DataTAG
Advance Reservation WP2
Database Management in GARA

Test Specification

Network reservation in Gara
With database
MySQL/MyODBC

Chiara Curti-Emanuela Bertelli

Test Specification 1

1 Introduction 3

2 Module Tests 3

3 Integration Tests 6
3.1 Testing the setup flow script 6
3.2 Testing the Network Resource Manager 8

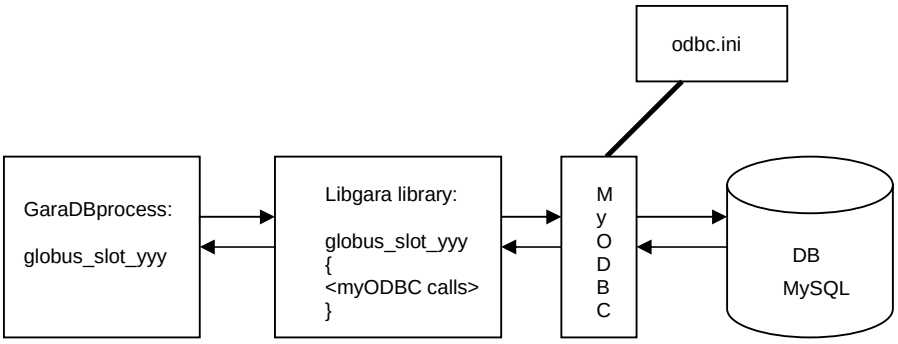
5 Conclusions 10

1 Introduction

This document would like to describe the tests that have been performed after the database insertion for network reservation in Gara. At first a list of tests done with the new library in the stand alone case is proposed. The library is called libgara_slot_db_manager.la and it is linked by network diffserver manager in order to manage the database instead of the previous file support. It can be linked by an independent test program and can be tested without involving the Gara resource manager system. This test phase is called Module Test Phase because it tests only the new slot_db_manager module. After that the tests performed at CNAF with Gara and Globus are described. They constitute the integration phase that concerns the database managed by network resource manager of Gara and the consequenting router configuration.

2 Module Tests

The following tests concern the use of the library libgara_slot_db_manager.la which contains the interfaces of type globus_slot_yyy necessary to manage the data stored in the database.
To test the single interface contained in the mentioned library a program called garaDBprocess has been used.
In the picture below the plant built to perform the module tests is shown.



The garaDBprocess executable runs and asks the user to write in the command line the number which corresponds to the interface that one wants to test. The selected procedure is invoked with the parameters that have been set previously in the garaDBprocess code.
The output is a line in the shell where the test runs where the error code is displayed and depending on the interface type also some other parameter can be shown.
If no error occurs the return code is 0 and the test output is a line of type:
“globus_slot_yyy called with result 0”.

At first the environment variables listed below have to be set in the same command shell where the tests will run. More details are available in the Installation&Configuration Manual.

1) export DBUSER=root
   → the connection to the database is performed by the superuser 'root'

2) export DBHOST=datatag4
   → put here the name of the machine where the MySQL server runs

3) export DBNAME=gara
   → 'gara' is the database name

4) export LD_LIBRARY_PATH=/usr/lib:/usr/local/lib:$LD_LIBRARY_PATH
   → insert here the path of MySQL libraries (/usr/lib) and of MyODBC libraries (/usr/local/lib). The location of the mentioned libraries depends on the type of the installation that has been performed.

5) export ODBCINI=/…/tmp/instdb/etc/odbc.ini
   → modify the path to the odbc.ini file. The path will depend on the location where gara code has been downloaded.

6) export ODBCSYSINI=/…/tmp/instdb/etc
   → similar to 5). In performed tests this variable was not necessary but it was suggested by the installation guide retrieved on the MySQL web site.

Moreover it is also necessary the modification of the odbc.ini.in file located under the path_to_gara_location/gara/slot_db_manager/etc directory where path_to_gara_location is supposed to be the path to reach the directory where Gara has been installed.
For example some changes are mandatory if the database is called in a different way instead of 'gara' or if the MyODBC libraries are not installed under / MYODBC_INSTALL_PATH/lib.
The changes have to be compatible with the environment variables set in the shell where the tests run (see the list above).
Before the modifications become valid, compiling and installing steps have to be re-applied as described at the beginning of the paragraph 1.7 of the Installation&Configuration Manual.

The following steps are required in order to begin the tests.

