Configure DataMorph

Database size issues

Some databases can grow so much that it is very hard to store them on fast storage, let alone fit to RAM. This is where OpenLDAP's `back-mdb` comes in handy, showing that it is possible to reduce the DB footprint compared to other backends, often close to a factor of two when compared to `back-hdb/bdb`. But even that might not be enough and admins will once more start looking whether there is another way to squeeze more space out of a database.

This article will discuss a way to improve the storage efficiency in some circumstances and tries to explain how entries are actually stored in the database as well.

Data types

Whenever there is a larger number of data used by string attributes with a very limited number of distinct, but long values, that is usually an obvious candidate to see whether the schema and applications using it can be adapted to use something more compact. However, in some cases that is not possible.

Since just about everything in LDAP is handled and stored as text, if there is a large number of integers stored that will always fit within a known range, the string representation can incur some space overhead as well.

We now present an OpenLDAP overlay that allows the two cases above be handled while preserving the same string values as far as the clients are concerned.

Datamorph overlay

This overlay can be used to handle all mentions of configured attributes and while maintaining the presentation to the outside world that they are still string attributes, in the underlying database they are stored as binary strings encoding the value in a short, even single byte representation.

Similar to the techniques ASN.1 PER uses to compact enumerated values, it will convert the names to a number as configured, stored directly as a single byte representing that number.

It can also handle numeric attributes of various sizes, signed or unsigned and store them as big-endian machine integers.

All basic LDAP operations are handled, including search filter handling, although the overlay will refuse to save an entry with one of the configured attributes in its relative DN.

Configuration

The variant overlay is available in Symas OpenLDAP builds from version **FIXME** onwards.

1. Before you can use it, instruct the server to load the module, this will load some new syntaxes into OpenLDAP that you will use for the attributes.

```
$ ldapmodify -x -D cn=config -W -H ldap://localhost
dn: cn=module{0},cn=config
changetype: modify
add: olcModuleLoad
olcModuleLoad: datamorph.la
```

2. You can then define new attributes in the schema according to your needs. An example of a signed integer attribute looks like this:

```
( <OID for the attribute>
  NAME 'signed'
  DESC 'Signed integer attribute'
  EQUALITY fixedSizeSignedIntegerMatch
  ORDERING fixedSizeSignedIntegerOrderingMatch
  SYNTAX 1.3.6.1.4.1.4203.666.11.12.1.4 )
```

   To define an unsigned integer you would add a similar definition:

```
( <OID for the attribute>
  NAME 'number'
  DESC 'Integer attribute'
  EQUALITY fixedSizeIntegerMatch
  ORDERING fixedSizeIntegerOrderingMatch
  SYNTAX 1.3.6.1.4.1.4203.666.11.12.1.3 )
```

   An example of an enumerated attribute looks like this:

```
( <OID for the attribute>
  NAME 'enumerated'
  DESC 'Enumerated attribute'
  EQUALITY fixedSizeIntegerMatch
  SYNTAX 1.3.6.1.4.1.4203.666.11.12.1.2 )
```

3. Now add the overlay to the database you intend to use it with. If you replicate that database, you probably want this overlay to be configured on each server.

```
dn: olcOverlay=datamorph,$DATABASE
changetype: add
objectClass: olcDataMorphConfig
```

4. With the overlay in place, you can configure how each attribute should be handled. For integer attributes you want to do something like this

```
dn: olcDatamorphAttribute=signedInteger,olcOverlay={x}datamorph,$DATABASE
objectclass: olcDatamorphInteger
olcDatamorphIntegerBytes: 2
olcDatamorphIntegerSigned: TRUE
olcDatamorphIntegerLowerBound: -20000
olcDatamorphIntegerUpperBound: 30000
```

