when entries and their attributes are not found in the entry cache. Buffer pool tuning typically needs to be done when the database is initially loaded and when the database size changes significantly.

There are several considerations to keep in mind when tuning the DB2 buffer pools; for example:

- If there are no buffer pools, all database activity results in disk access.
- If the size of each buffer pool is too small, LDAP must wait for DB2 disk activity to satisfy DB2 SQL requests.
- If one or more buffer pools is too large, memory on the LDAP server might be wasted.
- If the total amount of space used by the LDAP caches and both buffer pools is larger than physical memory available on the server, operating system paging (disk activity) will occur.

To get the current DB2 buffer pool sizes, run the following commands:

```
db2 connect to database_name
db2 "select bpname,npages,pagesize from sysibm.sysbufferpools"
```

where *database_name* is the name of the database.

The following example output shows the default settings for the example above:

BPNAME	NPAGES	PAGESIZE
IBMDEFAULTBP	29500	4096
LDAPBP	1230	32768

2 record(s) selected.

Buffer pool sizes

The LDAP directory database (DB2) has two buffer pools: LDAPBP and IBMDEFAULTBP. The size of each buffer pool needs to be set separately, but the method for determining how big each should be is the same: Run your workload with the buffer pool sizes set to different values and measure the differences in operations per second.

Note: DB2 does not allow buffer pools to be set to zero.

LDAPBP buffer pool size

This buffer pool contains cached entry data (`ldap_entry`) and all of the associated indexes. LDAPBP is similar to the entry cache, except that LDAPBP uses different algorithms in determining which entries to cache. It is possible that an entry that is not cached in the entry cache is located in LDAPBP.

To determine the best size for your LDAPBP buffer pool, run your workload with the LDAPBP buffer pool size set to different values and measure the differences in operations per second. For example, Figure 8 shows varying operations per second based on different LDAPBP buffer pool sizes:

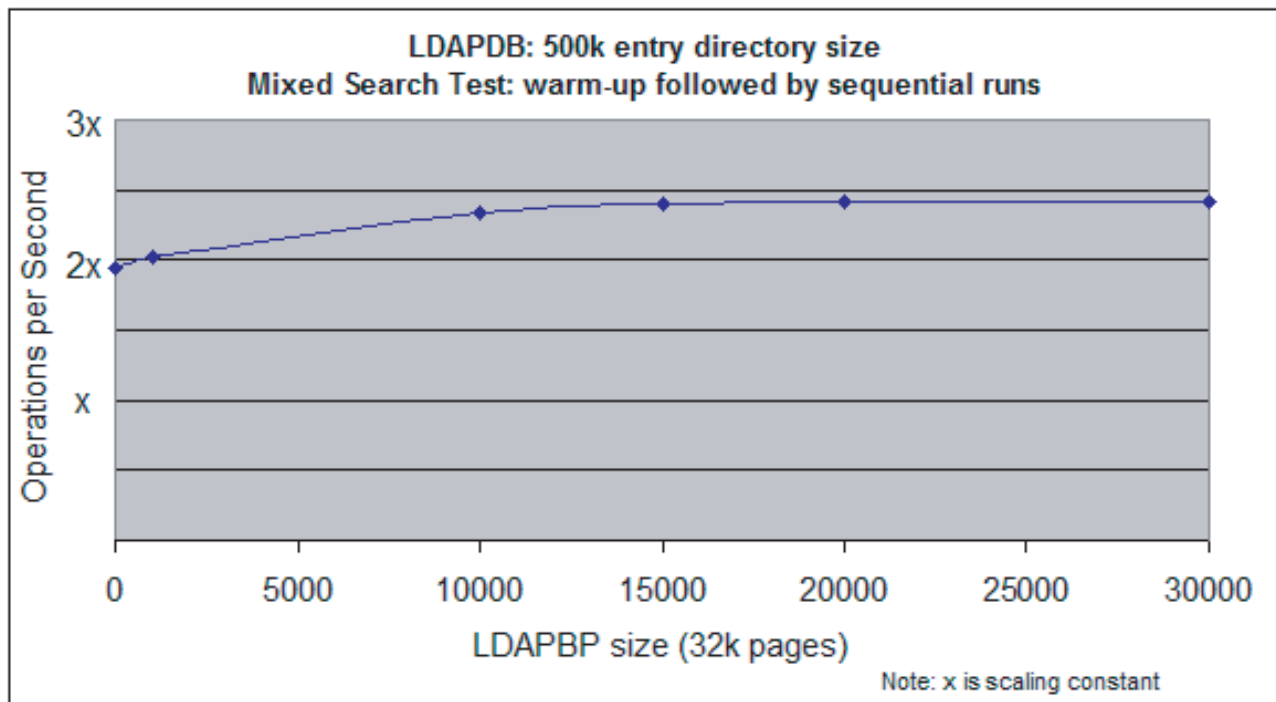


Figure 8. Varying the size of LDAPBP

For the workload in the above example, the best performance results from a LDAPBP size of approximately 15,000 32K pages. However, the performance gain of 15,000 over a size of 9,800 is slight. In a memory-constrained environment, setting the LDAPBP size to 9,800 saves approximately 166 MB of memory.

IBMDEFAULTBP buffer pool size

DB2 system information, including system tables and other information that is useful in resolving filters, is cached in the IBMDEFAULTBP buffer pool. You might need to adjust the IBMDEFAULTBP cache settings for better performance in the LDAPBP.

To determine the best size for your IBMDEFAULTBP buffer pool, run your workload with the buffer pool sizes set to different values and measure the differences in operations per second. For example, Figure 9 on page 26 shows varying operations per second based on different IBMDEFAULTBP buffer pool sizes:

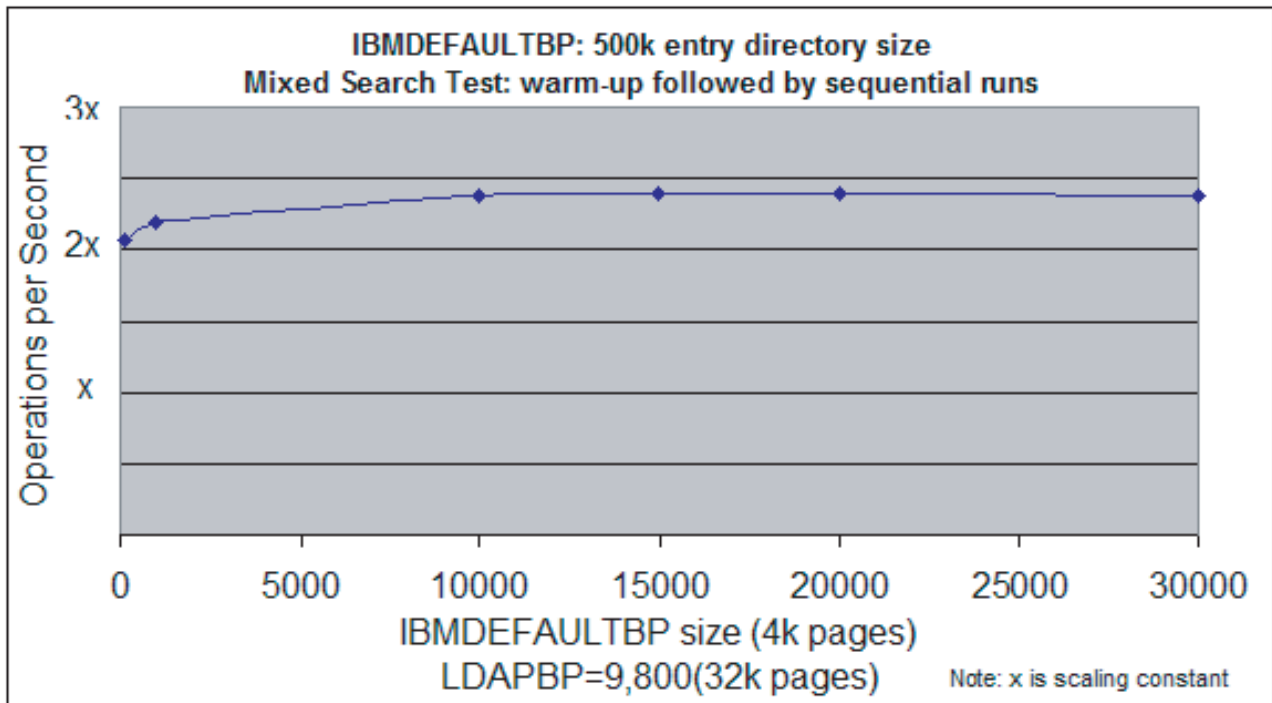


Figure 9. Varying the size of IBMDEFAULTBP

For the workload in the above example, setting the IBMDEFAULTBP large enough to hold the working set improves throughput approximately 20 percent over a small buffer pool size. There is little additional benefit to setting IBMDEFAULTBP larger than 20,000 4K pages.

Setting buffer pool sizes

Use the alter bufferpool command to set the IBMDEFAULTBP and LDAPBP buffer pool sizes. The following example shows the IBMDEFAULTBP and LDAPBP buffer pools being set:

```
db2 alter bufferpool ibmdefaultbp size 20000
db2 alter bufferpool ldapbp size 9800
db2 force applications all
db2stop
db2start
```

Note: The LDAP server (idsslapd) must be stopped while setting buffer pool sizes.

The Performance Tuning Tool (idsperftune)

The IBM Tivoli Directory sever provides a tool named **idsperftune** (the Performance Tuning Tool) that enables administrators to achieve higher directory server performance by tuning different caches, DB2 buffer pools, and DB2 parameters. The **idsperftune** tool works in two modes: basic and advanced.

Basic tuning

The basic tuning mode of operation deals with the tuning of the LDAP caches and the DB2 buffer pools. The LDAP caches include entry cache, filter cache, group member cache, and group member cache bypass limit while the DB2 buffer pools include IBMDEFAULTBP and LDAPBP. The basic tuning mode recommends optimum tuning values for LDAP caches and DB2 buffer pools and also updates the LDAP cache and DB2 buffer pool parameters to the settings that are specified.

The tool takes inputs from a property file named `perftune_input.conf`. The administrators must provide all inputs in the property file. Following are the input values that are taken from the property file. If values are not specified, then the default values are taken into consideration.

- Amount of system memory (%) to be allotted to Tivoli Directory Server instance:
This is the total memory that will be allocated to an instance and will be used as an input to the tool while tuning the size of entry cache, filter cache, and group member cache. If not specified, then the default value of 90% of system memory will be taken.
- Total number of entries that will reside in the directory:
This value is used as an input to the tool to estimate the size that should be assigned to the cache.

Note: The **idsperftune** tool provides a command line option to calculate the total number of entries and average size of entries that are present in the directory. For instance, if an administrator provides the command line option `"-s "` then **idsperftune** will compute total number of entries and average size of entries and log the details in the `perftune_input.conf` file. If the administrator does not provide the command line option, then the total number of entries is set to 10000. For further information about the **idsperftune** tool refer *IBM Tivoli Directory Server version 6.1 Command Reference*. For information about running the Performance Tuning Tool through the Configuration Tool, see the *IBM Tivoli Directory Server version 6.1 Installation and Configuration Guide*.

- Average size of entry (in bytes):
This value represents the average size of an entry that is expected to reside in memory. The average size of an entry and the total number of entries is used by the **idsperftune** tool to calculate the total size of the directory. Based on this, the size that should be allotted to Entry and Filter cache is calculated.

Note: The **idsperftune** tool provides a command line option to calculate the total number of entries and average size of entries that are present in the directory. For instance, if an administrator provides the command line option `"-s "` then **idsperftune** will compute total number of entries and average size of entries and log the details in the `perftune_input.conf` file. If the administrator does not provide the command line option, then the total number of entries is set to 10000. For further information about the **idsperftune** tool refer *IBM Tivoli Directory Server version 6.1 Command Reference*.

- Update Frequency:
The administrator must specify whether frequent updates, or only batch updates, are expected. If the administrator specifies that frequent updates are expected, then the filter cache is set to 0. Otherwise it is set to 1 KB.
- Total number of groups to be cached:
An administrator can tune this value by providing an estimate of the total number of groups whose members need to be cached. This should be the number of groups frequently used. If not specified, the default value of 25 is used.
- Average number of members in a group:
An administrator can tune this value to set the total number of members within a group that will be cached. If not specified, the default value of 25000 is used.
- Server instance name:

This value is taken from `IDS_LDAP_INSTANCE` environment variable. If this environment variable is not set then the server instance name is set to the name of the directory server instance that is present. However, if more than one instance is present and no instance name is provided by the administrator, then an appropriate error message is displayed.

