DataTAG
Advance Reservation WP2
Database Management in GARA

Database Specification

Network reservation in Gara
With database

Chiara Curti-Emanuela Bertelli

10/30/2003

DATA TAG
Advanced Reservation WP2
Database Management in GARA

1 Introduction

Aim of this document is to provide software specification for database handling instead of slot manager module in network reservation of Gara.

In the second paragraph the schema of the database is described. In particular three tables have been defined in order to contain all needed information to switch from a file-oriented mechanism to a database management. Some columns in defined tables are not handled at the moment but are useful for possible future features without any additional cost for database structure.

In the third paragraph the software detailed specification for the globus_slot_yyy interfaces is provided. In particular it is described how the functions add, modify, delete and in general handle the fields of the tables in the Gara database.

2 Database Schema

In following paragraphs three tables are described:

- RESOURCE;
- SLOT;
- DIFFSERV.

RESOURCE and SLOT tables are generic for all types of resources managed by Gara (network, CPU and disk) while DIFFSERV table is specific for diffserver network reservation.

To implement the database we have used the MySQL for database server and MySQL Connector/ODBC (also known as MyODBC) to connecting the database server via ODBC database API. The MyODBC is one of the most popular ODBC Driver in the open source market, used by many users to access the MySQL functionality. Open Database Connectivity (ODBC) is a widely accepted application-programming interface (API) for database access.

The following tables are of type transaction safe because a transactional behaviour has been chosen. A transaction is a sequence of one or more SQL statements that form a logical unit of work. Each SQL statement in the transaction performs a part of a task and only when all statements in the transaction are executed successfully the task is completed. The general flow control of transaction is: start a transaction, execute the statements and at the end commit or rollback the statements, according to their success or failure. By default, Connector/ODBC runs in autocommit mode. This means that MySQL will store the statements on disk as soon as they are performed.

If transaction-safe tables are used, MySQL can be put into non-autocommit mode and in this case COMMIT has to be used to store the changes to disk or ROLLBACK to ignore the changes that have been done since the beginning of the transaction.

In order to support the transactional behaviour the creation of transaction-safe tables like InnoDB is required.

2.1 Table RESOURCE

This table lists all resources present at the moment whatever is their type.

The resources are inserted by GARA code: one called PRIMARY from resource manager and the others called slot_table n inserted by specific manager.

To explain that is necessary to considerate the structure of the code.

The code is structured to have a common part, independent of resources, and a specific part for a specific resource.

When the resource manager is mentioned that refers to the common code and this part inserts always the resource called PRIMARY.

When the diffserv manager is mentioned, that indicates the specific code of resource manager to reserve the network. This is the specific part of the code that creates other resources and adds new lines in the current table.

For the diffserv manager is possible to insert a number of resources equal to the number of routers (slot_table n). To modify this number it is necessary to modify the current code. The resources besides the PRIMARY inserted by diffserv manager, indicate the other routers that can be configured in the domain. Each router controls a set of machines that are nearest to a defined router interface (for example atm, Ethernet).

Regarding the RESOURCE table, the couple (Unique_text_id, Resource_type) present in each line is different from the other analogous couples present in the table. To this unique couple is associated the unique identifier Resource_id by the database so each resource can be distinguished among all.

```
CREATE TABLE RESOURCE (
        Resource_id        INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
-- resource unique code, primary key of this table;
        Total_quantity     DOUBLE,
-- total quantity of available resource;
         Last_check_time   MEDIUMINT,
-- it indicates the last time the resource was verified;
         Unique_text_id    VARCHAR (40),
-- string identifier, which identifies the resource together with Resource_type.
-- When a new line is inserted, the resource manager provides this parameter
-- and it can be:
-- "Primary" if it is provided as first by the resource manager
-- "slot-table %d" if it is created by diffserver manager
        Resource_type      ENUM("DIFFSERV", "DPSS", "DSRT","MPLS"),
-- type of the resource to be reserved (disk, cpu or network).
-- In case of network reservation it can be distinguished between diffserver or mpls, but
-- at the moment just diffserver is managed.
        Policy             BLOB,
-- Allocation policies for the resources.
        PRIMARY KEY (Resource_id)
) TYPE=InnoDB;
```

Note 1: In this table the primary key Resource_id is specified with the attribute AUTO_INCREMENT because the key is provided automatically by the database. It can be retrieved from the database with the query of select LAST_INSERT_ID ().

