SUN BLUEPRINTS APPROVED
[illegible]

Directory Server Indexing

By Tom Bialaski - Enterprise Engineering

Sun BluePrints™ OnLine - November 2000

Sun microsystems
ENTERPRISE ENGINEERING

http://www.sun.com/blueprints

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303 USA
650 960-1300 fax 650 969-9131

Part No.: 806-7230-10
Revision 01, November 2000

Directory Server Indexing

As part of the setup and configuration of the Solaris™ Directory Extensions for the iPlanet™ Directory Server (iPlanet Directory Server) to support the native Lightweight Directory Access Protocol (LDAP) implementation in the Solaris™ 8 Operating Environment (Solaris OE), performance tuning adjustments are recommended as best practices. This article discusses some of those tuning options with explanations about what they do and why they are important.

Directory Tuning Overview

Tuning a directory is much like tuning a database. That is, you want to create a database cache large enough to prevent repeated disk access and you want to create indexes for frequently accessed data searches. You can determine the need for larger database caches when your server starts experiencing excessive paging or swapping. However, the effect of a poorly indexed directory is not always obvious. Often what you will experience is subtle like longer than expected delays during searches and update operations.

The key to proper indexing is knowing what type of searches are most likely to occur. If you are unsure of what search types are most frequent, you can examine the directory access log to find out which attributes are being accessed on a regular basis after the directory is placed into production. However, it is a better practice to create the indexes *before* the server is placed into production instead of waiting for performance problems to occur. Since the type of searches the Solaris OE naming service is likely to perform can be anticipated, appropriate indexes can and should be created in advance.

The attributes listed in this article are good candidates for indexing. While you could just follow the recommended procedures provided in this article, it is worthwhile to understand what you are indexing and why, since additional fine-tuning may be required. Therefore, both procedures and explanations are provided.

Closely related to attribute indexing is Virtual List View (VLV) indexing, which is a required feature of any directory server that will be configured to support Solaris OE LDAP clients. The VLV is also referred to as a *browse* index and facilitates browsing through a large number of directory entries. Since the mechanism for creating VLVs is quite different than the procedure for creating attribute indexing, they are discussed separately.

The Search Algorithm

Before you can appreciate the role indexing plays, you need to understand how the directory server performs searching. Search requests are sent by an LDAP client, which specifies where to start looking in the directory, a search filter, and an optional list of attribute values to retrieve. For example, a search might start at the container `ou=People` with a search filter of `uid=`*username* and a requested attribute value of `userpassword`. When the server receives the request, the first thing it does is check to see if an index exists which matches the search filter, for example `uid=`. If an index that matches `uid=` exists, then a list of candidate entries are retrieved. In this case, the attribute, `uid` is unique within the directory, so only one entry is retrieved.

If an index that matches the search filter does not exist, then the directory server searches all the entries in the database, examining each one to see if it meets the search criteria. This obviously requires much more work and can be a lengthy process. The process can take even longer if multiple search filters are specified and those searches are not indexed.

Not only is it necessary to know which attributes are likely to be searched, it is important to know what type of searches will be performed. The next section explains what those types are and the relative resource cost of deploying them. While you may be tempted to simply index every attribute with every type of indexing, you need to be aware of what the effect of resource consumption will be.

Types of Indexes

Indexes are created on a per attribute basis and are stored in a file bearing the attribute's name, for example, `uid.db`. This file can contain multiple types of indexes which search for the following:

- Presence Index (**pres**)—Looks for all entries that contain a certain attribute.
- Equality Index (**eq**)—Requires an exact match on an attribute value.

- Approximate Index (**approx**)—Uses the common name (`cn=`) attribute for sound-like searches.
- Substring Index (**sub**)—Specifies only a portion of a desired string.

The **pres** and **eq** types are the most common since you may often want to know which entries contain a particular attribute and which attributes contain a particular value. The **approx** index type is not used by the Solaris OE LDAP client but is used in applications like address books. The **sub** index type is used by the Solaris OE LDAP client to help speed up searches that only specify a portion of a domain name. For example, you could search for the domain `east`, instead of specifying the full name of `east.sun.com`. TABLE 1 lists which attributes should be indexed and what their index types are.

