Advanced Information System Queries: ldapsearch

Last review date	Reviewer
2009-09-15	Marco Bencivenni Enrico Fattibene

Table of Contents

Advanced Information System Queries: ldapsearch

This page explains how to use the ldapsearch command to perform complex queries of the EGEE information system.

It is assumed that the reader is familiar with the GLUE schema and the general structure of the information system. An introduction to this area can be found in the page "Querying the Information System".

Useful references:

- A general introduction to LDAP (IBM Red Book)
- The definition of the LDIF format (RFC 2849)
- The definition of an LDAP search filter (RFC 2254)
- The OpenLDAP web site
- An LDAP module for perl
- A python LDAP API
- An LDAP interface for PHP
- The GLUE schema specification (schema version 1.3)
- The GLUE LDAP schema in the EGEE CVS repository

LDAP Overview

LDAP is a protocol through which information can be queried. An LDAP server supports this protocol and answers queries using locally stored data. LDAP is not described in detail here; see the references above for more details. However, in general terms information is organised in a set of objects, each with one or more objectclasses (types). Objectclasses generally represent a concept in a data model, similarly to tables in a relational database although the details are somewhat different. A specific object instance is then analogous to a row in a table.

Each objectclass is defined to have a set of named attributes, some of which may be optional, and which may be either single- valued (one such attribute per object) or multi- valued. Attribute names have a global scope, so different objectclasses cannot use the same attribute name for different things. Attributes also have types, typically string or integer althought they can be more complex. Strings can be defined as either case- sensitive or case- insensitive.

The objects are organised into a hierarchical tree, such that each object is identified by a unique *Distinguished Name* (DN) of the form `attribute1=value1,attribute2=value2,attribute3=value3,...` with the root object name to the right and the leaf object to the left, so that each object apart from the root is a child (or component) of some other object. This is known as the *Directory Information Tree* (DIT). Each object has a specific attribute which is used to construct the DN. The value of this attribute need not be unique, since objects with the same value can be distinguished by their position in the tree.

Note that the DIT is constructed from actual objects, so the existence of an object requires the existence of its parent objects. For example, if an object has a DN of Name=Fred,Department=hr,Site=London then there must be objects with attribute values Department=hr and Site=London. However, the converse is not necessarily true.

The attributes and objectclasses, together with the way they are linked to form a DIT, are defined in a *schema* which is stored in the LDAP server. The server configuration also defines the port on which it listens for queries, and a DN to act as the root of the DIT.

In general LDAP is optimised to be able to serve large query volumes efficiently. However, for any query which is likely to be repeated frequently it may be worth giving some thought to minimising the number of queries, the number of objects searched by the query, and the amount of data returned.

The EGEE information system

The EGEE Grid uses an information schema known as the*GLUE schema*, currently at version 1.3. This is defined in an abstract way, and mappings exist for several different technologies including LDAP. The actual LDAP schema used in EGEE is referenced above, although the format is not especially easy to read. At present the schema used in EGEE defines all strings to be insensitive, although this may change in the future.

The DIT for the GLUE schema is relatively shallow, with the primary objects (e.g. `GlueCE`, `GlueSE`, `GlueService`,`GlueSite`) grouped below a single object for each site, and a layer of subsidiary objects below them, e.g. `GlueSA` below the corresponding `GlueSE`. For most purposes the DN of an object is not very important as the schema defines `UniqueID`attributes which are used to link objects together in a technology- independent way. However, it cay be useful to use the DN to restrict the scope of a query to a subtree if efficiency is important.

The LDAP mapping for GLUE defines two extra attributes to help with queries. A `GlueForeignKey` contains a reference to the `UniqueID` of a different object, e.g.`GlueForeignKey: GlueSEUniqueID=castorsrm.cern.ch`. Similarly a `GlueChunkKey` contains a reference to an object which is a parent in the LDAP tree, e.g. a `GluseSA` object might contain`GlueChunkKey: GlueSEUniqueID=castorsrm.cern.ch`.

The LDAP server in EGEE is known as a *BDII* (Berkely Database Information Index), which relates to the technology used to implement it. However, as far as queries go it is just a standard LDAP server. BDIIs are organised in a hierarchy, with so- called *top- level*BDIIs giving a view of the entire Grid, aggregating the content of site- level BDIIs which publish the data relevant to their site. These in turn aggregate the content of resource- level BDIIs running on each Grid service node, which collect information about the services on the node by running programs known as *information providers*.