1) To start the MySQL server give the command on the shell where the environment variables have been already set:

   > mysqld -u root → 'mysqld' is under /usr/sbin

   Verify that the mysqld daemon is alive through the ps –ef command.
2) Run the script dbgarastart in order to create the database (the script is provided with Gara);
3) Run the MySQL client with the following command:

   > mysql –u root –h datatag4 → with –h specify the hostname where the server runs.

   > use gara; → specify the name of the database to use.

> show tables; → verify that the tables RESOURCE, SLOT and DIFFSERV have been created

> show columns from RESOURCE; → to verify the fields of this table. Do the same for the other two tables

> select * from RESOURCE; → verify that this table is empty. Do the same for the other two tables

4) Launch the executable garaDBprocess.
At first connect to the DB by typing 1 on the command line when the test program has started.
Then the test program asks the user to choose the following action to perform.
It writes on the shell "Choose what to do:". By typing a number, the correspondent procedure is tested.
Before running the garaDBprocess test program, see the correspondent garaDBprocess.c file in order to see the values that have to be stored into the database.
After each action, check on the database client shell if the expected behaviour is performed (for example after a globus_slot_add action a new slot has to be present in table SLOT and after a globus_slot_bind the table DIFFSERV must contain a new row).
At the end disconnect from database by typing the number 0 which corresponds to the disconnect procedure.

In the following table the invoked procedures are listed. For each of them the expected behaviour regarding the tables in the database and the test result are reported.

Procedure name	**Expected behaviour**	**Test Result (passed/failed)**
globus_i_slot_table_deactivate	Db disconnect	Passed
globus_i_slot_table_activate	Db connect	Passed
globus_slot_table_create	Insert into RESOURCE if the resource is not present	Passed
globus_slot_add	Insert into SLOT if the slot can fit	Passed
globus_slot_delete	Delete from SLOT and DIFFSERV if the bound object exists	Passed
globus_slot_modify	Update of SLOT table	Passed
globus_slot_properties	Select from SLOT	Passed
globus_slot_bind_object	Update or insert into DIFFSERV	Passed
globus_slot_status	Select from slot and status run time calculation	Passed
globus_slot_get_object	Select from DIFFSERV	Passed
globus_slot_table_find	Return the Resource_id associated to a given slot_id	Passed
globus_slot_table_dispose	Delete in RESOURCE	Passed

globus_slot_dump	Print out information from table SLOT given a Resource_id	Passed
globus_slot_get_quantity	Select from RESOURCE	Passed
globus_slot_dump_all	Dump the whole table into memory	Passed
globus_slot_check_for_changes	Update the Status field of SLOT	Passed
globus_slot_callback_register	This test can be done just if Globus module is active and not in the stand alone case.	Not performed

3 Integration Tests

This phase refers to the tests done at CNAF on the machine datatag4.
The new module slot_db_manager has been included in Gara 1.3.0 with Globus 2.2. After the modifications in Gara generating system the network resource manager links the libgara_slot_db_manager.la library in order to store the reservations in the database.
In this test phase the “Administrators Guide to Gara” (March 2000) has been followed in order to verify that the network reservations are stored properly in the database and the router Cisco 7200 is configured in the correct way.

3.1 Testing the setup flow script

At first we tried to launch the setup_flow script from command line in order to configure a router Cisco 7200. Usually the script is launched automatically by Gara network resource manager. In this case its correct behaviour has been verified before using it inside the Gara code. Following the Gara Guide mentioned above the script has been modified in order to support a different prompt that the router requires at the beginning.
The following lines have been added in the gara/resource_manager/utilities/ setup_flow. in file where the telnet to the router is done:

```
spawn telnet $router
expect_after default { close; continue }

expect  "name: "  { send "$username\r" }
expect  ">"         { send "enable\r" }
expect  "word: "  { send "$password\r" }
expect "#"          { send "configure terminal\r" }
```

After the telnet, the router at CNAF is configured to receive the username and to use the prompt “>”. But the script needs the prompt: “#”. Using the command “enable” the user becomes root and the current prompt changes in the correct one.

As second step we have modified the setup_flow.cfg file where we have inserted the login ID and the password to access the router, its IP address, the name of each ingress interface and for each interface, the computers that are nearest to that interface.