TABLE 1 Solaris Indexed Attributes and Types

Attribute	pres	eq	sub	approx
`uid`	X	X		
`uidnumber`	X	X		
`gidnumber`	X	X		
`ipHostNumber`	X	X		
`ipNetworkNumber`	X	X		
`ipProtocolNumber`	X	X		
`macAddress`	X	X		
`oncRpcNumber`	X	X		
`ipServiceProtocol`	X	X		
`ipServicePort`	X	X		
`nisDomain`	X	X		
`nisMapName`	X	X		
`mail`	X	X		
`memberuid`	X	X		
`membernisnetgroup`	X	X	X	
`nisnetgrouptriple`	X	X	X	

The number of indexed attributes may seem excessive, but it doesn't cost you much in terms of resources to maintain them. The Section, "Cost of Indexing," on page 7, discusses the costs associated with maintaining indexes.

How Indexes are Created

Indexes are files that get created in the *install_dir*/`slapd-`*instance*/`db` directory, or in the directory where the `db` directory has been relocated to. As mentioned earlier, these files can contain multiple types of indexing. Several system and default indexes get created automatically when you install the iPlanet Directory Server software.

The system indexes are:

- `aci.db`
- `changenumber.db`
- `copiedfrom.db`
- `dncomp.db`
- `objectclass.db`
- `entrydn.db`
- `parentid,db`
- `numsubordinates.db`

The default indexes include:

- `cn.db`
- `givenName.db`
- `sn.db2`
- `member.db2`
- `telephoneNumber.db2`
- `mailHost.db`
- `owner.db`

Please note that the `*.db2` files do not get created until an attribute is specified in an entry. If you set up indexes before you import NIS data, you will not see these files initially. Once the first entry is imported, the files will appear.

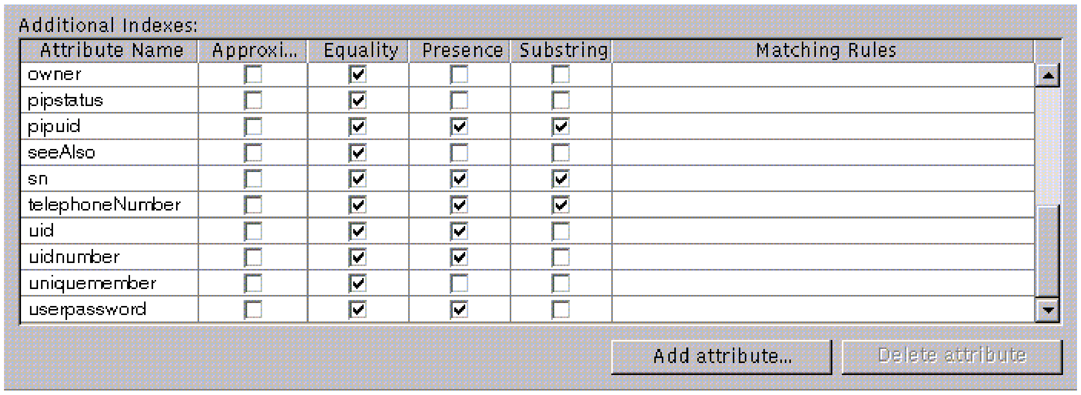
Indexes can be created in two ways:

1. From the Directory Console
2. From the command line

While the iPlanet Directory Console is the easiest method, the command line method is useful for creating automated scripts. Both methods are discussed in the following sections.

Creating Indexes From the iPlanet Directory Console

The easiest way to create indexes is through the Directory Console. You must be logged in as `Directory Manager` to create indexes. Once logged in, go to the **Configuration** tab and highlight **Database** in the left pane. In the right pane, under the **Indexes** tab you will see a list of attributes and their types as shown here:



Additional Indexes:

Attribute Name	Approxi...	Equality	Presence	Substring	Matching Rules
owner	☐	✓	☐	☐	
pipstatus	☐	✓	☐	☐	
pipuid	☐	✓	✓	✓	
seeAlso	☐	✓	☐	☐	
sn	☐	✓	✓	✓	
telephoneNumber	☐	✓	✓	✓	
uid	☐	✓	✓	☐	
uidnumber	☐	✓	✓	☐	
uniquemember	☐	✓	☐	☐	
userpassword	☐	✓	✓	☐	

Click the **Add attribute...** button to choose the attribute you want to index. Make sure you have already added the Solaris OE naming service attributes to the `slapd.user_at.conf` file or else they will not appear. After you choose an attribute, check the type(s) of indexing you want to create. After you click the **Save** button, the index is created.