The basic mode of operation also involves the tuning of the DB2 buffer pools `IBMDEFAULTBP` and `LDAPBP`. On execution of `idsperftune`, the size that is to be allotted to LDAP entry cache is calculated. If the system memory allotted to the directory server is sufficient to cache 80% of directory entries then the LDAP entry cache size is set to the size required to cache 80% of total entries. The DB2 buffer pools will then be set to the remaining available system memory.

On the other hand, if the system memory allotted to the directory server is not enough to cache 80% of total entries present in the directory then the entry cache is set to a minimum value which is 1000. Filter cache size is set to 0 and group member cache and bypass limit is set to 25 and 25000 respectively. The DB2 buffer pools are then assigned the amount of system memory that remains after allocation of the default values to LDAP caches. The system memory will be divided between DB2 bufferpools, `IBMDEFAULTBP` and `LDAPBP`, in the ratio of 1:3.

Advanced tuning

For this phase of tuning, the directory server should be deployed and populated with entries. Also, the directory server should be servicing client requests for some period of time. Administrators can run `idsperftune` with appropriate advanced tuning option to monitor different DB2 parameters. Different DB2 parameters that can be monitored for tuning at run time are as follows:

- `CATALOGCACHE_SZ`
- `PCKCACHESZ`
- `LOGFILSIZ`
- `LOCKLIST`
- `SORTHEAP`
- `MAXFILOP`
- `DBHEAP`
- `CHNGPGS_THRESH`
- `NUM_IOSERVERS`
- `NUM_IOCLEANERS`

To monitor DB2 parameters `SORTHEAP`, `MAXFILPO`, `DBHEAP`, `CHNGPGS_THRESH`, `NUM_IOSERVERS`, and `NUM_IOCLEANERS`, monitor switches `BUFFERPOOL` and `SORTHEAP` must be enabled. These switches allow DB2 to collect additional runtime data. However, enabling these monitor switches will have some negative impact on the performance of the directory server. If monitor switches `BUFFERPOOL` and `SORTHEAP` are not enabled then the status of these parameters is displayed as "Not Collected" in the property file.

Information gathered during the advanced tuning phase is logged in the property file `perftune_stat.log`. The information gathered describes if a particular DB2 parameter value needs to be increased or decreased in order to get better performance out of the directory server. Given below is a section from the property file `perftune_stat.log`.


```

# DB2 PARAMETER STATUS
#-----
CATALOGCACHE_SZ=      OK
SORTHEAP=             Increase
MAXFILOP=             OK
CHNGPGS_THREASH=     Decrease
SORTHEAP=             Not Collected
-----

```

Description of DB2 parameter status as mentioned above:

- **OK:** The value currently used for the DB2 parameter is optimal.
- **Increase:** The value of DB2 parameter must be increased to achieve optimal performance.
- **Decrease:** The value of DB2 parameter must be decreased to achieve optimal performance.
- **Not Collected:** The value of DB2 parameter is not monitored, this state will be observed for the DB2 parameters which need monitor switches to be turned on.

Optimization and organization (reorgchk and reorg)

DB2 uses a sophisticated set of algorithms to optimize the access to data stored in a database. These algorithms depend upon many factors, including the organization of the data in the database, and the distribution of that data in each table. Distribution of data is represented by a set of statistics maintained by the database manager.

In addition, IBM Tivoli Directory Server creates a number of indexes for tables in the database. These indexes are used to minimize the data accessed in order to locate a particular row in a table.

In a read-only environment, the distribution of the data changes very little. However, with updates and additions to the database, it is not uncommon for the distribution of the data to change significantly. Similarly, it is quite possible for data in tables to become ordered in an inefficient manner.

To remedy these situations, DB2 provides tools to help optimize the access to data by updating the statistics and to reorganize the data within the tables of the database.

Optimization

Optimizing the database updates statistics related to the data tables, which improves performance and query speed. Optimize the database periodically or after heavy database updates (for example, after importing database entries). The **Optimize database** task in the IBM Tivoli Directory Server Configuration Tool uses the DB2 **runstats** command to update statistical information used by the query optimizer for all the LDAP tables.

Note: The **reorgchk** command also updates statistics. If you are planning to do a **reorgchk**, optimizing the database is unnecessary. See “Database organization (reorgchk and reorg)” on page 30 for more information about the **reorgchk** command.

To optimize the database using the Configuration Tool:

1. Start the Configuration Tool by typing **idsxcfg** on the command line.
2. Click **Optimize database** on the left side of the window.

3. On the Optimize database window, click **Optimize**.

After a message displays indicating the database was successfully optimized, you must restart the server for the changes to take effect.

To optimize the database using the command line, run the following command:

```
runstats -I <instancename>
```

See "idsrunstats, runstats" in the *IBM Tivoli Directory Server Version 6.1 Administration Guide* for more information.

Run the following commands to update more db2 stats that might improve performance:

```
DB2 RUNSTATS ON TABLE table_name WITH DISTRIBUTION AND DETAILED INDEXES ALL SHRLEVEL REFERENCE
```

```
DB2 RUNSTATS ON TABLE ldapdb2.objectclass WITH DISTRIBUTION AND DETAILED INDEXES ALL SHRLEVEL REFERENCE
```

where *table_name* is the name of the table.

Note: Use the runstats utility to update the database statistics. This utility preserves some LDAP-specific tuning done on statistics. If you use the DB2 RUNSTATS command, do the following to restore the LDAP-specific settings:

```
db2 "connect to database <ldap_db_name>"
db2 "update sysstat.tables set card=9E18 where tablename='LDAP_DESC'
    and card<>9E18"
db2 "terminate"
```

Database organization (reorgchk and reorg)

Tuning the organization of the data in DB2 using the **reorgchk** and **reorg** commands is important for optimal performance.

The **reorgchk** command updates statistical information to the DB2 optimizer to improve performance, and reports statistics on the organization of the database tables.

The **reorg** command, using the data generated by **reorgchk**, reorganizes table spaces to improve access performance and reorganizes indexes so that they are more efficiently clustered. The **reorgchk** and **reorg** commands can improve both search and update operation performance.

Note: Tuning organizes the data on disk in a sorted order. Sorting the data on disk is beneficial only when accesses occur in a sorted order, which is not typically the case. For this reason, organizing the table data on disk typically yields little change in performance.

Performing a reorgchk

After a number of updates have been performed against DB2, table indexes can become sub-optimal and performance can degrade. Correct this situation by performing a DB2 **reorgchk** as follows:

```
db2 connect to ldapdb2
db2 reorgchk update statistics on table all
```

Where *ldapdb2* is the name of your database.

To generate a **reorgchk** output file (recommended if you plan to run the **reorg** command) add the name of the file to the end of the command, for example:
 db2 reorgchk update statistics on table all > reorgchk.out

The following is a sample **reorgchk** report:

db2 => reorgchk current statistics on table all

Table statistics:

F1: $100 * \text{OVERFLOW} / \text{CARD} < 5$
 F2: $100 * \text{TSIZE} / ((\text{FPAGES}-1) * (\text{TABLEPAGESIZE}-76)) > 70$
 F3: $100 * \text{NPAGES} / \text{FPAGES} > 80$

CREATOR	NAME	CARD	OV	NP	FP	TSIZE	F1	F2	F3	REORG
LDAPDB2	ACLPERM	2	0	1	1	138	0	-	100	---
LDAPDB2	ACLPROP	2	0	1	1	40	0	-	100	---
LDAPDB2	ALIASEDOBJECT	-	-	-	-	-	-	-	-	---
LDAPDB2	AUDIT	1	0	1	1	18	0	-	100	---
LDAPDB2	AUDITADD	1	0	1	1	18	0	-	100	---
LDAPDB2	AUDITBIND	1	0	1	1	18	0	-	100	---
LDAPDB2	AUDITDELETE	1	0	1	1	18	0	-	100	---
LDAPDB2	AUDITEXTOPEVENT	1	0	1	1	18	0	-	100	---
LDAPDB2	AUDITFAILEDOPONLY	1	0	1	1	18	0	-	100	---
LDAPDB2	AUDITLOG	1	0	1	1	77	0	-	100	---
...										
SYSIBM	SYSINDEXCOLUSE	480	0	6	6	22560	0	100	100	---
SYSIBM	SYSINDEXES	216	114	14	28	162216	52	100	50	*--
...										
SYSIBM	SYSPLAN	79	0	6	6	41554	0	100	100	---
SYSIBM	SYSPLANAUTH	157	0	3	3	9106	0	100	100	---
SYSIBM	SYSPLANDEP	35	0	1	2	5985	0	100	50	---

Index statistics:

F4: CLUSTERRATIO or normalized $\text{CLUSTERFACTOR} > 80$
 F5: $100 * (\text{KEYS} * (\text{ISIZE}+8) + (\text{CARD}-\text{KEYS}) * 4) / (\text{NLEAF} * \text{INDEXPAGESIZE}) > 50$
 F6: $(100-\text{PCTFREE}) * (\text{INDEXPAGESIZE}-96) / (\text{ISIZE}+12) ** (\text{NLEVELS}-2) * (\text{INDEXPAGESIZE}-96) / (\text{KEYS} * (\text{ISIZE}+8) + (\text{CARD}-\text{KEYS}) * 4) < 100$

CREATOR	NAME	CARD	LEAF	LVLS	ISIZE	KEYS	F4	F5	F6	REORG
---------	------	------	------	------	-------	------	----	----	----	-------

Table: LDAPDB2.ACLPERM

```

LDAPDB2 ACLPERM_INDEX          2    1    1    6    2 100  -  -  ---
Table: LDAPDB2.ACLPROP
LDAPDB2 ACLPROP_INDEX         2    1    1    6    2 100  -  -  ---
Table: LDAPDB2.ALIASEDOBJECT
LDAPDB2 ALIASEDOBJECT         -    -    -    -    -  -  -  -  ---
LDAPDB2 ALIASEDOBJECTI        -    -    -    -    -  -  -  -  ---
LDAPDB2 RALIASEDOBJECT        -    -    -    -    -  -  -  -  ---
Table: LDAPDB2.AUDIT
LDAPDB2 AUDITI                1    1    1    4    1 100  -  -  ---
Table: LDAPDB2.AUDITADD
LDAPDB2 AUDITADDI            1    1    1    4    1 100  -  -  ---
Table: LDAPDB2.AUDITBIND
LDAPDB2 AUDITBINDI          1    1    1    4    1 100  -  -  ---
Table: LDAPDB2.AUDITDELETE
LDAPDB2 AUDITDELETEI        1    1    1    4    1 100  -  -  ---
Table: LDAPDB2.AUDITEXTOPEVENT
...
Table: LDAPDB2.SN
LDAPDB2 RSN                   25012 148    2    14 25012 99 90  0  ---
LDAPDB2 SN                     25012 200    3    12 25012 99 61 119 --*
LDAPDB2 SNI                     25012 84     2    4 25012 99 87  1  ---
...
Table: LDAPDB2.TITLE
LDAPDB2 TITLEI                -    -    -    -    -  -  -  -  ---
Table: LDAPDB2.UID
LDAPDB2 RUID                   25013 243    3    17 25013  0 62 79  *--
LDAPDB2 UID                     25013 273    3    17 25013 100 55 79  ---
LDAPDB2 UIDI                     25013 84     2    4 25012 100 87  1  ---
Table: LDAPDB2.UNIQUEMEMBER
LDAPDB2 RUNIQUEMEMBER         10015 224    3    47 10015  1 60 44  *--
LDAPDB2 UNIQUEMEMBER          10015 284    3    47 10015 100 47 44  -*
LDAPDB2 UNIQUEMEMBERI         10015 14     2    4    7 100 69  8  ---
...
Table: SYSIBM.SYSFUNCTIONS
SYSIBM IBM127                   141    1    1    13 141 65  -  -  *--
SYSIBM IBM25                     141    2    2    34 141 100 72 60  ---
SYSIBM IBM26                     141    2    2    32 141 78 68 63  *--
SYSIBM IBM27                     141    1    1    23 68 80  -  -  *--
SYSIBM IBM28                     141    1    1    12 2 99  -  -  ---
SYSIBM IBM29                     141    1    1    4 141 100  -  -  ---
SYSIBM IBM30                     141    3    2    59 141 78 76 38  *--
SYSIBM IBM55                     141    2    2    34 141 99 72 60  ---
...

```

CLUSTERRATIO or normalized CLUSTERFACTOR (F4) will indicate REORG is necessary for indexes that are not in the same sequence as the base table. When multiple indexes are defined on a table, one or more indexes may be flagged as needing REORG. Specify the most important index for REORG sequencing.

Note: After performing a reorgchk, do the following to restore the LDAP-specific settings:

```

db2 "connect to database <ldap_db_name>"
db2 "update sysstat.tables set card=9E18 where tabname='LDAP_DESC'
and card<>9E18"
db2 "terminate"

```

See the information about the **idsrunstats** command in the *IBM Tivoli Directory Server Version 6.1 Command Reference*.