Here below the setup_flow.cfg is shown:

```
#
# First specify the ingress router
#
1_ROUTER          131.154.99.201
1_USERNAME        garr-ip
1_PASSWORD        xxxxxxxx
#  Up to six Interfaces. Declare only those you need
#              Their Names
#                     End-System's address connected to interface
#                     Note: This is not the router's interface address
1_INTERFACE1      FastEthernet0/0  131.154.99.24
1_INTERFACE2      FastEthernet3/0 192.168.2.24
#
# If you have additional interface, insert it in the form of
#1_INTERFACE3      atm6/0/0 140.221.50.100
#
# Now we need to specify the egress router
#
#2_ROUTER          egressrouter.mcs.anl.gov
#2_USERNAME        username
#2_PASSWORD        password
#  Up to three Interfaces. Declare only those you need
#              Their Names
#
```

In the test just one router Cisco 7200 with two interfaces has been used.

After the configurations described above the setup_flow script has been run from command line with the following parameters:

setup_flow < setup | teardown> <config_file> <source-ip> <source-port> <dest-ip>
<dest-port> <bps> <burst-size> <burst-size-exceed> <acl-to-use>"
<protocol (tcp | udp)> <Priority-Queuing (0 | 1)>"

In our case:

setup | teardown = setup (teardown means the deletion of the configuration on the router);
config_file = …/ setup_flow.cfg (specify the path to the file);
source-ip = 131.154.99.24;
source-port = 20000;
dest-ip = 192.168.2.24;
dest-port = 30000 ;
bps = 8000 ;
burst-size = 8000 ;
burst-size-exceed = 9180 ;
acl-to-use = 107 ;
protocol = tcp;
priority = 1.

3.2 Testing the Network Resource Manager

To configure the resource manager is necessary to modify the file
gara/resource_manager/etc/ diffserv_manager.conf.
The configuration file used in the performed tests is:

```
# A Configuration file for the Diffserv Manager

Port          5692
Quantity      45000
ClearSlotTable true
LoggingFileName   /tmp/diffserv_manager.log
SlotTableFilename diffserv_manager.slot_table
Verbose True
PublicationMethod web
WEBPublishFile diffserv_manager_slots.html
NoOfRouters  1
IPAddressesServed[1] 131.154.99.24,192.168.2.24
```

Note that NoOfRouters is the number of routers that can be configured and in our case is one. That means that in the mentioned tests only one edge router is used at CNAF to perform network reservation. With IPAddressesServed[1] are indicated the IP addresses of

the machines that are controlled by the router. This last information is the same as the one contained in setup_flow.cfg.

To run the diffserv manager it is necessary to prepare two shells where the environment variable are set as described in paragraph 1.2.

In the first shell the network resource manager runs:

> ./diffserv_manager

In the second shell, when the diffserv manager has already started, the test program is launched:

> ./diffserv_manager_test 131.154.99.24 192.168.2.24

Note that the IP addresses listed above are the ones of the computers involved in the reservation and they are stored in setup_flow.cfg and in diffserv_manager.conf files.

In the next table the actions performed during this test phase are listed.

Each line of the table contains the action on the reservation as is written in the test program and the correspondent test result is associated. In the table some actions are performed twice to make stand out the modifications obtained through a particular action. For example the status action makes stand out the changes in the data that the modify action has done.

Action	**Test Result (passed/failed)**
Connect	Passed
Create a reservation	Passed
Status of the reservation	Passed
Bind the reservation	Passed
Status of the reservation	Passed
Get the reservation properties	Failed / Passed after bug fix
Modify the reservation	Passed
Get the reservation properties	Failed / Passed after bug fix
Register a Callback	Passed
Status of the reservation	Passed
Wait for receiving Callbacks	Passed (11 Callbacks received)
Destroy the slot (delete the reservation)	Passed

The action of “Get the reservation properties” performs a call to the globus_slot_properties interface in the slot_db_manager module.

Sometime in the diffserv_manager code this procedure is invoked with two arguments set to NULL. In this case no data have to be given back as function output. In the interface implementation the check of the arguments was missing and a segmentation fault occurred in those cases of NULL argument. The bug has been fixed and both the module and the integration tests have been performed again.

5 Conclusions

After the database insertion in the Gara code two different types of tests have been performed. At first the module tests have been applied in order to test the library which manages the database as stand alone module. All these tests passed.
After that the integration phase has followed and it has been verified that the database is properly handled by the network resource manager of Gara.
In this test phase a bug has been found and fixed.