The ldapsearch command

The main command line tool to query an LDAP server is`ldapsearch`. The general structure of the command is:

```
ldapsearch -x -LLL -h <bdii-host> -p <bdii-port> -b mds-vo-name=local,o=grid \
               [<filter>] [attribute1 [attribute2] ...]
```

or the equivalent:

```
ldapsearch -x -LLL -H ldap://<bdii-host>:<bdii-port>/ -b mds-vo-name=local,o=grid \
               [<filter>] [attribute1 [attribute2] ...]
```

Going through the parameters in order:

- `-x`: turns off SASL authentication, which is not used with the BDII.
- `-LLL`: removes some optional and largely useless elements in the output, although specifying this option is not essential as the output will usually be post- processed in any case. `-L` and`-LL` can also be used for intermediate levels of output reduction.

- `-h`: specifies the BDII host. Queries should generally be made to a top-level BDII to get a view of the entire Grid, unless you are only interested in a specific site. If you don't know a suitable BDII you can use `lcg-bdii.cern.ch`, or whatever is set in the `LCG_GFAL_INFOSYS` environment variable on your UI. Both top-level and site-level BDIIs are published in the information system itself as GlueService objects, so if you know one top-level BDII you can find all the others.

- `-p`: specifies the port. This is normally 2170, although older systems may still have a server on port 2135 if you query a CE or SE directly.

- `-b`: specifies/ restricts the branch of the LDAP DIT in which the query is made (the *query base*). BDII trees are rooted at `o=grid`, and you can usually use just that for a query to a top-level BDII. However, for compatibility with the old MDS system the first node under that is called `mds-vo-name=local`, and it is somewhat safer to include that as well, especially in scripts. At the site level there may be other branches under `o=grid`, e.g. `mds-vo-name=<site-name>` or `mds-vo-name=resource`. You can also specify a query base further down the tree, e.g. to restrict the query to a specific CE or SE.

The filter expression defines the query, and is described in more detail below. It defaults to `objectclass=*`, i.e. to query for all objects. Note that you may need to use quotes with expressions containing wildcards to avoid shell expansion. The query selects all objects which match the filter expression.

The final part of the command is a list of attributes to return for the selected objects; if this is omitted all attributes are returned. In practice it is often more convenient to return everything and use a tool like `grep` to select the attributes to print.

One other option which is sometimes useful is `-s one`, which restricts the query to one level of objects below the base DN, rather than searching the entire subtree.

There are various other options, but these are not usually needed for Grid queries. See the `ldapsearch` man page for full details.

ldapsearch output

The output from `ldapsearch` is sent to `stdout` as *LDIF* (LDAP Interchange Format), see the LDIF man page or the RFC referenced above for details. The format is reasonably readable, but not very flexible. In particular, lines longer than 78 characters are split, with the following line starting with a space. In many cases it is necessary to post-process the output to make it more readable; this can often be done with standard Unix pipeline tools like `grep`, `awk`, `cut`, `sort` etc. One useful trick is that piping the LDIF through `perl -p00e 's/\r?\n //g'` will undo the line splitting mentioned above (lines starting with a space are joined to the previous line).

LDAP APIs

For more complex applications it may be more useful to use an API rather than a command-line tool, but these are not described here. For the C API on which the command-line tool is based, see the man page for `ldap_search(3)`. There are also APIs available for other languages, e.g. perl and python (see the references above). For an example of use of the perl API, have a look at the source code for the `lcg-infosites` and `lcg-info` tools, which should be installed on a standard UI.

LDAP queries

The filter expression used to define the query has a somewhat idiosyncratic syntax, but reasonably complex queries can be constructed with practice, although the flexibility is much less than with a language like SQL. The formal definition of the filter expression format can be found in RFC 2254, referenced above, but the following description covers the things which are generally needed in practice.

Basically the filter consists of a set of expressions in parentheses which can be combined via & (AND), | (OR) and ! (NOT) using a prefix notation, i.e. if A, B and C are valid expressions then `(&ABC)`, `(|ABC)` and `(!A)` are also valid. The simplest expression is an equality test, `(a=b)`, where `a` is an attribute name and `b` is a value.