Using the statistics generated by **reorgchk**, run **reorg** to update database table organization. See “Performing a reorg.”

Keep in mind that **reorgchk** needs to be run periodically. For example, **reorgchk** might need to be run after a large number of updates have been performed. Note that LDAP tools such as **ldapadd**, **ldif2db**, and **bulkload** can potentially do large numbers of updates that require a **reorgchk**. The performance of the database should be monitored and a **reorgchk** performed when performance starts to degrade. See “Monitoring performance” on page 51 for more information.

reorgchk must be performed on all LDAP replicas because each replica uses a separate database. The LDAP replication process does not include the propagation of database optimizations.

Because LDAP caches prepared DB2 statements, you must stop and restart **ibmslapd** for DB2 changes to take effect.

Performing a reorg

After you have generated organizational information about the database using **reorgchk**, the next step in reorganization is finding the tables and indexes that need reorganizing and attempting to reorganize them. This can take a long time. The time it takes to perform the reorganization process increases as the DB2 database size increases.

In general, reorganizing a table takes more time than updating statistics. Therefore, performance might be improved significantly by updating statistics first.

To reorganize database table information:

1. If you have not done so already, run **reorgchk**:

```
db2 reorgchk update statistics on table all > reorgchk.out
```

The **reorgchk** update statistics report has two sections; the first section is the table information and the second section is the indexes. An asterisk in the last column indicates a need for reorganization.

2. To reorganize the tables with an asterisk in the last column:

```
db2 reorg table table_name
```

where *table_name* is the name of the table to be reorganized; for example, LDAPDB2.LDAP_ENTRY.

Generally speaking, because most data in LDAP is accessed by index, reorganizing tables is usually not as beneficial as reorganizing indexes.

3. To reorganize the indexes with an asterisk in the last column:

```
db2 reorg table table_name index index_name
```

where

- *table_name* is the name of the table; for example, LDAPDB2.LDAP_ENTRY.
- *index_name* is the name of the index; for example, SYSIBM.SQLO00414155358130.

4. Run **reorgchk** again. The output from **reorgchk** can then be used to determine whether the reorganization worked and whether it introduced other tables and indexes that need reorganizing.

Some guidelines for performing a reorganization are:

- If the number on the column that has an asterisk is close to the recommended value described in the header of each section and one reorganization attempt has already been done, you can probably skip a reorganization on that table or index.
- In the table LDAPDB2.LDAP_ENTRY there exists a LDAP_ENTRY_TRUNC index and a SYSIBM.SQL index. Preference should be given to the SYSIBM.SQL index if attempts to reorganize them seem to alternate between one or the other needing reorganization.
- When an attribute length is defined to be less than or equal to 240 bytes, the attribute table contains three columns: EID, attribute and reversed attribute columns. In this case, the forward index is created using the EID and attribute columns as index keys. For example, the attribute SN is defined to have the maximum length which is less than or equal to 240 bytes, so the attribute table contains the EID, SN and RSN columns and the following indexes are created for this attribute table:


```
LDAPDB2.RSN <----- A reverse index whose defined index keys are the EID
and RSN columns.
LDAPDB2.SN <----- A forward index whose defined index keys are the EID
and SN columns.
LDAPDB2.SNI <----- An update index whose defined index key is the EID column.
```
- Reorganize all the attributes that you want to use in searches. In most cases you will want to reorganize to the forward index, but in cases with searches beginning with '*', reorganize to the reverse index.
- When an attribute length is defined to be greater than 240 bytes, the attribute table contains four columns: EID, attribute, truncated attribute and reversed truncated attribute columns. In this case, the forward index is created using the EID and truncated attribute columns as index keys. For example, the attribute CN is defined to have the maximum length which is greater than 240 bytes, so the attribute table contains the EID, CN, CN_T and RCN_T columns and the following indexes are created for this attribute table:


```
LDAPDB2.RCN <----- A reverse index whose defined index keys are the EID
and RCN_T columns.
LDAPDB2.CN <----- A forward index whose defined index keys are the EID
and CN_T columns.
LDAPDB2.CNI <----- An update index whose defined index key is the EID column.
```

The following is another example showing reverse, forward, and update indexes example:

Table: LDAPDB2.SECUUID

```
LDAPDB2 RSECUUID <- This is a reverse index
LDAPDB2 SECUUID <- This is a forward index
LDAPDB2 SECUUIDI <- This is an update index
```

Data Row Compression

The data row compression feature provided by DB2 V9 reduces disk space requirements and also enhances the overall performance of the Tivoli Directory Server. The performance improvement in case of a compressed database could be as much as 10% more than databases without compression.

Use the DB2 commands in this section to estimate the possible storage savings for a given table. The examples demonstrate how to estimate the savings for OBJECTCLASS table. The following command creates a file called objectclass.inspect in the sqllib/db2dump directory:

```
db2 inspect rowcompeestimate table name OBJECTCLASS results keep objectclass.inspect
```

The following command formats a binary file to a readable format. The resulting file "objectclass.out" indicates the amount of disk space the compression would save on OBJECTCLASS table.

```
db2inspf objectclass.inspect objectclass.out
```

Notes:

1. Tables with reported saving estimate of more than 50% are good candidates for compression.
2. In a typical Tivoli Directory Server deployment, compression on LDAP_ENTRY and OBJECTCLASS tables is recommended.

Use the following steps to compress a DB2 table:

1. db2 ALTER TABLE <db2instance>.<tableName> COMPRESS YES
2. db2 REORG TABLE <db2instance>.<tableName>
3. db2 REORG INDEXES ALL FOR TABLE <db2instance>.<tableName>
4. db2 RUNSTATS ON TABLE <db2instance>.<tableName> WITH DISTRIBUTION AND DETAILED INDEXES ALL

Indexes

Indexing results in a considerable reduction in the amount of time it takes to locate requested data. For this reason, it can be very beneficial from a performance standpoint to index all attributes used in searches.

Use the following DB2 commands to verify that a particular index is defined. In the following example, the index being checked is for the attribute **seeAlso**:

```
db2 connect to database_name
db2 list tables for all | grep -i seeAlso
db2 describe indexes for table database_name.seeAlso
```

Where *database_name* is the name of your database.

If the second command fails or the last command does not return three entries, the index is not properly defined. The last command should return the following results:

IndexSchema	Index Name	Unique Rule	Number of Columns
-----	-----	-----	-----
LDAPDB2	SEEALSOI	D	1
LDAPDB2	SEEALSO	D	2
LDAPDB2	RSEEALSO	D	2

3 record (s) selected.

To have IBM Tivoli Directory Server create an index for an attribute, do one of the following:

- To create an index using the Web Administration Tool:
 1. Expand **Schema management** in the navigation area, and click **Manage attributes**.
 2. Click **Edit attribute**.
 3. On the **IBM extensions** tab, select the **Equality** check box under **Indexing rules**.
- To create an index from the command line, issue the following command:

```
ldapmodify -D cn=root -w root -i addindex.ldif
```

The addindex.ldif file should look like the following:

```
dn: cn=schema
changetype: modify
replace: attributetypes
attributetypes: ( 2.5.4.34
  NAME 'seeAlso'
  DESC 'Identifies another directory server entry that may
  contain information related to this entry.'
  SUP 2.5.4.49
  EQUALITY 2.5.13.1
  USAGE userApplications )
-
replace: ibmattributetypes
ibmattributetypes: ( 2.5.4.34
  DBNAME( 'seeAlso' 'seeAlso' )
  ACCESS-CLASS normal
  LENGTH 1000
  EQUALITY )
```

Note: After adding an index, you should run reorgchk to update statistical information for the DB2 optimizer regarding Index statistics for the new index.

Other DB2 configuration parameters

Performance benefits can come from setting other DB2 configuration parameters, such as APPLHEAPZ and LOGFILSIZ. The current setting of parameters can be obtained by issuing the following command:

```
db2 get database configuration for database name
```

where *database name* is the name of your database.

This command returns the settings of other DB2 configuration parameters as well.

The following command also shows the DB2 configuration parameters for the entire database instance:

```
db2 get database manager configuration
```

To set the DB2 configuration parameters use the following syntax:

```
db2 update database configuration for database name using \
parm name parm value
db2 force applications all
db2stop
db2start
```

where *database name* is the name of your database and where *parm name* is the parameter to change and *parm value* is the value it is to be assigned.

Changes to DB2 configuration parameters do not take effect until the database is restarted with **db2stop** and **db2start**.

Note: If applications are currently connected to the database, you must also do a **db2 force applications all** command prior to the db2stop.

For a list of DB2 parameters that affect performance, visit the DB2 Web site:
<http://www.ibm.com/software/data/db2>

Note: If DB2 recognizes that a parameter is configured insufficiently, the problem is posted to the diagnostic log (db2diag.log). For example, if the DB2 buffer pools are too large, DB2 overrides the buffer pool settings and uses a minimal configuration. No notice of the change in buffer pool sizes is given except in the diagnostic log, so it is important to view the log if you are experiencing poor performance. The db2diag.log file is located in the sqllib/db2dump directory under the instance owner's home directory. For example, the ldapdb2 instance can find the db2diag.log file in the /home/ldapdb2/sqllib/db2dump directory.

Database backup and restore considerations

When using the database **backup** and **restore** commands it is important to keep in mind that when you restore over an existing database, any tuning that has been done on that existing database is lost.

Chapter 4. AIX operating system tuning

This chapter discusses the following performance tuning tasks for the AIX operating system:

- Enabling large files
- Setting MALLOCTYPE
- Setting other environment variables
- Viewing **ibmslapd** environment variables

Enabling large files

The underlying AIX operating system files that hold the contents of a large directory can grow beyond the default size limits imposed by the AIX operating system. If the size limits are reached, the directory ceases to function correctly. The following steps make it possible for files to grow beyond default limits on an AIX operating system:

1. When you create the file systems that are expected to hold the directory's underlying files, you should create them as Enhanced Journaled File Systems or as Journaled File Systems with Large File Enabled. The file system containing the DB2 instance's home directory, and, if **bulkload** is used, the file system containing the **bulkload** temporary directory, are file systems that can be created this way.

Note: The default path is:

```
<instance_home>/tmp
```

2. Set the soft file size limit for the root, ldap, and the DB2 instance owner users to **-1**. A soft file size limit of **-1** for a user specifies the maximum file size for that user as unlimited. The soft file size limit can be changed using the **smitty chuser** command. Each user must log off and log back in for the new soft file size limit to take effect. You must also restart DB2.

Setting MALLOCTYPE

Set the MALLOCTYPE environment variable as follows:

On all AIX 5.x versions

Set MALLOCTYPE as follows:

```
export MALLOCTYPE=buckets
```

Note: If you want to use MALLOCTYPE buckets, you must use ML03 (contains the fix for APAR IY50668) or higher. You can get this from IBM Support (www.ibm.com/support). If you are using MALLOCTYPE buckets, you must set ulimits for the LDAP instance to the following:

```
# ulimit -m unlimited
# ulimit -d unlimited
```

You can find more information about MALLOCTYPE in the AIX documentation.

It is essential to note that the following environment variables improve the performance of the IBM Tivoli Directory server:

- SPINLOOPTIME=650 (for SMP systems)

- MALLOCMULTIHEAP=1 (for SMP systems)

Setting other environment variables

You might experience better performance by setting the AIXTHREAD_SCOPE and NODISCLAIM environment as shown in the following commands. Check the AIX documentation to see if these settings might be right for your installation.

AIXTHREAD_SCOPE

To set AIXTHREAD_SCOPE, use the following command:

```
export AIXTHREAD_SCOPE=S
```

NODISCLAIM

To set NODISCLAIM, use the following command:

```
export NODISCLAIM=TRUE
```

Viewing ibmslapd environment variables (AIX operating system only)

To view the environment settings and variables for your **ibmslapd** process, run the following command:

```
ps ewww PID | tr ' ' '\012' | grep = | sort
```

where *PID* is the **ibmslapd** process ID.

