MRuby Backend for OpenLDAP

Open Source Solution Technology Corporation
HAMANO Tsukasa <hamano@osstech.co.jp>
LDAPCon 2017 Brussels October 2017

Abstract

We often receive requests that require us to use various data storage (Cloud Database and NoSQL) as user store and allow the frontend to speak in LDAP protocol. OpenLDAP is designed to allow you to develop flexibly these backend modules you want, but it requires a lot of development resources and time efforts. To develop OpenLDAP backends rapidly, we already have back-perl, but the development with it is still not practical due to the fact that it has only one interpreter although OpenLDAP is a multi-threaded application. This paper introduces back-mruby that is a new OpenLDAP backend that makes backend development easy for everyone. Reasons why we should use Ruby rather than Perl or Python are also explained.

1 Arrival of MRuby

There are many implementations of the Ruby language. The most popular and official implementation is CRuby. MRuby is a new implementation that is compatible with the Ruby 1.9 specification. MRuby has small footprint compared with conventional Ruby implementations and works on small devices well. MRuby is also assumed to be embedded into applications rather than executing as stand alone application.

Matz, the creator of CRuby, started MRuby project in 2010. This project is supported by METI (ministry of the Government of Japan). Currently, MRuby is being developed on github.com and is distributed under the MIT License.[*1]

2 Limitation of back-perl

The Perl backend (back-perl) already exists in OpenLDAP. It enables us to develop flexible OpenLDAP backends rapidly by implementing LDAP request handlers in Perl language. back-perl is better suited for integrating with high-level datastores rather than low-level databases such as BDB and LMDB. It can perform to append LDAP protocol interface to various datastore models such as RDBMS, NoSQL and cloud datastore services.

However, the back-perl has a limitation that the slapd process has only one Perl interpreter. Theoretically multiple Perl interpreters can be created in a process, but the API of interpreter operation is not thread-safe so these cannot be executed in parallel. Therefore, back-perl is not practical because it only executes one Perl handler at a time although slapd is multi-threading.

[*1] https://github.com/mruby/mruby

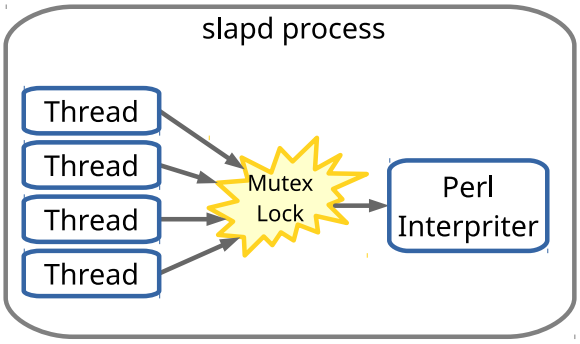

Figure1　back-perl process model

At first, We tried to replace Perl interpreter of back-perl with Python and CRuby, but they have the same issue. Python allows us to create multiple sub-interpreters in a process, but it needs to acquire GIL (global interpreter lock) when python objects are in operation.

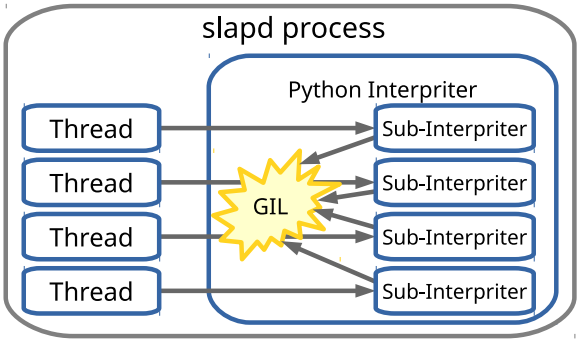

Figure2　back-python process model

3　New backend: back-mruby

Secondly, We have started to implement the MRuby backend for OpenLDAP. We can create MRuby virtual machines as many as we want within a single process. MRuby solves the issue around locking. Memory spaces of these VMs are completely independent and do not affect each other. In back-mruby, slapd assigns MRuby virtual machine for each thread. These OpenLDAP threads are thus able to execute Ruby code in parallel effectively.

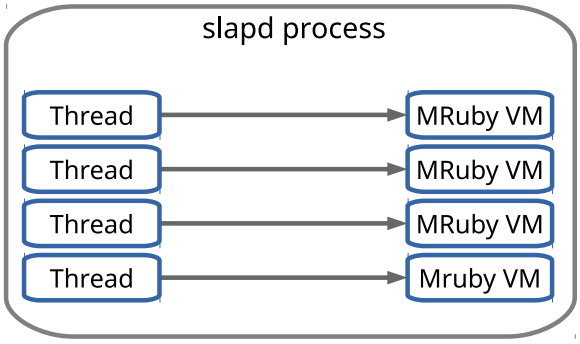

Figure3　back-mruby process model

4　Usage

4.1　Configuration

The configuration of back-mruby is very simple. It just requires to set rubyfile parameter to slapd.conf as below. This makes slapd load the Ruby script file in which LDAP request handlers are defined.

In slapd.conf:

```
rubyfile example.rb
```

4.2　BIND Example

The LDAP request handler as ruby script can be defined as follows. The first example shows how to implement BIND request handler that successfully authenticates with a probability of $\frac{1}{2}$.

```ruby
def bind(op)
  if rand(2) == 0
    LDAP_SUCCESS
  else
    LDAP_INVALID_CREDENTIALS
  end
end
```

The next example shows how to implement BIND request handler with static user entries. The data of LDAP request are passed as the op argument to which the op structure of slapd is simply mapped. For example, a normalized DN is available with op['ndn'].

```
def bind(op)
  # User DN and password mappings
  users = {
    'uid=alice,dc=example,dc=com' =>
      'secret1',
    'uid=bob,dc=example,dc=com' =>
      'secret2',
  }
  ndn = op['ndn']
  cred = users[ndn]
  if cred == op['req']['cred']
    LDAP_SUCCESS
  else
    LDAP_INVALID_CREDENTIALS
  end
end
```

4.3 SEARCH Example

The Search request handler function should return a list of LDIF entries or a response code. The following example shows how to implement SEARCH request handler that processes requests which scope is base or one.

```
def search(op)
  scope = op["req"]["scope"]
  if scope == LDAP_SCOPE_BASE
    search_base(op)
  elsif scope == LDAP_SCOPE_ONE
    search_one(op)
  else
    LDAP_NO_SUCH_OBJECT
  end
end
```

```
# Returns base entry.
def search_base(op)
  ldif = <<END_OF_LDIF
dn: dc=example,dc=com
objectClass: top
objectClass: dcObject
objectClass: organization
dc: example
o: example
END_OF_LDIF
  [ ldif ]
end

# Returns two user entries.
def search_one(op)
  entries = []
  ldif = <<END_OF_LDIF
dn: uid=%s,dc=example,dc=com
objectClass: top
objectClass: account
uid: %s
END_OF_LDIF
  entries << ldif % (["alice"] * 2)
  entries << ldif % (["bob"] * 2)
  entries
end
```

5 Future tasks

- back-mruby creates and destroys one VM for each LDAP request. VM creation is fast enough, but reusing VM may increase efficiency.
- We are currently developing back-mruby on github.com[*2]. I will request to merge official OpenLDAP repository in the future.

[*2] https://github.com/osstech-jp/openldap/tree/mruby