For example, `(&(a=b)(c=d))` means that attributes `a` and `c` must have values `b` and `d` respectively to give a match. Similarly, `(|(!(a=b))(c=d))` matches if attribute `c` has the value `d` or attribute `a` does not have the value `b`. It is not always obvious when parentheses are needed; in general it's best to use them everywhere.

Objectclasses can be matched using an expression like `(objectclass=GlueCE)`. Components of DNs can be matched with an expression like `(GlueSEUniqueID:dn:=castorsrm.cern.ch)` - in other words, this matches any object with a DN containing `GlueSEUniqueID=castorsrm.cern.ch`, as opposed to `(GlueSEUniqueID=castorsrm.cern.ch)` which matches an object containing that attribute value. The `*` character can be used as a generic wildcard - but not in a query on a DN component. Numeric comparisons can use >= and <= (only, i.e. there is no < or >).

It is not possible to do relational queries, for example selecting the`GlueSE` objects with `UniqueID`s which are referred to in those `GlueSA` objects which have a`GlueSAAccessControlBaseRule` attribute matching a given VO. In such cases you have to do multiple queries.

Cookbook

The following examples give an idea of what can be achieved, building from simple cases to more complex ones. Usually the best way to develop something is to build the query up gradually so you can check that each stage does the right thing. The more complex examples are somewhat baroque and might be better done using an API from a programming language, but they do illustrate what can be achieved just with pipelines on the command line.

Note that the examples are not necessarily the best way of achieving a given result, they are just used to illustrate various strategies. For one- off queries it may well not matter exactly how you do it as long as you get the information you want, but for queries which will be done regularly it may be worth taking some time to find a way which is reasonably efficient and robust.

Getting the Unique ID for all CEs

This is an elementary query. It would work without any filter as the attribute name only appears in the GlueCE object, but would be somewhat less efficient. Similarly the attribute name could be omitted as it gets selected anyway with grep - this is done to remove DNs and blank lines from the LDIF output. An extra step could be to use `cut -d: -f2` to strip off the attribute name.

```
ldapsearch -x -LLL -H ldap://lcg-bdii.cern.ch:2170/ -b mds-vo-name=local,o=grid \
    objectclass=GlueCE GlueCEUniqueID | grep GlueCEUniqueID:
```

```
[...]
GlueCEUniqueID: gridce.iihe.ac.be:2119/jobmanager-pbs-betest
GlueCEUniqueID: lcg002.ihep.ac.cn:2119/jobmanager-lcgpbs-euchina
GlueCEUniqueID: ce114.cern.ch:2119/jobmanager-lcglsf-grid_2nd_lhcb
GlueCEUniqueID: polgrid1.in2p3.fr:2119/jobmanager-pbs-grif
GlueCEUniqueID: ce103.cern.ch:2119/jobmanager-lcglsf-grid_geant4
GlueCEUniqueID: gridgate.cs.tcd.ie:2119/jobmanager-pbs-solovo
[...]
```

Finding out which CE job managers are in use at UK sites

The query here is slightly more complex, in that it uses a wildcard filter to select only CEs with hostnames in UK domains. There is also a bit more post- processing to present the result in a useful form. This also illustrates the lack of case- sensitivity in the filter. However, remember that `grep` is case- sensitive by default.

```
ldapsearch -x -LLL -H ldap://lcg-bdii.cern.ch:2170/ -b mds-vo-name=local,o=grid \
    '(&(objectclass=gluece)(gluece infohostname=*.ac.uk))' \
    glueceinfojobmanager | grep GlueCEInfoJobManager: \
    | cut -d: -f2 | sort | uniq -c
```

```
     11  lcgcondor
    313  lcgpbs
      3  lcgsge
     21  pbs
     16  sge
```

How many SEs support ATLAS?

This uses the fact that the `GlueSA` object, which contains the VO assignment information, has an attribute `GlueChunkKey` to provide a link to the parent SE. There could potentially be several matching SAs per SE, but the "`sort | uniq`" ensures that only one is counted. The access control format is expanding, but this gets all possible variants currently in use.