Example output:

```
ACLCACHE=YES
ACLCACHE_SIZE=25000
AIXTHREAD_SCOPE=S
AUTHSTATE=compat
A_z=!
CLASSPATH=/home/ldapdb2/sql1lib/java/db2java.zip:/home/ldapdb2/sql1lib/java/
db2jcc.jar:/home/ldapdb2/sql1lib/function:/home/ldapdb2/sql1lib/java/
db2jcc_license_cisuz.jar:/home/ldapdb2/sql1lib/java/db2jcc_license_cu.jar:.
DB2CODEPAGE=1208
DB2INSTANCE=ldapdb2
HOME=/
IDS_LDAP_HOME=/opt/IBM/ldap/V6.1
LANG=en_US
LC_FASTMSG=true
LD_LIBRARY_PATH=/home/ldapdb2/sql1lib/lib
LIBPATH=/opt/IBM/ldap/V6.1/lib64:/usr/lib:/home/ldapdb2/idsslapd-ldapdb2/
db2instance/lib:/opt/IBM/ldap/V6.1/db2/lib64:/usr/lib:/lib:/home/ldapdb2/
sql1lib/lib:.
LOCPATH=/usr/lib/nls/loc
LOGIN=root
LOGNAME=root
MAIL=/usr/spool/mail/root
MAILMSG=[YOU
MALLOCTYPE=buckets
NLSPATH=/opt/IBM/ldap/V6.1/nls/msg/%L/%N:/opt/IBM/ldap/V6.1/nls/msg/%L/%N.cat:/
usr/lib/nls/msg/%L/%N:/usr/lib/nls/msg/%L/%N.cat
NODISCLAIM=TRUE
ODBCCONN=15
ODMDIR=/etc/objrepos
PATH=/opt/IBM/ldap/V6.1/sbin:/opt/IBM/ldap/V6.1:/usr/bin:/etc:/usr/sbin:/usr/
ucb:/usr/bin/X11:/sbin:/usr/java14/jre/bin:/usr/java14/bin:/usr/java131/jre/
bin:/usr/java131/bin:/home/ldapdb2/sql1lib/bin:/home/ldapdb2/sql1lib/adm:/
home/ldapdb2/sql1lib/misc
PWD=/home/ldapdb2/idsslapd-ldapdb2/workdir
RDBM_CACHE_BYPASS_LIMIT=100
RDBM_CACHE_SIZE=25000
RDBM_FCACHE_SIZE=25000
```

```
SHELL=/usr/bin/ksh
SSH_CLIENT=9.48.85.122
SSH_CONNECTION=9.48.85.122
SSH_TTY=/dev/pts/1
TERM=xterm
TISDIR=/opt/IBM/ldap/V6.1
TZ=CST6CDT
USER=root
VWSPATH=/home/ldapdb2/sql1lib
_=/opt/IBM/ldap/V6.1/sbin/64/ibmslapd
instance=ldapdb2
location=/home/ldapdb2
```

Chapter 5. Hardware tuning

This chapter contains some suggestions for improving disk drive performance.

Disk speed improvements

With millions of entries in LDAP server, it can become impossible to cache all of them in memory. Even if a smaller directory size is cacheable, update operations must go to disk. The speed of disk operations is important. Here are some considerations for helping to improve disk drive performance:

- Use fast disk drives
- Use a hardware write cache
- Spread data across multiple disk drives
- Spread the disk drives across multiple I/O controllers
- Put log files and data on separate physical disk drives

Chapter 6. IBM Tivoli Directory Server features

The sections in this chapter briefly describe the following additional performance-related IBM Tivoli Directory Server features.

- Bulk loading (**bulkload**)
- Replication
- Monitoring Performance
- When to configure the LDAP change log

Bulkload

The bulkload utility accepts many command line options introduced in previous releases for performance tuning. Many of these tuning options are deprecated. The default for the following options are optimal and should not be specified.

-A <yes|no>

Specifies whether to process the ACL information contained in the LDIF file. The default is **yes**. The **no** parameter loads the default acs.

-c | -C <yes|no>

Allows you to skip index recreation. For example, if you are running successive idsbulkloads and you want to skip recreation between loads, you can postpone index creation until the last idsbulkload. Issue the final idsbulkload with **-c yes**.

-e <yes|no>

Drop indexes before load.

Effects of using the **-k** option

The **-k** option enables users to bulkload their data in smaller chunks. It is especially useful for systems where memory is limited. What happens when utilizing this option is that the parsing and corresponding loading is done in smaller increments.

Note: Saving on memory by specifying small chunksize can result in the user experiencing longer bulkload times.

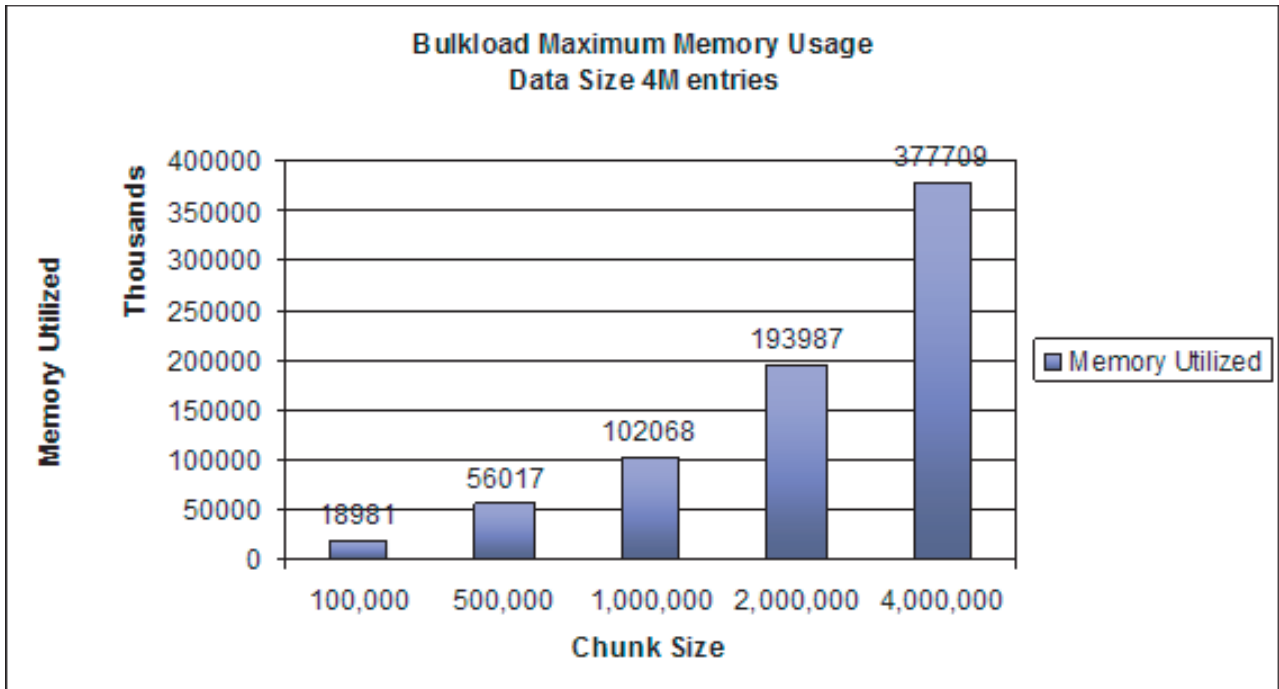


Figure 10. Effects of using the -k option

This graph illustrates the effects on memory usage. As the chunk size increases, the amount of memory utilized increases.

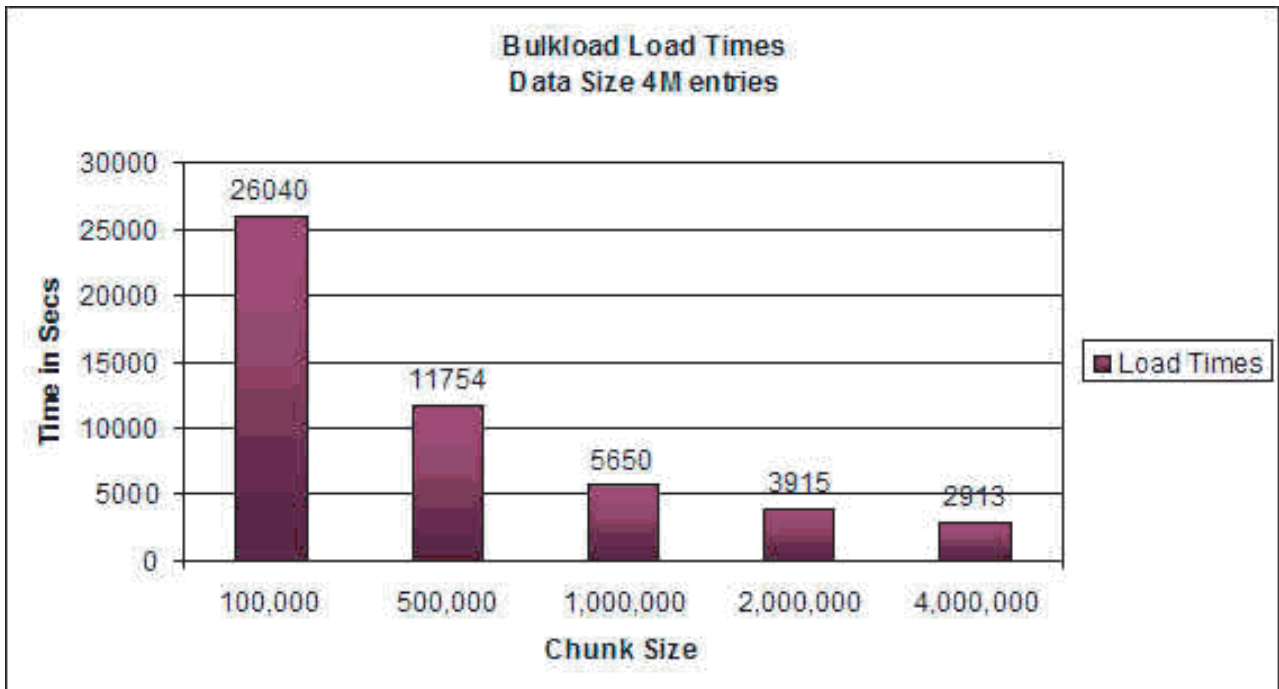


Figure 11. Effects of using the -k option

This graph illustrates that as the chunk size increases, the load time decreases. The recommendation is to use chunk sizes of one million entries at least.

Replication tuning

Replication is a technique used by directory servers to improve performance, availability, and reliability. The replication process synchronizes data stored in multiple directory servers.

Using multi-threaded (asynchronous) replication, administrators can replicate using multiple threads. These features were added to improve overall throughput of replication. For more information about asynchronous replication, see "Multi-threaded (asynchronous) replication" in the *IBM Tivoli Directory Server Version 6.1 Administration Guide*.

Anyone with a replication backlog might consider switching to multi-threaded (asynchronous) replication. Candidate environments can include the following:

- A high update rate
- No downlevel servers
- Common AES salt and synchronization if encryption is AES and passwords are updated often
- Small fanout (for example, 8 connections per agreement with 24 replicas might be too complicated depending on system configuration)
- Available servers and reliable network
- Real-time data consistency is not critical
- All replication schedules are immediate
- Multiprocessor machines

Multi-threaded (asynchronous) replication is difficult to administer if servers or networks are not reliable.

When errors occur, the errors are logged and can be replayed by the administrator, but the error logs must be monitored closely. The following is a search to show the replication backlog for all agreements supplied by one server:

```
ldapsearch -h supplier-host -D cn=admin -w ? -s sub -b <replication_context>
objectclass=ibm-replicationagreement
ibm-replicationpendingchangeount ibm-replicationstate
```

If the replication state is active, and the pending count is growing, there is a backlog that won't decrease unless the update rate decreases or the replication mode is changed from synchronous to asynchronous (multi-threaded).

Replication also adds to the workload on the master server where the updates are first applied. In addition to updating its copy of the directory data, the master server must send the changes to all replica servers. If your application or users do not depend on immediate replication, then careful scheduling of replication to avoid peak activity times will help minimize the impact to throughput on the master server. See "Creating replication schedules" in the *IBM Tivoli Directory Server Version 6.1 Administration Guide*.

The following are areas where tuning adjustment can be made to improve performance:

- Number of replication threads per supplier and consumer
- Replication context cache size
- Replication ready size limit

Number of replication threads

The number of replication threads (`ibm-replicaconsumerconnections`) attribute represents the number of connections used for each replication agreement. In testing, as the number of threads on both the supplier and consumer are increased, the transaction rate also increased. In the following graph, a transaction is defined as a queued replication record that is sent over, in this case an `ldap_modify`, to the supplier. The queued replication records (`ldap_modify`) are run with replication in a pending state. The replication state is then changed to **resume**, which starts the replication process.

Note: As the number of threads increases, so does the CPU usage on both supplier and consumer systems. Adjust this attribute as needed based on acceptable CPU usage and desired throughput.