The Directory Console automatically performs a two-step process. The first step is to add a line to the database configuration file `slapd.ldbm.conf` followed by the creation of the index files. After the configuration file is modified, restart the server so the changes will take effect. If there are only a few changes, or no entries containing the indexed attributes, the process will happen very quickly.

The same steps performed in the Directory Console can also be done through the command line as described in the next section.

Creating Indexes From the Command Line

While the iPlanet Directory Console may be easier to use for a novice, adding indexing from the command line has certain advantages. You can create scripts to automatically set up indexing which is repeatable and reuseable. The following steps are required to create indexes from the command line.

1. **Edit the** `slapd.ldbm.conf` **file.**

   Add the new lines specifying attributes and indexing types as shown in the following example, then restart the server.

```
root# vi slapd.ldbm.conf
.....
index pipuid pres,eq,sub
index seeAlso eq
index sn pres,eq,sub
index telephoneNumber pres,eq,sub
index uid pres,eq
index uidnumber pres,eq
index uniquemember eq
index userpassword pres,eq
....
root# stop-slapd
root# start-slapd
```

2. **Run the** `ns-slapd` **command with the** `db2index` **option.**

   You can run the `ns-slapd db2index` command either with the server running or stopped. If you issue the command with the server running you need to place it in read-only mode. Indexes are created faster if the server is stopped, so this may be preferable.

```
root# ns-slapd db2index -f install-dir/slapd-instance \ /config/
slapd.conf -t uid:eq,pres,sub
root#
```

Note – When setting up the Solaris OE LDAP naming service, it is best to create your indexes before any data is loaded. If there is a large amount of entries containing the attributes you are indexing, then running `db2index` will be slow, especially if the server is running.

Once indexes are set up, they will automatically be updated as entries are added or deleted. However, adding or deleting entries which contain indexed attributes will take longer as described in the next section.

Cost of Indexing

Indexing is an effective way to substantially increase search speeds, but it does come at a cost. There are two types of cost. One is the additional memory the index takes up and the other is the increase in the amount of time it takes to update entries. The size of your index cache can easily reach or exceed the size of your entry cache and updates can take an order of magnitude longer if excessive indexing is deployed.

Each entry in an equality index consists of an attribute value and the entry ID associated with the entry containing that attribute. As more equality index attributes are added, the larger the total index cache becomes. However, since you are only indexing some of the attributes, the effect is lessened. If you already use equality indexing, then the added overhead of presence indexing is minimal. Substring indexing stores several permutations of each string value and can get quite large if the strings are long, thus using more memory.

The second cost associated with indexing is the write or update speed. Every time an entry with an indexed value is added, modified, or deleted, the index file needs to be regenerated. If an entry has multiple indexed values, then multiple files need to be updated. Also, if substring indexing is used the system needs to calculate all the possible substring values, which is very compute intensive. Abnormally long delete times can occur if the entries being removed have indexed values, since the index files will need to be regenerated.

Fortunately, Solaris OE naming service data does not get updated frequently. Once the directory is populated with your current NIS data, updates should be minimal. In most cases, the increased search speed outweighs the disadvantage of longer write speeds.

Virtual List Views

A Virtual List View (VLV), or browsing list, is used to increase the efficiency of displaying large numbers of entries. Instead of sending a client unsorted entries, the server presorts the entries before they are sent. This makes it easier for the client to page through the returned entries. The iPlanet Directory Console makes use of VLVs for displaying containers which have several hundred or even thousands of entries in them. The Solaris OE LDAP client also makes use of VLVs for displaying large amounts of information such as all the user account entries on the server.