For variety, this query uses a different BDII, the alternative format to specify the host and port, and queries from `o=grid`. It also doesn't explicitly select an attribute name or use the `-LLL` option as the output gets filtered by the `grep` anyway. The objectclass is omitted as the attribute filter implies it.

```
ldapsearch -x -h lcgbdii02.gridpp.rl.ac.uk -p 2170 -b o=grid \
    '(|(GlueSAAccessControlBaseRule=atlas)(GlueSAAccessControlBaseRule=VO:atlas)\
    (GlueSAAccessControlBaseRule=VOMS:/atlas/*))' \
    | grep 'GlueChunkKey: GlueSEUniqueID=' | sort | uniq | wc -l
```

```
218
```

Get all CE information for a specific VO

This kind of query is used by the WMS to collect all information about sites to be used in matchmaking, and illustrates a more complex use of AND and OR conditions. The `wc -c` just counts the number of bytes.

```
ldapsearch -x -h lcgbdii02.gridpp.rl.ac.uk -p 2170 -b o=grid \
    '(|(objectclass=GlueSubCluster)(&(|(objectclass=GlueVOView)(objectclass=GlueCE)) \
    (|(GlueCEAccessControlBaseRule=VO:cms)(GlueCEAccessControlBaseRule=VOMS:/cms/*))))' \
    | wc -c
```

```
2523721
```

Which sites have SEs which support the gsidcap protocol?

This example illustrates a double query: first find the`GlueSEAccessProtocol` objects which match a condition, extract the `GlueChunkKey` which contains a reference to the parent`GlueSE` object, and then query that for the`GlueForeignKey` which contains the site name. The post- processing here just strips out the attribute name and removes any duplication.

One wrinkle is the use of a wildcard to match the GlueSEUniqueID. In general this is not needed. However, if the value is long enough it will get wrapped onto another line so the value extracted may be truncated; the wildcard ensures that it will still match, although in a sufficiently pathological case it might match other SEs too.

Efficiency is more of an issue here because the second query is executed multiple times (once per matching SE). Adding an explicit restriction`objectclass=GlueSE` reduces the time for the query by around a factor 2, from 20 seconds to 10.

```
for i in `ldapsearch -x -H ldap://lcg-bdii.cern.ch:2170 -b o=grid \
    GlueSEAccessProtocolType=gsidcap | grep GlueChunkKey: | cut -d= -f2 \
    | sort | uniq`; do ldapsearch -x -H ldap://lcg-bdii.cern.ch:2170 \
    -b o=grid "GlueSEUniqueID=$i*" GlueForeignKey; done \
    | grep GlueSiteUniqueID | cut -d= -f2 | sort | uniq
```

```
ALBERTA-LCG2
BEgrid-ULB-VUB
BEIJING-LCG2
BG01-IPP
BG04-ACAD
BNL-LCG2
CIEMAT-LCG2
DESY-HH
[...]
```

Here is another way to get the same result, using perl to join any split lines and sed to process the DN and use the result in the second query. This is somewhat more efficient (it takes about 7 seconds), but harder to write.

```
for dn in `ldapsearch -h lcg-bdii.cern.ch -p 2170 -x -b o=grid \
    GlueSEAccessProtocolType=gsidcap | perl -p00e 's/\r?\n //g' \
    | sed -ne 's/^dn: [^,]*,[^,]*,\(.*\)/\1/p' | sort -u`; do \
    ldapsearch -h lcg-bdii.cern.ch -p 2170 -x -b $dn \
    '(objectClass=GlueSite)' GlueSiteUniqueID \
    | perl -p00e 's/\r?\n //g' | sed -ne 's/^GlueSiteUniqueID: \(.*\)/\1/p'; done
```

```
ALBERTA-LCG2
BEgrid-ULB-VUB
BEIJING-LCG2
BG01-IPP
BG04-ACAD
BNL-LCG2
CIEMAT-LCG2
CSC
CSCS-LCG2
DESY-HH
[...]
```

Which UK sites support each VO?

This example first queries for the `AccessControlBaseRule`entry, basically the VO name, for all CEs with a UK domain name, and processes this to get a list of all supported VOs. One point to note is that this does not cope with the new VOMS FQAN format for ACBRs as it stands, but in practice it's unlikely that any CEs support only a subset of a VO. There is then an outer loop over the VO list, which prints the VO name and then queries for the CE hostnames, and sorts and prints them.