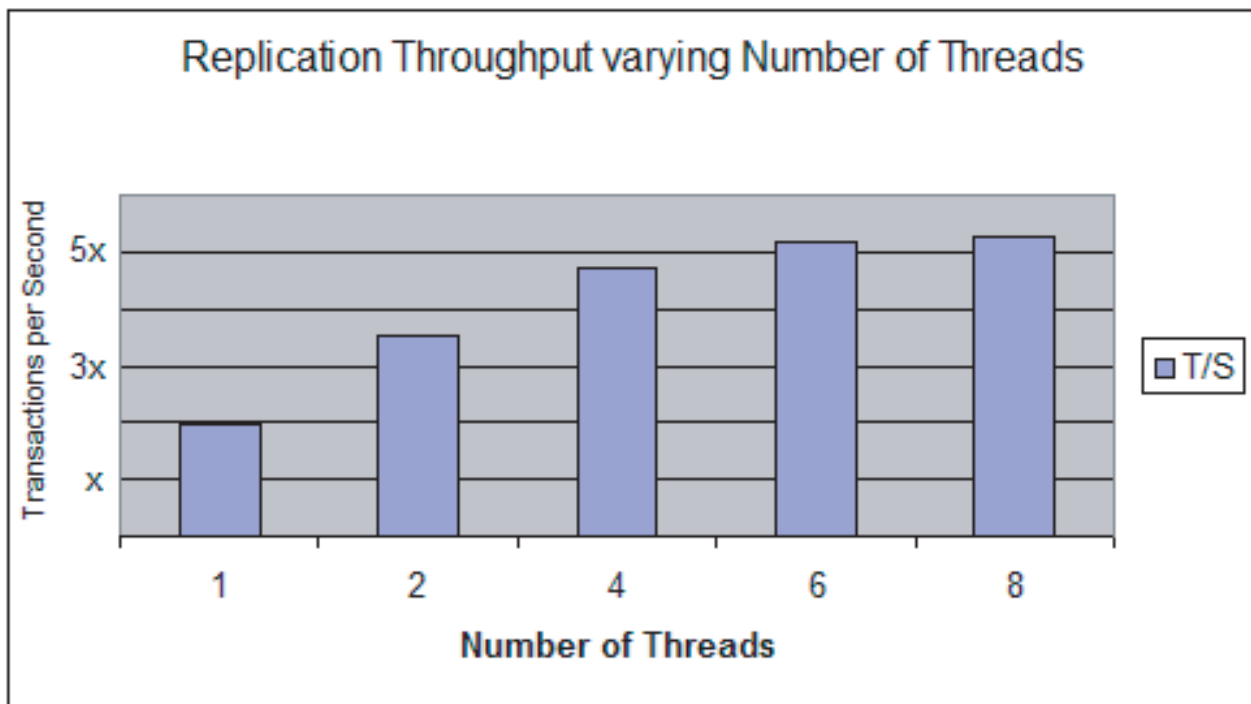


Figure 12. Number of replication threads

As the throughput increased, the CPU consumption on both the supplier and consumer increased. The CPU cost per transaction on the consumer increased slightly when adding threads, as there were more threads to manage.

Replication context cache size

The replication context cache size (`ibm-slapdReplContextCacheSize`) is an attribute that specifies, in bytes, the size in memory that is allocated to cache updates to be replicated. The default setting is 100,000 bytes. This attribute cannot be updated dynamically.

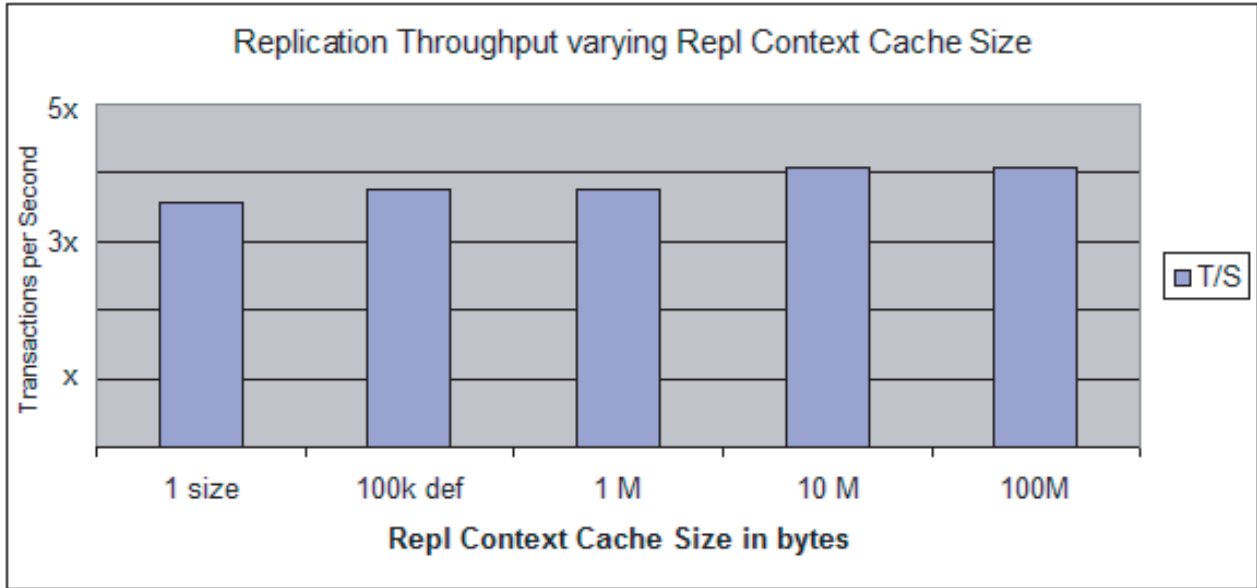


Figure 13. Replication context cache size

Replication ready size limit

The replication ready size limit (environment variable `IBMSLDAPD_REPL_READY_SIZE_LIMIT`) controls the size of the queues of replication operations from the list of updates still to be replicated. The default size is 10. There is one queue per connection to a given replica. Related updates (for example, modifications or children of new entries) will be placed in the same queue. If the size of this queue exceeds the specified size limit, the main replication thread waits for the queue size to get below the limit again. This prevents the main replication thread from using too much CPU determining dependencies between the updates. In testing, the size of this queue was varied from 1 entry up to 200 entries. Although an increase in raw throughput was not evident, CPU savings were realized at certain settings of this parameter. The following chart displays throughput normalized to 100% CPU. Absolute throughput did not change in this test. A larger number in this graph means less CPU cost per transaction. In this graph a transaction is defined as a queued replication record (`ldap_modify`) that is sent to the supplier.

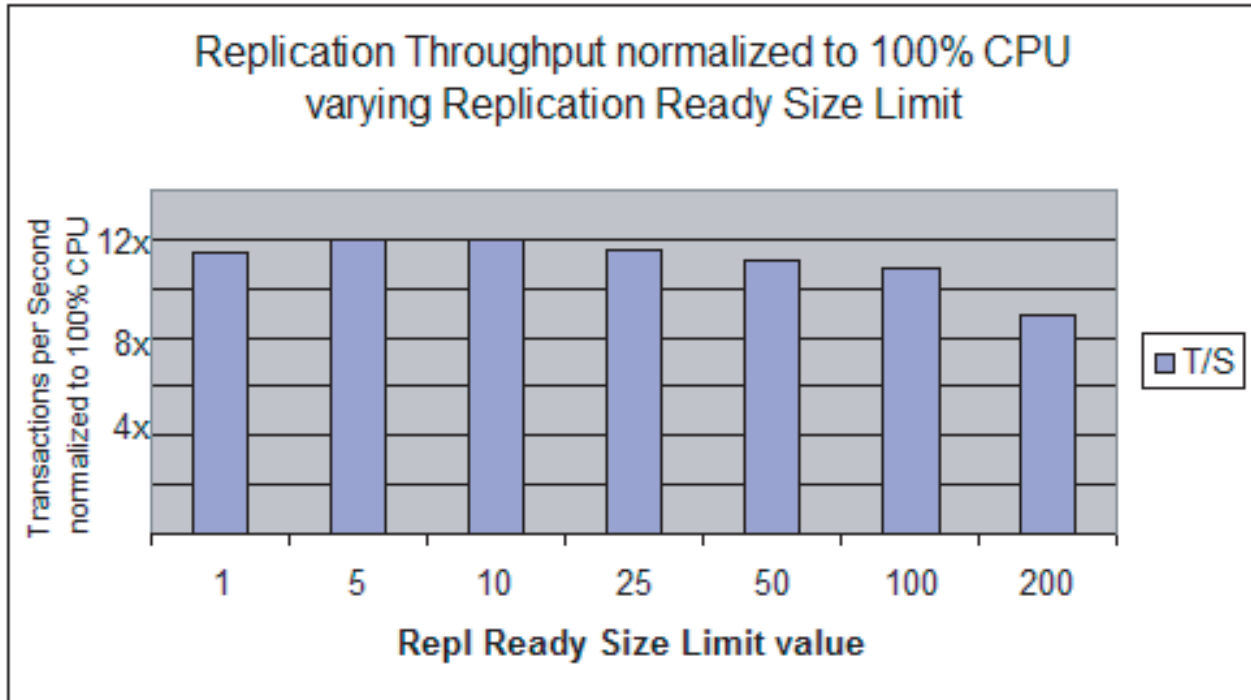


Figure 14. Replication ready size limit

Proxy server tuning

The proxy server is recommended for use in environments where the size of the data store exceeds the processing power and physical capacity of a single machine. Directory sizes greater than 40 M entries are candidates for a distributed directory environment. The proxy server gives customers the ability to distribute data across multiple backend servers.

Throughput performance of the proxy server can be affected by the size of the connection pool. This is a parameter configured on the Proxy server. For best results, the following guidelines to set an appropriate connection pool size must be followed:

- Configure more than one connection to the back-end server.
- Limit the connection pool size to the number of connections the operating system can support. The connection pool is a static pool of connections that the proxy server sets up during the proxy server startup. The operating system imposes a limit on the number of open file descriptors. The connection pool size should be less than this limit.
- Ensure that the connection pool size is less than the number of database connections configured with the back-end server, keeping a buffer for replication and changelog.

For better performance, all back-end servers and the proxy server should share the same stash files.

Monitoring performance

The `ldapsearch` command can be used to monitor performance, as shown in the following sections.

ldapsearch with "cn=monitor"

The following `ldapsearch` command uses "cn=monitor".

```
ldapsearch -h ldap_host -s base -b cn=monitor objectclass=*
```

where *ldap_host* is the name of the LDAP host.

The monitor search returns some of the following attributes of the server:

cn=monitor

version=IBM Tivoli Directory, Version 6.1

total connections

The total number of connections since the server was started.

current connections

The number of active connections.

maxconnections

The maximum number of active connections allowed.

writewaiters

The number of threads sending data back to the client.

readwaiters

The number of threads reading data from the client.

livethreads

The number of worker threads being used by the server.

filter_cache_size

The maximum number of filters allowed in the cache.

filter_cache_current

The number of filters currently in the cache.

filter_cache_hit

The number of filters retrieved from the cache rather than being resolved in DB2.

filter_cache_miss

The number of filters that were not found in the cache that then needed to be resolved by DB2.

filter_cache_bypass_limit

Search filters that return more entries than this limit are not cached.

entry_cache_size

The maximum number of entries allowed in the cache.

entry_cache_current

The number of entries currently in the cache.

entry_cache_hit

The number of entries that were retrieved from the cache.

entry_cache_miss

The number of entries that were not found in the cache that then needed to be retrieved from DB2.

acl_cache

A Boolean value indicating that the ACL cache is active (TRUE) or inactive (FALSE).

acl_cache_size

The maximum number of entries in the ACL cache.

currenttime

The current time on the server. The current time is in the format:
year month day hour:minutes:seconds GMT

Note: If expressed in local time the format is

day month date hour:minutes:seconds timezone year

starttime

The time the server was started. The start time is in the format:
year month day hour:minutes:seconds GMT

Note: If expressed in local time the format is

day month date hour:minutes:seconds timezone year

en_currentregs

The current number of client registrations for event notification.

en_notificationssent

The total number of event notifications sent to clients since the server was started.

The following attributes are for operation counts:

bindsrequested

The number of bind operations requested since the server was started.

bindscompleted

The number of bind operations completed since the server was started.

unbindsrequested

The number of unbind operations requested since the server was started.

unbindscompleted

The number of unbind operations completed since the server was started.

addsrequested

The number of add operations requested since the server was started.

addscompleted

The number of add operations completed since the server was started.

deletesrequested

The number of delete operations requested since the server was started.

deletescompleted

The number of delete operations completed since the server was started.

modrdnsrequested

The number of modify RDN operations requested since the server was started.

modrdnscompleted

The number of modify RDN operations completed since the server was started.

modifiesrequested

The number of modify operations requested since the server was started.

modifiescompleted

The number of modify operations completed since the server was started.

comparesrequested

The number of compare operations requested since the server was started.

comparescompleted

The number of compare operations completed since the server was started.

abandonsrequested

The number of abandon operations requested since the server was started.

abandonscompleted

The number of abandon operations completed since the server was started.

extopsrequested

The number of extended operations requested since the server was started.

extopscompleted

The number of extended operations completed since the server was started.

unknownopsrequested

The number of unknown operations requested since the server was started.

unknownopscompleted

The number of unknown operations completed since the server was started. Unrecognized operations are rejected with a result message to the client including the LDAP_UNWILLING_TO_PERFORM result code.

opsinitiated

The number of initiated requests since the server was started.

opscompleted

The number of completed requests since the server was started.

entriessent

The number of entries sent by the server since the server was started.

searchesrequested

The number of initiated searches since the server was started.

searchescompleted

The number of completed searches since the server was started.