   For an enumerated attribute, you will have to specify which values map to what in the database, like this:

```
# to handle attribute 'enumeratedAttribute'
dn: olcDatamorphAttribute=enumerated,olcOverlay={x}datamorph,$DATABASE
objectClass: olcDatamorphEnum

# value 'value1' corresponds to 'AQ==' (0x01)
dn: olcDatamorphValue=value1,olcDatamorphAttribute={0}enumerated,olcOv
 erlay={x}datamorph,$DATABASE
objectclass: olcDatamorphEnumValue
olcDatamorphIndex: 1

# value 'value11' corresponds to 'Cw==' (0x0B)
dn: olcDatamorphValue=value11,olcDatamorphAttribute={0}enumerated,olcO
 verlay={x}datamorph,$DATABASE
objectclass: olcDatamorphEnumValue
```

olcDatamorphIndex: 11

Permissible `olcDatamorphIndex` values are numbers between `0` and `255` inclusive.

Showtime

With the configuration done, the server is ready to store the values in a more efficient manner and you can point your clients to it. However if you try exporting the database with `slapcat`, you will notice that the values for those attributes is quite different from what you see with `ldapsearch`. This is due to the values being translated by the overlay and since no overlays are in operation when using any of the `slap*` tools, you will see the "raw" value as stored in the database. This is something to keep in mind when loading data into the database in bulk using `slapadd`, you will have to either use an export produced by `slapcat` or generate the values as they should exist in the database. For example, take the following attributes:

```
enumerated: value11
signed: -19858
```

When passing them over to be loaded by `slapadd` to a server configured as in the examples above, they should be written out as:

```
enumerated:: Cw==
signed:: sm4=
```

Expected space savings

As well as providing a more compact representation of the values, especially for attributes with enumerated values, the space saving achieved by this overlay can be higher or lower than could be explained by this fact alone. To explain why, we will have to take a look at one aspect of how `back-mdb` works. `liblmdb` allocates space in terms of pages, usually 4 KiB in size. If an entry does not fit within that space, it will occupy as many pages as necessary to hold it and that space is not shared with anything else (these are called *overflow pages*). If an entry fits within 4 KiB, `liblmdb` will consider sharing the page with other entries. This is a major factor in space efficiency and should be considered first. If most entries occupy 70-80 % of a page, it would take a major improvement in the storage efficiency to allow 2 or more entries to fit in the same page, on the other hand if entries regularly overshoot the page size by a few bytes, it might not take much effort to make sure they do, which is where the space savings could easily accumulate (overflow pages require contiguous area to be allocated for them, leading to fragmentation and higher disk usage).

When storing an entry, `back-mdb` will do a good job of only storing what is necessary for fast and efficient operation. To quote the documentation to `mdb_entry_encode`, the function handling this task:

> Flatten an Entry into a buffer. The buffer starts with the count of the number of attributes in the entry, the total number of values in the entry, and the e_ocflags. It then contains a list of integers for each attribute. For each attribute the first integer gives the index of the matching AttributeDescription, followed by the number of values in the attribute. If the high bit of the attr index is set, the attribute's values are already sorted. If the high bit of numvals is set, the attribute also has normalized values present. (Note - a_numvals is an unsigned int, so this means it's possible to receive an attribute that we can't encode due to size overflow. In practice, this should not be an issue.) Then the length of each value is listed. If there are normalized values, their lengths come next. This continues for each attribute. After all of the lengths for the last attribute, the actual

values are copied, with a NUL terminator after each value. The buffer is padded to the sizeof(ID). The entire buffer size is precomputed so that a single malloc can be performed.

Note that if there are attributes that need to be normalized for matching and indexing, the normalized version of each value gets stored as well, which helps performance as no normalization needs to be performed when retrieving the entry.
In the case of attributes handled by the Datamorph overlay, there is only one possible encoding of each value and whenever the attribute is processed, translation between that encoded value and what is exposed over the wire is always performed. This is why the overlay never stores the normalized values as there is no performance benefit in maintaining it. Additionally, if substantial space is used by these attributes, omitting that will help further reduce the size of the database, especially in the case of MDB.
To get a rough estimate how much space might be saved, you can check the total size of all affected values in your database at the moment, if you double that, you will have an approximation of how much space is being used to store them right now. For enumerated fields, the new space would be around 4-6 bytes per value, for integer fields, adjust this to match the configured size.
The actual space savings will depend heavily on whether this allows `back-mdb` to reduce the number of overflow pages used for huge entries and whether smaller entries can fit in the same page.