This is not especially elegant or efficient because it uses multiple queries simply to format the output. For serious use it would be better to use an API, store the output of a single query and format it programmatically. However, for ad- hoc queries this sort of thing gives a useful result with relatively little effort.

As an exercise for the reader, consider how to invert this to print a list of supported VOs for each CE.

```
for VO in `ldapsearch -x -h lcgbdii02.gridpp.rl.ac.uk -p 2170 -b o=grid \
    "(&(objectclass=GlueCE)(GlueCEUniqueID=*.ac.uk*))" \
    | grep "GlueCEAccessControlBaseRule: VO:" | sort | uniq | cut -d: -f3` ; \
    do echo; echo $VO:; echo; \
    ldapsearch -x -h lcgbdii02.gridpp.rl.ac.uk -p 2170 -b o=grid \
    "(&(GlueCEUniqueID=*.ac.uk*)(GlueCEAccessControlBaseRule=VO:$VO))" \
    GlueCEInfoHostName | grep GlueCEInfoHostName: | sort | uniq \
    | cut -d: -f2; done
```

```
[...]

esr:

 ce.epcc.ed.ac.uk
 fal-pygrid-18.lancs.ac.uk
 helmsley.dur.scotgrid.ac.uk
 hepgrid2.ph.liv.ac.uk
 heplnx206.pp.rl.ac.uk
 heplnx207.pp.rl.ac.uk
 lcgce02.gridpp.rl.ac.uk
 t2ce02.physics.ox.ac.uk
 t2ce03.physics.ox.ac.uk

euindia:

 serv03.hep.phy.cam.ac.uk

fusion:
```

```
 ce.epcc.ed.ac.uk
 fal-pygrid-18.lancs.ac.uk
 hepgrid2.ph.liv.ac.uk
 heplnx206.pp.rl.ac.uk
 heplnx207.pp.rl.ac.uk
 lcgce02.gridpp.rl.ac.uk
 t2ce02.physics.ox.ac.uk
 t2ce03.physics.ox.ac.uk

[...]
```

How many CPUs does each site have in nodes that run SL4?

This is getting to the point of being too complex for a command- line query, but does illustrate what can be done if you really try. The problem here is that the OS information is in the `GlueSubCluster` object, which is linked to a `GlueCluster` object, which is linked to `GlueCE` objects which contain the CPU counts, so you need to do at least two queries. The first one returns the `GlueChunkKey` (containing the cluster ID) for all subclusters supporting SL4, which itself requires a condition on two different attributes (with some allowance for varying formats in the way the values are published).

The Cluster IDs are then carved out with `cut` and piped into another query with `xargs`, to find `GlueCE` objects with those IDs in their `GlueForeignKey` attributes. The query returns the hostname and CPU count, which are then processed to remove the attribute names and join them onto the same line.

The final "`sort | uniq`" tries to allow for the fact that CEs (queues) publishing the same number of CPUs are probably feeding jobs to the same underlying set of nodes. This is not totally robust, but this is a feature of the schema and information providers and can't be fixed at the query level.

```
ldapsearch -x -h lcg-bdii.cern.ch -p 2170 -b o=grid \
    '(&(objectclass=gluesubcluster)(gluehostoperatingsystemname=scien*)\
    (gluehostoperatingsystemrelease=4*))' \
    | grep GlueChunkKey: | cut -d= -f2 | xargs -n1 -i \
    ldapsearch -x -h lcg-bdii.cern.ch -p 2170 -b o=grid \
    '(&(objectclass=gluece)(glueforeignkey=glueclusteruniqueid={}))' \
    GlueCEInfoHostName GlueCEInfoTotalCPUs | grep -A 1 GlueCEInfoHostName: \
    | grep : | cut -d: -f 2 | xargs -l2 echo | sort | uniq
```

```
[...]
ce.grid.hepg.sdu.edu.cn 8
ce.grid.tuke.sk 16
ce.grid.vgtu.lt 20
ce-iep-grid.saske.sk 57
ceitep.itep.ru 66
ce.phy.bg.ac.yu 49
ce.reef.man.poznan.pl 352
ce-test.pic.es 4
ce.ui.savba.sk 41
ce.ulakbim.gov.tr 65
cluster50.knu.ac.kr 276
cluster.pnpi.nw.ru 184
[...]
```