The following attributes are for server logging counts:

slapderrorlog_messages

The number of server messages recorded since the server was started or since a reset was performed.

slapdclierrors_messages

The number of DB2 error messages recorded since the server was started or since a reset was performed.

auditlog_messages

The number of audit messages recorded since the server was started or since a reset was performed.

auditlog_failedop_messages

The number of failed operation messages recorded since the server was started or since a reset was performed.

The following attributes are for connection type counts:

total_ssl_connections

The total number of SSL connections since the server was started.

total_tls_connections

The total number of TLS connections since the server was started.

The following attributes are for tracing:

trace_enabled

The current trace value for the server. TRUE, if collecting trace data, FALSE, if not collecting trace data.

trace_message_level

The current ldap_debug value for the server. The value is in hexadecimal form, for example:

0x0=0
0xffff=65535

trace_message_log

The current LDAP_DEBUG_FILE environment variable setting for the server.

The following attributes are for denial of service prevention:

available_workers

The number of worker threads available for work.

current_workqueue_size

The current depth of the work queue.

largest_workqueue_size

The largest size that the work queue has ever reached.

idle_connections_closed

The number of idle connections closed by the Automatic Connection Cleaner.

auto_connection_cleaner_run

The number of times that the Automatic Connection Cleaner has run.

The following attribute is for alias dereference processing:

bypass_deref_aliases

The server runtime value that indicates if alias processing can be bypassed. It displays TRUE if no alias object exists in the directory, and FALSE if at least one alias object exists in the directory.

The following attributes are for the attribute cache:

cached_attribute_total_size

The amount of memory used by the directory attribute cache, in kilobytes. This number includes additional memory used to manage the cache that is not charged to the individual attribute caches. Consequently, this total is larger than the sum of the memory used by all the individual attribute caches.

cached_attribute_configured_size

The maximum amount of memory, in kilobytes, that is enabled to be used by the directory attribute cache.

cached_attribute_hit

The number of times the attribute has been used in a filter that could be processed by the attribute cache. The value is reported as follows:

```
cached_attribute_hit=attrname:####
```

cached_attribute_size

The amount of memory used for this attribute in the attribute cache. This value is reported in kilobytes as follows:

```
cached_attribute_size=attrname:#####
```

cached_attribute_candidate_hit

A list of up to ten most frequently used noncached attributes that have been used in a filter that could have been processed by the directory attribute cache if all of the attributes used in the filter had been cached. The value is reported as follows:

```
cached_attribute_candidate_hit=attrname:####
```

You can use this list to help you decide which attributes you want to cache. Typically, you want to put a limited number of attributes into the attribute cache because of memory constraints.

Examples

The following sections show examples of using values returned by the **ldapsearch** command with "cn=monitor" to calculate the throughput of the server and the number of add operations completed on the server in a certain timeframe.

Throughput example: The following example shows how to calculate the throughput of the server by monitoring the server statistic called **opscompleted**, which is the number of operations completed since the LDAP server started.

Suppose the values for the **opscompleted** attribute obtained by issuing two **ldapsearch** commands to monitor the performance statistics, one at time **t1** and the other at a later time **t2**, were **opscompleted (t1)** and **opscompleted (t2)**. The average throughput at the server during the interval between **t1** and **t2** can be calculated as:

```
(opscompleted(t2) - opscompleted(t1) - 3)/(t2 -t1)
```

(3 is subtracted to account for the number of operations performed by the **ldapsearch** command itself.)

Workload example: The monitor attributes can be used to characterize the workload, similar to the throughput example but split out by type of operation. For example, you can calculate the number of add operations that were completed in a certain amount of time.

Suppose the values for the **addscompleted** attribute obtained by issuing two **ldapsearch** commands to monitor the performance statistics, one at time **t1** and the other at a later time **t2**, were **addscompleted (t1)** and **addscompleted (t2)**. The number of add operations completed on the server during the interval between **t1** and **t2** can be calculated as:

```
(addscompleted(t2) - addscompleted(t1) / (t2 -t1)
```

Similar calculations can be done for other operations, such as **searchescompleted**, **bindscompleted**, **deletescompleted**, and **modifiescompleted**.

Idapsearch with "cn=workers,cn=monitor"

An administrator can run a search using "cn=workers,cn=monitor" to get information about what worker threads are doing and when they started doing it.

```
ldapsearch -D <adminDN> -w <adminpw> -b cn=workers,cn=monitor -s base objectclass=*
```

This information is most useful when a server is performing poorly or not functioning as expected. It should be used only when needed to give insight into what the server is currently doing or not doing.

The "cn=workers, cn=monitor" search returns detailed activity information only if auditing is turned on. If auditing is not on, "cn=workers, cn=monitor" returns only thread information for each of the workers.

Attention: The "cn=workers,cn=monitor" search suspends all server activity until it is completed. For this reason, a warning should be issued from any application before issuing this feature. The response time for this command will increase as the number of server connections and active workers increase.

For more information, see the *IBM Tivoli Directory Server Version 6.1 Administration Guide*.

Idapsearch with "cn=connections,cn=monitor"

An administrator can run a search using "cn=connections,cn=monitor" to get information about server connections:

```
ldapsearch -D<adminDN> -w <adminPW> -h <servername> -p <portnumber>  
-b cn=connections,cn=monitor -s base objectclass=*
```

This command returns information in the following format:

```
cn=connections,cn=monitor  
connection=1632 : 9.41.21.31 : 2002-10-05 19:18:21 GMT : 1 : 1 : CN=ADMIN : :  
connection=1487 : 127.0.0.1 : 2002-10-05 19:17:01 GMT : 1 : 1 : CN=ADMIN : :
```

Note: If appropriate, an SSL or a TLS indicator is added on each connection.

For more information, see the *IBM Tivoli Directory Server Version 6.1 Administration Guide*.

Idapsearch with "cn=changelog,cn=monitor"

You can run a search using "cn=changelog,cn=monitor" to obtain information about the changelog attribute cache. (See "When to configure the LDAP change log" on page 57 for information about the change log.) The command returns the following information:

cached_attribute_total_size

The amount of memory used by the changelog attribute cache, in kilobytes. This number includes additional memory used to manage the cache that is not charged to the individual attribute caches. Consequently, this total is larger than the sum of the memory used by all the individual attribute caches.

cached_attribute_configured_size

The maximum amount of memory, in kilobytes, that is enabled to be used by the changelog attribute cache

cached_attribute_hit

The number of times the attribute has been used in a filter that could be processed by the changelog attribute cache. The value is reported as follows:

```
cached_attribute_hit=attrname:####
```

cached_attribute_size

The amount of memory used for this attribute in the changelog attribute cache. This value is reported in kilobytes as follows:

```
cached_attribute_size=attrname:#####
```

cached_attribute_candidate_hit

A list of up to ten most frequently used noncached attributes that have been used in a filter that could have been processed by the changelog attribute cache if all of the attributes used in the filter had been cached. The value is reported as follows:

```
cached_attribute_candidate_hit=attrname:####
```

You can use this list to help you decide which attributes you want to cache. Typically, you want to put a limited number of attributes into the attribute cache because of memory constraints.

When to configure the LDAP change log

IBM Tivoli Directory Server has a function called **change log** that results in a significantly slower LDAP update performance. The **change log** function should be configured only if needed.

The **change log** function causes all updates to LDAP to be recorded in a separate change log DB2 database (that is, a different database from the one used to hold the LDAP server Directory Information Tree). The change log database can be used by other applications to query and track LDAP updates. The change log function is disabled by default.

One way to check for existence of the **change log** function is to look for the suffix CN=CHANGELOG. If it exists, the **change log** function is enabled.

Chapter 7. Capacity Planning

There are several hardware decisions to be made before deploying IBM Tivoli Directory Server. Hardware resources to consider are:

- Hard disk
- Memory
- CPUs

IBM Tivoli Directory Server can perform differently for various hardware configurations, and it is important to understand which hardware configurations cause the directory server to perform best.

Several tuning measures were taken to find the best settings under which the server provides best results. Tuning factors tested were:

IBM Tivoli Directory Server tuning

LDAP Entry cache

DB2 tuning

- DB2 bufferpool
 - LDAP bufferpool
 - IBMDEFAULT bufferpool
- Optimization and organization (reorgchk and reorg)
- Other DB2 configuration parameters
- Backing up and restoring the database (backup and restore)
- Splitting of database

The capacity planning information in this chapter includes results observed from data loading (using the **bulkload** utility) and running specific benchmarks on a variety of AIX and Linux systems.

The results provided were obtained through the following methods: Two types of LDIF files were used. The first type of LDIF file contains small entries with a flat tree structure. The other type of LDIF file contains larger entry sizes as well as a deeper tree structure. LDIF files with 100,000 and 1 million entries were created. A 5-million-entry LDIF file is created for the smaller entries. The first type of LDIF file was used only for search operations. The second type was used for both searches and updates. In each case, a directory server instance is created on the system. The LDIF file is loaded into the database and the benchmark is run. Operating system information and information related to the directory server instance is collected at intervals throughout the load and the benchmark run.

Note: The statistics reported here are specific to the particular hardware setups used and were generated in a lab environment. These results might not be reproducible in other environments. The results reported here should be used only as guidelines.

The LDAP server performance was monitored by running various workloads on different hardware configurations. These workloads were run on a number of AIX and Linux systems.

The systems had the following software installed:

- IBM Tivoli Directory Server 6.0
- DB2 8.2 with fix pack 8

Disk requirements

Because large amounts of data are stored in the directory server, it would be useful to have a formula that determines the capacity of the hard disk that is required to store the complete data. Also useful is a measure of how much CPU, memory, and time is required to load this large amount of data into the directory server from an LDIF file. The two most important factors here are:

- Time required to load the data into the directory server
- Space required to store the data on the hard disk

Bulkload time and space information

Two types of LDIF files were used to gather data statistics. The first is a flat structured LDIF file with small entries. The second is a deeper tree with larger entries. There are no access control lists (ACLs) or groups in either LDIF file.

Example small entry, approximately 415 bytes:

```
dn: cn=Joline Hickey, ou=Accounting, o=IBM.com
objectClass: top
objectClass: person
objectClass: organizationalPerson
cn: Joline Hickey
sn: Hickey
description: Joline_Hickey
facsimileTelephoneNumber: +1 71 631-7308
l: Palo Alto
ou: Accounting
postalAddress: IBM.com$Accounting$Dept # 363$Room # 890
telephoneNumber: +1 408 995-7674
title: Supreme Accounting Mascot
userPassword: yekciHenil
seeAlso: cn=Joline
```

Example large entry, approximately 10,390 bytes:

```
dn: mdsListName=List219, mdsContainerName=container22, mdsUID=22, mdso=393, mdsc=C1,
  dc=IBM, dc=com
objectclass: MDSBlobList
mdsListName: List219
mdsHeadTitle: Listen Titel 9
mdsdata:: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
...
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
```


The attribute “mdata” contains binary data. Binary data is not stored in attribute tables and has a big impact on total disk space and on parsing and loading times.

Time Information

The following graphs and tables show results from running the **bulkload** utility with sizes of 100,000 entries and 1 million entries (for small and large entries), as well as 5 million small entries. The **bulkload** was run in two passes, first the parse and then the load. The information was collected on an AIX 5.3 system with two 1656 MHz POWER5 processors and 3792 MB of RAM. No tuning was done on the database settings before running the **bulkload** utility. For the smaller entry data, an index for the seeAlso attribute was added.

Note: The following results were obtained by using the **bulkload** utility with DB2 v8.1.