Complete cn=config Example

```
dn: cn=module{0},cn=config
changetype: modify
add: olcModuleLoad
olcModuleLoad: datamorph.la

dn: cn=datamorph,cn=schema,cn=config
changetype: add
objectClass: olcSchemaConfig
olcAttributeTypes: ( 1.3.6.1.4.1.4203.666.11.12.123.1
  NAME 'enumerated'
  DESC 'Enumerated attribute'
  EQUALITY fixedSizeIntegerMatch
  ORDERING fixedSizeIntegerOrderingMatch
  SYNTAX 1.3.6.1.4.1.4203.666.11.12.1.2 )
olcAttributeTypes: ( 1.3.6.1.4.1.4203.666.11.12.123.2
  NAME 'number'
  DESC 'Integer attribute'
  EQUALITY fixedSizeIntegerMatch
  ORDERING fixedSizeIntegerOrderingMatch
  SYNTAX 1.3.6.1.4.1.4203.666.11.12.1.3 )
olcAttributeTypes: ( 1.3.6.1.4.1.4203.666.11.12.123.3
  NAME 'signed'
  DESC 'Signed integer attribute'
  EQUALITY fixedSizeSignedIntegerMatch
  ORDERING fixedSizeSignedIntegerOrderingMatch
  SYNTAX 1.3.6.1.4.1.4203.666.11.12.1.4 )
olcObjectClasses: ( 1.3.6.1.4.1.4203.666.11.12.123.4
```

```
  NAME 'transformedObject'
  DESC 'Testing objectclass'
  SUP top AUXILIARY
  MAY ( enumerated $ number $ signed ) )

dn: olcOverlay={0}datamorph,olcDatabase={1}mdb,cn=config
changetype: add
objectClass: olcOverlayConfig
objectclass: olcDatamorphConfig

# a basic enum
dn:
olcDatamorphAttribute={0}enumerated,olcOverlay={0}datamorph,olcDatabase={1}mdb
,cn=config
changetype: add
objectclass: olcDatamorphEnum

dn:
olcDatamorphValue=bjensen,olcDatamorphAttribute={0}enumerated,olcOverlay={0}da
tamorph,olcDatabase={1}mdb,cn=config
changetype: add
objectclass: olcDatamorphEnumValue
olcDatamorphIndex: 1

dn:
olcDatamorphValue=bjorn,olcDatamorphAttribute={0}enumerated,olcOverlay={0}data
morph,olcDatabase={1}mdb,cn=config
changetype: add
objectclass: olcDatamorphEnumValue
olcDatamorphIndex: 11

dn:
olcDatamorphValue=dots,olcDatamorphAttribute={0}enumerated,olcOverlay={0}datam
orph,olcDatabase={1}mdb,cn=config
changetype: add
objectclass: olcDatamorphEnumValue
olcDatamorphIndex: 12

dn:
olcDatamorphValue=jaj,olcDatamorphAttribute={0}enumerated,olcOverlay={0}datamo
rph,olcDatabase={1}mdb,cn=config
changetype: add
objectclass: olcDatamorphEnumValue
olcDatamorphIndex: 13

dn:
olcDatamorphValue=jjones,olcDatamorphAttribute={0}enumerated,olcOverlay={0}dat
amorph,olcDatabase={1}mdb,cn=config
changetype: add
objectclass: olcDatamorphEnumValue
olcDatamorphIndex: 14
```

```
dn:
olcDatamorphValue=jdoe,olcDatamorphAttribute={0}enumerated,olcOverlay={0}datam
orph,olcDatabase={1}mdb,cn=config
changetype: add
objectclass: olcDatamorphEnumValue
olcDatamorphIndex: 10

dn:
olcDatamorphValue=jen,olcDatamorphAttribute={0}enumerated,olcOverlay={0}datamo
rph,olcDatabase={1}mdb,cn=config
changetype: add
objectclass: olcDatamorphEnumValue
olcDatamorphIndex: 101

dn:
olcDatamorphValue=johnd,olcDatamorphAttribute={0}enumerated,olcOverlay={0}data
morph,olcDatabase={1}mdb,cn=config
changetype: add
objectclass: olcDatamorphEnumValue
olcDatamorphIndex: 20

dn:
olcDatamorphValue=melliot,olcDatamorphAttribute={0}enumerated,olcOverlay={0}da
tamorph,olcDatabase={1}mdb,cn=config
changetype: add
objectclass: olcDatamorphEnumValue
olcDatamorphIndex: 51

dn:
olcDatamorphValue=uham,olcDatamorphAttribute={0}enumerated,olcOverlay={0}datam
orph,olcDatabase={1}mdb,cn=config
changetype: add
objectclass: olcDatamorphEnumValue
olcDatamorphIndex: 31

dn:
olcDatamorphAttribute=signed,olcOverlay={0}datamorph,olcDatabase={1}mdb,cn=con
fig
changetype: add
objectclass: olcDatamorphInteger
olcDatamorphIntegerBytes: 2
olcDatamorphIntegerSigned: TRUE
olcDatamorphIntegerLowerBound: -20000
olcDatamorphIntegerUpperBound: 30000

dn:
olcDatamorphAttribute=number,olcOverlay={0}datamorph,olcDatabase={1}mdb,cn=con
fig
changetype: add
objectclass: olcDatamorphInteger
olcDatamorphIntegerBytes: 1
olcDatamorphIntegerUpperBound: 1
olcDatamorphIntegerSigned: FALSE
```