When used with the Directory Console, the VLV can be created easily from the GUI. However, the Solaris LDAP client has different requirements, so the VLVs must be created from the command line by importing an LDAP Data Interchange Format (LDIF) file. To understand the syntax of the required LDIF, it helps to know a little about how VLVs are implemented in the iPlanet Directory Server which is discussed next.

VLV Implementation

The VLV mechanism is implemented as a control in the iPlanet Directory Server. Clients wishing to use this mechanism do so by accessing the VLV control. To do this, the client must have permission to access it.

Note – By default, the VLV control does not grant anonymous access. Since the Solaris OE LDAP client requires an anonymous bind during initialization, the ACI for the VLV control must be modified.

Through the VLV control, the client has access to two object classes: `vlvSearch` and `vlvIndex`. The `vlvSearch` object class specifies the result set of the search and the `vlvIndex` object class specifies how the returned result set is sorted. This is done through the `vlvSort` attribute.

Creating a VLV requires a pair of entries which include the `vlvSearch` and `vlvIndex` object classes. The `vlvSearch` entry includes a search base and the `vlvFilter` attribute which specifies the object class that contains the attribute(s) you intend to sort. The `vlvIndex` object class includes the `vlvSort` attribute which specifies one or more attributes to sort and in which order to sort them (a minus “-” sign denotes reverse order).

The following is an example of the LDIF which creates the `getpwent` VLV index that is used by the Solaris OE LDAP client when retrieving a list of user login IDs or common names. The entries are always stored in the directory information tree (DIT) under `cn=config,cn=ldbm` and must each be given a unique name. The `vlvBase` is an attribute which reflects where your naming service data is stored and varies depending on your configuration.

```
dn: cn=getpwent,cn=config,cn=ldbm
objectclass: top
objectclass: vlvSearch
cn: getpwent
vlvBase: ou=people,dc=blueprints,dc=com
vlvScope: 1
vlvFilter: (objectclass=posixAccount)
aci: (target="ldap:///
cn=getpwent,cn=config,cn=ldbm")(targetattr="*)
(version 3.0; acl
"Config";allow(read,search,compare)userdn="ldap:///anyone";)

dn: cn=getpwent,cn=getpwent,cn=config,cn=ldbm
cn: getpwent
vlvSort: cn uid
objectclass: top
objectclass: vlvIndex
```

Note – The distinguished name (DN) given to the `vlvIndex` entry is confusing since it also contains the component `cn=getpwent`. However, this is the naming convention chosen and it must be adhered to.

TABLE 2 lists the other VLV indexes recommended along with the changes to the search base, filter, and attributes to be sorted on.

TABLE 2 Solaris OE LDAP Client VLV Indexes

VLV Name	Search Base:	Filter on:	Sort on:
`getpwent`	`ou=people`	`posixAccount`	`cn.uid`
`getgrent`	`ou=group`	`posixGroup`	`cn,gidNumber`
`gethostent`	`ou=hosts`	`ipHost`	`cn,ipHostNumber`
`getnetent`	`ou=networks`	`ipNetwork`	`cn,ipNetworkNumber`
`getspent`	`ou=people`	`posixAccount`	`cn,uidNumber`

After you create an LDIF file with the information provided in TABLE 2, you need to import it into your directory server, it assumes the LDIF file name of `vlv.ldif` (see example here).

```
# ldapmodify -a -D “cn=Directory Manager” -w password -f vlv.ldif
```

Once the VLV indexes are added to the directory, you still need to enable them before they can be used. To do this, run the `vlvindex` command as shown here.

```
# install-dir/slapd-instance/vlvindex getpwent
# install-dir/slapd-instance/vlvindex getgrent
# install-dir/slapd-instance/vlvindex gethostent
# install-dir/slapd-instance/vlvindex getnetent
# install-dir/slapd-instance/vlvindex getspent
```