Times for bulkload for LDIF files with 100,000 entries

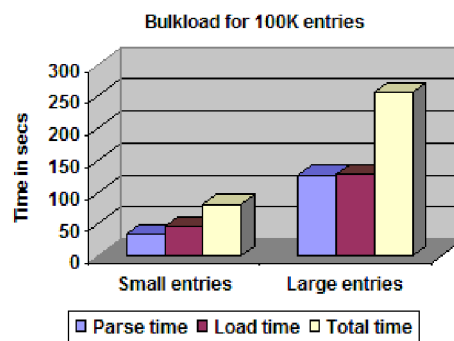


Table 1. Bulkload times for 100,000 entries

Time in seconds	Small entries	Large entries
Parse time	35	127
Load time	47	128
Total time	82	255

Times for bulkload for LDIF files with 1,000,000 entries

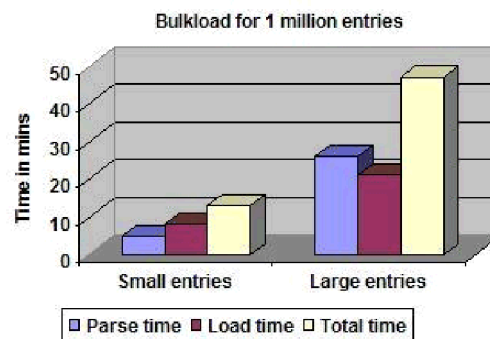


Table 2. Bulkload times for 1,000,000 entries

Time in minutes	Small entries	Large entries
Parse time	5	25
Load time	8	112
Total time	13	137

Times for bulkload for LDIF file with 5,000,000 small entries

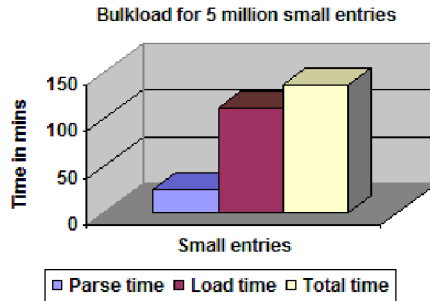


Table 3. Bulkload times for 5,000,000 small entries

Time in seconds	Small entries
Parse time	25
Load time	112
Total time	137

By comparing the **bulkload** times for the same LDIF file on three different systems, the time variations and impact of the hardware can be observed. The following graph and table show the results for three **bulkload** runs with the 1,000,000 small entries LDIF file. The three systems used were:

System1

AIX 5.2 with 4 RS64-IV 752 MHz processors and 6144 MB of RAM

System2

AIX 5.2 with 4 1453 MHz POWER4 processors and 8192 MB of RAM

System3

AIX 5.3 with 2 1656 MHz POWER5 processors and 3792 MB of RAM

Times for bulkload for LDIF file with 1,000,000 small entries

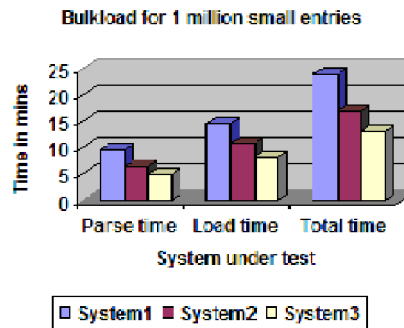


Table 4. Bulkload times for 1,000,000 small entries on three different systems

Time (minutes)	System1	System2	System3
Parse time	9.5	6.2	5
Load time	14.6	10.8	8
Total time	24.1	17	13

The AIX 5.3 system performed the **bulkload** operation in about half the time that it took the AIX 5.2 system for the same operation. The **bulkload** utility is single threaded, so improvements in time come from the CPU speed and disk speed of the hardware being used. This is shown by the improvement on the AIX 5.2 system when run with the faster, POWER4 processors.

In general, the parse time increases linearly as the number of entries in the LDIF file increases. For example, the parse time for 100,000 small entries was 35 seconds and the parse time for 1,000,000 small entries was roughly 10 times that, or 5 minutes. This is not true for the load times, however. Load time does not increase linearly with respect to the data set size.

During the parse phase, intermediate files are generated; these files are used in the load phase. If the input LDIF file is large, more data is written to the intermediate files. The time increase is mostly in disk I/O, and it can be seen in the results obtained for the different LDIF files. The average entry parse time per second is significantly greater for the larger entry size LDIF. For the 100,000 entry LDIF file with smaller sized entries, an average of 2800 entries were parsed per second. For the larger sized entries, however, only 780 entries were parsed per second.

Space information

The space needed in three different filesystem directories was calculated for each of the five LDIF files that were created. The first was the space in the temporary directory used by the parse phase of the **bulkload** operation. This can be referred as the parse size. The second is the space needed in the actual directory where the database is stored. This can be referred as the database size. The third is the space needed to hold a backup of the database. This can be referred as the backup size. The charts displayed below show these statistics for both the small and large entry files, and for 100,000, 1,000,000, and 5,000,000 entries (for small entries only), along with the size of the corresponding LDIF file.

Space used for bulkload for LDIF file with 100,000 entries

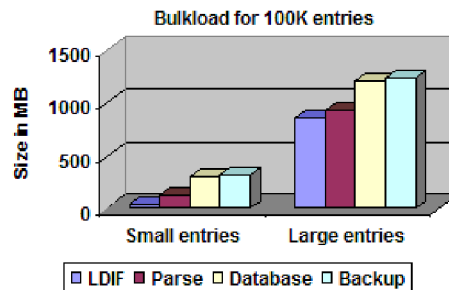


Table 5. Bulkload space for 100,000 entries

Size in MB	Small entries	Large entries
LDIF file size in MB	41	859
Parse space in MB	139	927
Database space in MB	304	1213
Backup space in MB	320	1233

Space used for bulkload for LDIF file with 1,000,000 entries

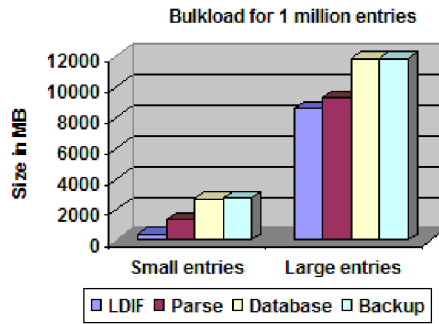


Table 6. Bulkload space for 1,000,000 entries

Size in MB	Small entries	Large entries
LDIF size in MB	407	8597
Parse space in MB	1405	9294
Database space in MB	2779	11864
Backup space in MB	2793	11866

Space used for bulkload for LDIF file with 5,000,000 small entries

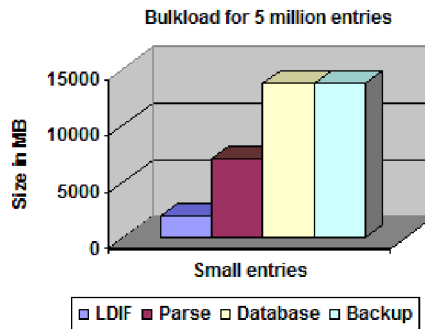


Table 7. Bulkload space for 5,000,000 small entries

Size in MB	Small entries
LDIF size in MB	2033
Parse space in MB	7116
Database space in MB	13780
Backup space in MB	13815

From this data, you can generalize the amount of space needed per entry for the different types of entries. For the smaller entries about 3 KB of space per entry is needed for database storage. For the larger entries, the requirement increases to 12 KB per entry.

The space needed to back up the database is roughly equivalent to the space needed for the database itself. The space needed for the parse phase of the

bulkload is about 3.5 times the size of the LDIF file for the smaller entries (or 1.5 KB per entry), and about 1.2 times the size of the LDIF file for the larger entries (or 9.5 KB per entry).

Memory requirements

When the LDAP server is tuned for performance gain, it is generally done by tuning the size of the LDAP caches or by setting the DB2 bufferpool. However, while doing this the memory capacity of the system must be considered. Allocating a very large amount for the bufferpool or caches or both will result in an increase in the overall memory requirement. Therefore, bufferpool and cache sizes must be allocated carefully. See Chapter 2, “IBM Tivoli Directory Server tuning,” on page 7 and Chapter 3, “Tuning DB2 and LDAP caches,” on page 23 for information.

CPU requirements

To ensure that the CPU is used at its optimum level, here are some guidelines:

- Ensuring that the required data is available in the caches or bufferpool results in low disk accesses, which are generally slower than memory accesses. This decreases the time that the CPU must wait for input/output to occur.
- Enabling simultaneous multithreading (SMT) on systems that are capable of hyperthreading increases the processing capability and the application throughput. See “Simultaneous multithreading” on page 68 for more information.
- Systems with fewer CPUs but greater CPU frequency work more efficiently than systems with more CPUs but lower CPU frequency. For example, a system with 2 processors with 1200 MHz speed performs better than a system with 4 processors with 600 MHz speed.

CPU scaling comparison for throughput (searches and updates)

The processor plays an important role in the overall performance of any application. There are two important factors for a processor that add to the overall performance:

- Number of processors
- Processor speed

Scaling of throughput for varying number of processors

To determine the effect of the number of processors on the overall performance of the directory server, all the hardware configurations were kept common, except for the number of processors. The workloads were run on 3 different AIX 5.3 systems with 1, 2, and 4 POWER V processors (1499 MHz) and 6144 MB RAM.

On each system, the directory server was loaded with 100,000 entries. The same update and search workload was run on all the systems. The update workload consisted of modify and modrdn operations, while the search workload consisted of rootdse searches along with subtree searches. The entry cache size was set to the default value of 25 KB. The results for the search and update workloads with varied number of processors are shown in the following sections.

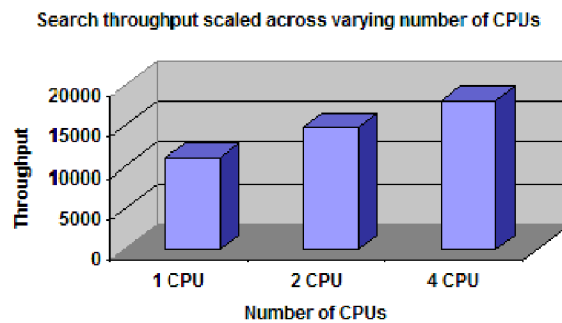
Search throughput: The following table shows the throughput figures for a directory server with 100,000 entries:

Table 8. Search throughput with 1, 2, and 4 CPUs

Number of CPUs	Throughput	CPU User time	CPU system time	CPU idle time	CPU wait time
1 CPU	11129.1	65	35	0	0
2 CPUs	14899.9	59	34	7	0
4 CPUs	18199.5	37	23	40	0

The results for a dual processor configuration are far better than the results for a computer with a single processor. Best results can be obtained using a multiprocessor computer.

The following graph shows the scaling of the search throughput across varying number of processors.

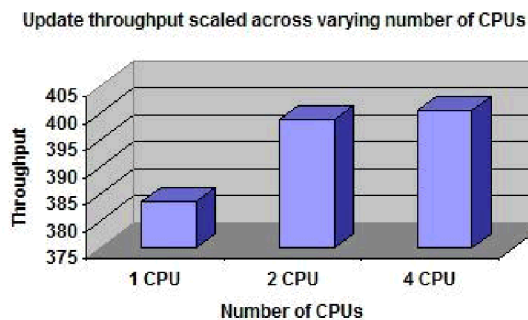


Update throughput: The following table shows the throughput figures for a directory server loaded with 100,000 entries:

Table 9. Update throughput for 1, 2, and 4 CPUs

Number of CPUs	Throughput	CPU User time	CPU system time	CPU idle time	CPU wait time
1 CPU	383.5	65	35	0	0
2 CPUs	398.7	59	34	7	0
3 CPUs	400.4	37	23	40	0

The following graph shows the scaling of the update throughput with varying numbers of processors.



The SMT option can be used to simulate the computer having more processors than are physically present. For more information about SMT, see “Simultaneous multithreading” on page 68.

Scaling of throughput with varying processor speeds

To determine the impact of the CPU speed on the overall performance, the search workload was run on the following systems:

- AIX 5.2 with 4 Power4 752 MHz processors
- AIX 5.2 with 4 Power4 1453 MHz processors

The workload consisted of LDAP search operations for various attributes in varying rates. The parameters that were configured for this workload are:

Table 10. Parameter settings for search workload

Parameter	Bind	UID searches	CN wildcard searches	Exact match on Given name	Exact match on SN	Exact match on CN	Not found
Setting	After every 5 operations	28%	24%; * at the end of the value searched for	16%	8%	16%	8%

The results of running this workload on a directory server with 100,000 and 1,000,000 entries are as follows:

- For a directory with 100,000 entries:

Table 11. Throughput for search for 100,000 entry directory

CPU speed	Throughput	CPU %	Idle %	Wait %
752MHz	1125	93	6	0
1453 MHz	1618	80	19	0

- For a directory with 1,000,000 entries:

Table 12. Throughput for search for 1,000,000 entry directory

CPU speed	Throughput	CPU %	Idle %	Wait %
752MHz	152	100	0	0
1453 MHz	246	97	1	0

The frequency of the second processor is almost double that of the first one. For a directory with 100,000 entries there is an increase in throughput of 43% and for a directory with 1,000,000 entries there is an increase of 61%. This shows that for an increase in the CPU frequency the throughput increases by the same factor as the CPU frequency.