Complete slapd.conf Example

```
include             ./schema/core.schema
include             ./schema/cosine.schema
include             ./schema/inetorgperson.schema
include             ./schema/openldap.schema
include             ./schema/nis.schema

pidfile      slapd.pid
argsfile     slapd.args

moduleload  back_mdb.la
moduleload  datamorph.la

database    monitor

database    mdb
suffix      "dc=example,dc=com"
rootdn      "cn=Manager,dc=example,dc=com"
rootpw      secret
directory   ./db
index       objectClass     eq
index       cn,sn,uid       pres,eq,sub

overlay     datamorph

attributetype ( 1.3.6.1.4.1.4203.666.11.12.123.1 NAME 'enumerated'
    DESC 'Enumerated attribute'
    EQUALITY fixedSizeIntegerMatch
    ORDERING fixedSizeIntegerOrderingMatch
    SYNTAX 1.3.6.1.4.1.4203.666.11.12.1.2 )

attributetype ( 1.3.6.1.4.1.4203.666.11.12.123.2 NAME 'number'
    DESC 'Integer attribute'
    EQUALITY fixedSizeIntegerMatch
    ORDERING fixedSizeIntegerOrderingMatch
    SYNTAX 1.3.6.1.4.1.4203.666.11.12.1.3 )

attributetype ( 1.3.6.1.4.1.4203.666.11.12.123.3 NAME 'signed'
    DESC 'Signed integer attribute'
    EQUALITY fixedSizeSignedIntegerMatch
    ORDERING fixedSizeSignedIntegerOrderingMatch
    SYNTAX 1.3.6.1.4.1.4203.666.11.12.1.4 )

objectclass ( 1.3.6.1.4.1.4203.666.11.12.123.4 NAME 'transformedObject'
    DESC 'Testing objectclass'
    SUP top AUXILIARY
    MAY ( enumerated $ number $ signed ) )

datamorph enum enumerated
datamorph_value 1 bjensen
datamorph_value 11 bjorn
```

```
datamorph_value 12 dots
datamorph_value "13" jaj
datamorph_value 14 jjones
datamorph_value 10 jdoe
datamorph_value 101 jen
datamorph_value 20 johnd
datamorph_value 51 "melliot"
datamorph_value 31 uham

datamorph int signed
datamorph_size 2
datamorph_signed TRUE
datamorph_lower_bound -20000
datamorph_upper_bound 30000

datamorph int number
datamorph_size 1
datamorph_signed no
datamorph_upper_bound 1
```