Since it is always a good practice to verify that the VLV indexes have been defined, run the `ldapsearch` command and observe the `vlvsearch` lines as shown here.

```
# ldapsearch -s base -b "" objectclass=\*
. . .
vlvsearch: cn=getspent,cn=config,cn=ldbm
vlvsearch: cn=getpwent,cn=config,cn=ldbm
vlvsearch: cn=getnetent,cn=config,cn=ldbm
vlvsearch: cn=gethostent,cn=config,cn=ldbm
vlvsearch: cn=getgrent,cn=config,cn=ldbm
. . .
```

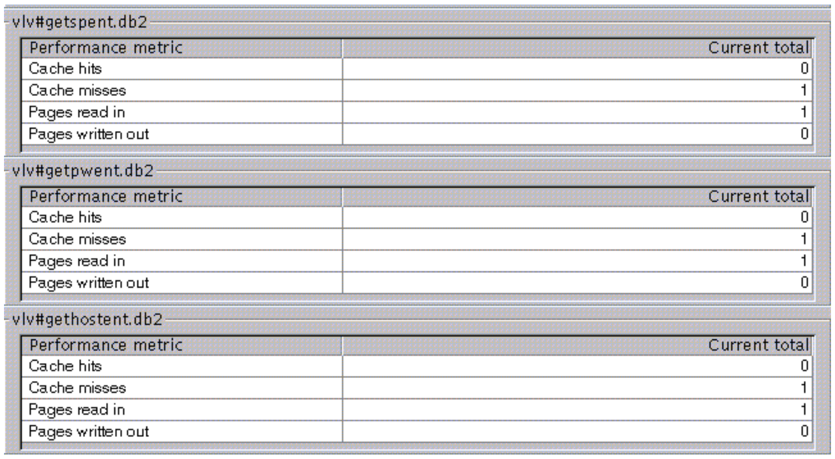
You should also observe the creation of the new `db` files.

```
# cd install-dir slapd-instance/db
# ls -l | grep vlv
-rw------- 1 nobody nobody 16384 Sep  8 10:07 vlv#gethostent.db2
-rw------- 1 nobody nobody 16384 Sep  8 10:07 vlv#getgrent.db2
-rw------- 1 nobody nobody 16384 Sep  8 10:07 vlv#getnetent.db2
-rw------- 1 nobody nobody 16384 Sep  8 10:07 vlv#getpwent.db2
-rw------- 1 nobody nobody 16384 Sep  8 10:07 vlv#getspent.db2
#
```

Viewing Index Activity

The iPlanet Directory Console provides a convenient way to monitor the database cache activity. Cumulative statistics are displayed which show how often the indexes have been accessed and what the cache hit ratio is. Once the caches are full, you should see close to a 100% hit ratio with no pages being written out. If you do not, this could be an indication that the database cache is too small and should be enlarged. Little or no activity on an index is an indication that the index is not being used and it may be a candidate for removal.

You can view the following screen by going to the **Status—>Performance Counters—>Database** tab in the Directory Console. Only a portion of the screen is shown here and it displays three of the VLV indexes that were created. Since this was taken just after the VLV indexes were created, there is no current activity.



vlv#getspent.db2

Performance metric	Current total
Cache hits	0
Cache misses	1
Pages read in	1
Pages written out	0

vlv#getpwent.db2

Performance metric	Current total
Cache hits	0
Cache misses	1
Pages read in	1
Pages written out	0

vlv#gethostent.db2

Performance metric	Current total
Cache hits	0
Cache misses	1
Pages read in	1
Pages written out	0

Conclusion

Indexing plays an important role in optimizing the performance of your directory server. Both types of indexing discussed in this article, attribute and VLV, should be deployed when configuring a directory server to support the native LDAP naming service which is included in the Solaris 8 OE.

After indexes are created you should always verify that they were configured correctly by examining the DIT and verifying that the associated index files were created. Since it is not always obvious that a poorly indexed server is the root cause of a performance problem, careful monitoring is also recommended.

Author's Bio: Tom Bialaski

Tom Bialaski is currently a Staff Engineer with the Enterprise Engineering group at Sun Microsystems, and is the author of "Solaris Guide for Windows NT Administrators." and co-author of "Solaris and LDAP Naming Services". Tom has nearly 20 years of experience with the UNIX® operating system and has been a Sun Engineer since 1984.