Splitting the database across multiple disks

The tests showed that when the database is split across two different disks instead of residing on one disk, the throughput achieved is doubled. CPU utilization is one important factor that is responsible for this. The following tables show the effect on CPU utilization of splitting the database across two hard disks.

Effect on search throughput of splitting the database across two hard disks

Table 13. Search throughput with one and two hard disks

Database	Throughput	IB	LB	CPU %	Idle %	Wait %	AVM (MB)	Memory (KB)
Normal	260	95	74	23	68	8	191	389044
Split	416	95	74	46	37	16	179	378464

The update workload consists of search and modify operations on a directory with 1 million entries. The computer is a Linux system with 2 Xeon 2400 MHz processors and 4001 MB RAM. The CPU idle % is reduced to half when the database is split across two different disks. The results can be further enhanced by using a high speed hard disk drive to save I/O time as well.

Effect on update throughput of splitting the database across two hard disks

Table 14. Update throughput with one and two hard disks

Database	Throughput	IB	LB	CPU %	Idle %	Wait %	AVM (MB)	Memory (KB)
Normal	118	92	73	23	68	9	496	400736
Split	237	93	72	47	36	16	451	398960

For information about splitting the database across multiple disks, see the redpaper entitled "Performance Tuning for IBM Tivoli Directory Server."

Simultaneous multithreading

Simultaneous multithreading is a processor design that combines hardware multithreading with superscalar processor technology to allow multiple threads to issue instructions each cycle. Unlike other hardware multithreaded architectures in which only a single hardware context (or thread) is active on any given cycle, SMT permits all thread contexts to simultaneously compete for and share processor resources. Unlike conventional superscalar processors, which suffer from a lack of per-thread instruction-level parallelism, simultaneous multithreading uses multiple threads to compensate for low single-thread instruction-level parallelism. Simultaneous multithreading allows multiple threads to run different instructions in the same clock cycle, using the execution units that the first thread left spare. This is done without great changes to the basic processor architecture: the main additions needed are the ability to fetch instructions from multiple threads in a cycle, and a larger register file to hold data from multiple threads. The number of concurrent threads can be decided by the chip designers, but practical restrictions on chip complexity usually limit the number to 2, 4 or sometimes 8 concurrent threads.

The performance consequence is significantly higher instruction throughput and program speedups on a variety of workloads that include commercial databases, web servers and scientific applications in both multiprogrammed and parallel environments.

SMT on AIX FAQs

How would I know if my system is capable of using simultaneous multithreading?)

Your system is capable of SMT if it is a POWER5-based system running AIX 5L Version 5.3.

How would I know if SMT is enabled for my system?

If you run the **smtctl** command without any options, the response tells you if SMT is enabled or not.

Is SMT supported for the 32-bit kernel?

Yes, SMT is supported for both 32-bit and 64-bit kernel.

How do I enable or disable SMT?

You can enable or disable SMT by running the **smtctl** command. The following is the syntax:

```
smtctl [ -m off | on [ -w boot | now]]
```

The following options are available:

-m off Sets SMT mode to disabled.

-m on Sets SMT mode to enabled.

-w boot

Makes the SMT mode change effective on next and subsequent restarts if you run the **bosboot** command before the next system restart.

-w now

Makes the SMT mode change immediately but the change does not persist across restart.

If neither the **-w boot** or the **-w now** options are specified, the mode change is made immediately. It persists across subsequent restarts if you run the **bosboot** command before the next system restart.

Appendix A. Workload description

The tests used in this document contain a mixture of searches and binds, including wildcard searches, which return multiple entries.

Each scenario consists of two phases, a warmup phase and a run phase. During the warmup phase, the searches primarily request entries that are not in the LDAP caches; most of these requests require interaction with the DB2 backing store. For all the measurements reported in this document, warmup consisted of running all queries at least once; consequently, during the run phase all entries requested are potentially already in LDAP caches in memory if the caches are large enough to hold all of them. Thus the warmup phase and the run phase comprise two distinctly different workloads.

During the run phase of the mixed search and bind test, a number of client threads issue search requests to the IBM Tivoli Directory Server from predetermined scripts. The scripts include a number of different kinds of searches, including wildcard and other searches that return multiple entries per request. The client threads run through their scripts continuously for three minutes. Throughput is measured on the server for each three-minute interval, and then each client starts over at the beginning of its script. Each three-minute interval is referred to as a run. The server is not restarted between runs.

Appendix B. Modifying TCP/IP settings

Closed TCP/IP connections between the client and the LDAP server are cleaned at system-specified intervals. In environments where the connections are opened or closed at a high frequency, this can degrade LDAP server performance. To shorten the cleaning intervals, modify the registry keys.

Do the following to modify the registry keys on a Windows platform:

1. Type the following at a command prompt to open Registry Editor:
`regedit`
2. Go to `HKey_Local_Machine\System\CurrentControlSet\Services\Tcpip\Parameters`.
3. Add `TcpTimedWaitDelay` entry (if not already in the registry).
4. Set the `DWORD` value to **1e** for 30 seconds.
5. Add `StrictTimeWaitSeqCheck` entry (if not already in the registry).
6. Set `DWORD Value` to **1**
7. Reboot the machine.

Do the following to modify the TCP/IP settings on an AIX platform:

1. Use the following command
`no -o <attributename>=<value>`

to set the following attributes and values for performance tuning:

tcp_keepidle

Specifies the length of time to keep the connection active. This value is defined in 1/2 second units, and defaults to 14,400 (7200 seconds or 2 hours). **tcp_keepidle** is a runtime attribute.

tcp_keepinit

Sets the initial timeout value for a tcp connection. This value is defined in 1/2 second units, and defaults to 150 (75 seconds). It can be changed to any value with the **-o** option. **tcp_keepinit** is a runtime attribute.

tcp_keepintvl

Specifies the interval between packets sent to validate the connection. This value is defined in 1/2 second units, and defaults to 150 (75 seconds). **tcp_keepintvl** is a runtime attribute.

Note: This applies to both client and server machines.

Appendix C. Platform configurations

The performance tuning examples in this guide use the following platform configurations:

- Clients
 - Four 1.8GHz, 512MB RAM, Intel® PRO/100 VE
 - Windows 2000 Professional with SP2
- Server
 - 4-Way 450MHz, pSeries(TM) eServer(TM) Model 7026-B80 with 1 or 4 processors active, 4 GB RAM.
 - IBM 10/100 Ethernet Adapter.
 - AIX 5.2
 - IBM Tivoli Directory Server Version 6.1
 - AIXTHREAD_SCOPE=S
 - MALLOCTYPE=buckets
 - NODISCLAIM=true (1way).
 - RDBM_CACHE_SIZE=460000 except where noted.
 - RDBM_FCACHE_SIZE=75000 except where noted.
 - RDBM_CACHE_BYPASS_LIMIT=100 except where noted.
 - Necessary Indexes created (for attribute seeAlso).
 - No ACLs were set. By default, anyone can search and compare. The directory administrator can update.
- DB2 v 8.1.1.16
 - maxlocks 100 sortheap 2500 dbheap 5000 ibmdefaultbp 20000 (4K pages)
ldapbp 9800 (32K pages)
 - logfilsiz 2048, logprimary 6
- Miscellaneous
 - Caches were warmed up by running all scripts once.
 - Measurements were taken using 50 client threads except where noted.
 - DB2 log files are not on the same disk as the containers.

Appendix D. Notices

This information was developed for products and services offered in the U.S.A. IBM might not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Department MU5A46
11301 Burnet Road
Austin, TX 78758
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

AIX	OMEGAMON	WebSphere
DB2	OS/400	World Registry
i5/OS	Passport Advantage	xSeries
IBM	pSeries	z/OS
iSeries	SecureWay	zSeries
Lotus	Tivoli	

Adobe, the Adobe logo, PostScript[®], and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft[®], Windows, Windows NT[®], and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.

Index

A

- accessibility vii
- ACL cache
 - description 14
- AIX environment variables
 - AIXTHREAD_SCOPE 40
 - MALLOCTYPE 39
 - NODISCLAIM 40
 - viewing 40
- AIX, enabling large files 39
- AIXTHREAD_SCOPE, setting 40
- alter bufferpool command 26
- APPLHEAPZ 36
- attribute cache
 - adding attributes to
 - using command line 17
 - complex filters resolved by 7
 - configuring 16
 - using command line 17
 - determining attributes for 8
 - language tags 8
 - processing queries 7
 - simple filters resolved by 7

B

- books
 - see publications v, vi
- buffer pools
 - Memory 3
- bulkload 45
 - k option 45

C

- cache configuration variables, setting
 - using command line 18
 - using Web Administration Tool 17
- cache entry sizes, determining 15
- caches, LDAP
 - configuration variables 15
 - description 3
 - directory size 21
 - listed 2
 - tuning to improve performance 7
- change log
 - checking for existence of 57
 - use of 57
- components
 - IBM Tivoli Directory Server 1
- configurations used
 - client 75
 - DB2 75
 - miscellaneous 75
 - server 75
- conventions
 - typeface viii

D

- DB2 buffer pools 2
 - and directory size 21
 - determining best size for 24
- IBMDEFAULTBP 2
- LDAPBP 2
 - setting sizes 26
 - tuning considerations 24
 - tuning overview 23
- DB2 configuration parameters
 - determining current settings 36
 - setting 36
- DB2 tuning
 - backup command 37
 - buffer pools 24
 - database organization 30
 - optimization and organization overview 29
 - optimizing
 - overview 29
 - using command line 30
 - using Configuration Tool 29
 - restore command 37
- directory names, notation viii
- directory size
 - measuring effect on performance 19
 - size of DB2 buffer pools 21
 - size of LDAP caches 21
- disk speed, improving 43

E

- education
 - see Tivoli technical training vii
- entry cache
 - description 13
 - determining best size for 13
 - determining entry size 15
- environment variables, notation viii

F

- filter cache
 - determining best size for 11
 - determining entry size 15
 - processing queries 11
 - size with updates 11
- filter cache bypass limit, determining best 12

I

- IBM Tivoli Directory Server
 - components 1
- IBM Tivoli Directory Server features
 - bulkload 45
 - change log 57
 - monitoring performance 51
 - Proxy server 50

- IBM Tivoli Directory Server features
 - (continued)
 - replication 47
- IBMDEFAULTBP
 - description 2
 - determining best size for 25
- improving disk speed 43
- indexes, DB2 35

L

- large files, enabling on AIX 39
- LDAP attribute cache 7
- LDAP caches 2
 - Memory 3
- LDAP filter cache 11
- LDAPBP
 - description 2
 - determining best size for 25
- ldapsearch
 - "cn=changelog,cn=monitor" 56
 - "cn=connections,cn=monitor" 56
 - "cn=monitor" 51
 - "cn=workers,cn=monitor" 56
- LOGFILSIZ 36

M

- MALLOCTYPE, setting 39
- manuals
 - see publications v, vi
- monitoring performance 51

N

- NODISCLAIM, setting 40
- notation
 - environment variables viii
 - path names viii
 - typeface viii

O

- online publications
 - accessing vi
- ordering publications vii

P

- path names, notation viii
- performance, monitoring 51
- Proxy server 50
- publications v
 - accessing online vi
 - ordering vii

R

- reorg command 33
- reorgchk command 30
- replication 47
 - context cache size 48
 - number of replication threads 48
 - ready size limit 49
- RUNSTATS command 30

S

- settings
 - ibm-slapdIdleTimeOut 18
 - ibm-slapdMaxEventsPerConnection 18
 - ibm-slapdMaxEventsTotal 18
 - ibm-slapdMaxNumOfTransactions 18
 - ibm-slapdMaxOpPerTransaction 18
 - ibm-slapdMaxTimeLimitOfTransactions 18
 - ibm-slapdPagedResAllowNonAdmin 18
 - ibm-slapdPagedResLmt 18
 - ibm-slapdSizeLimit 18
 - ibm-slapdSortKeyLimit 18
 - ibm-slapdSortSrchAllowNonAdmin 18
 - ibm-slapdTimeLimit 18

T

- TCP/IP settings
 - modifying 73
- tips for improving performance
 - disk speed 43
 - generic 4
- Tivoli software information center vi
- Tivoli technical training vii
- training, Tivoli technical vii
- tuning
 - LDAP 1
 - overview 1
 - tuning, DB2 3
 - tuning, LDAP
 - overview 3
 - tips 4
- typeface conventions viii

V

- variables, notation for viii

W

- Workload description 71



Printed in USA

SC23-6540-00