2.2 Table SLOT

The table records all reservations, which have been put forward for a given resource.

In the table many entries can refer to the same resource because more reservations can be performed on the same resource in different periods of time or in the same period but using just a percentage of the available total resource quantity.

To represent the time the Unix format has been chosen where Unix time tells the amount of seconds elapsed since 1.1.1970.

```
CREATE TABLE SLOT (
        Resource_id       INTEGER UNSIGNED NOT NULL,
-- resource unique code, to which the reservation refers.
-- Reference key for this table due to the fact that it is the RESOURCE table's primary key
        Slot_id           INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
-- reservation unique code, primary key of this table;
        Start_time        BIGINT UNSIGNED,
-- it indicates the start of the reservation.
-- time_t type (unsigned long)
        End_time          BIGINT UNSIGNED,
-- it indicates the end of the reservation.
-- It is a not mandatory value because it is known through Start_time and Duration
-- but it is recorded for convenience.
-- time_t type (unsigned long)
        Duration          BIGINT UNSIGNED,
-- temporal duration of reservation. If it is INDEFINITE_DURATION then also
-- End_time becomes INDEFINITE_DURATION. In other cases:
-- End_time = Start_time + Duration
-- time_t type (unsigned long)
        Quantity          DOUBLE,
-- quantity of resource reserved in a period of time.
-- In case of diffserver(network) or dpss (disk) it is the bandwith;
-- In case of dsrt (cpu) it is percent_cpu, the requested percentage of cpu.
```

```
            Call_callback         BOOL,
-- indicates if it is necessary a callback invocation since a certain event happened;
            Status               ENUM ("SLOT_ADDED", "SLOT_DELETED", "SLOT_MODIFIED",
“SLOT_STARTED”, “SLOT_ENDED”),
-- reservation status. The following values are calculated run time in the resource manager
--  code: SLOT_NOT_STARTED, SLOT_STARTED, SLOT_ENDED;
            Reported_start       SMALLINT,
-- indicates if the reservation has already started
            Reported_end         SMALLINT,
-- indicates if the reservation has already ended
            User_id              VARCHAR(40),
-- user name for future monitoring. It can be useful to know the name of the user who
-- asked for a resource;
             PRIMARY KEY (Slot_id)
) TYPE=InnoDB;
```

2.3 Table DIFFSERV

The table records the additional and specific information needed to perform a network reservation. In this table the data to bind to a reservation are stored.

```
CREATE TABLE DIFFSERV (
           Slot_id              INTEGER UNSIGNED NOT NULL,
-- reservation unique code, to which the bound data refer.
-- Primary key for this table.
           Reservation_started    BOOL,
-- indicates if the reservation has started
           Reservation_ended      BOOL,
-- indicates if the reservation has ended
            Source_ip             VARCHAR(30),
-- IP source address
            Dest_ip               VARCHAR(30),
-- IP destination address
             Source_port          INT UNSIGNED,
-- number of source port
             Dest_port            INT UNSIGNED,
-- number of destination port
```

```
        Subtype             ENUM("reservationsubType_Foreground",
                                 "reservationsubType_Background",
                                 "reservationsubType_LowLatency",
                                 "reservationsubType_Invalid"),
-- type of the network reservation:
-- normal or Foreground, bulk-transfer or Background, Low-latency
        Acl                 INT,
-- access list for Cisco routers
        Did_setup_flow      BOOL,
-- Boolean to indicate if the setup script has been sent to the router in order to
-- configure it after the binding of a reservation. If it is TRUE the script has been
-- sent and before performing an unbind or a slot destroy is needed to teardown
-- the router.
        Protocol            INT,
-- communication protocol, tcp or udp.
        Router_type          ENUM("Cisco", "Juniper", "Other"),
-- not used at the moment. This column will be used when more than one type of router will
-- be configured in Gara and it will be possible to associate a specific configuration script to
-- the correspondent available router.
        PRIMARY KEY (Slot_id)
) TYPE=InnoDB;
```

3 Software Detailed Specification

The following paragraphs describe the functions useful for advance reservation implemented in the slot_db_manager module.

In particular it is described how the interfaces add, modify, delete and in general handle the fields of the tables in the Gara database.

3.1 File organization

The files of the slot_db_manager module are organized as described below:

slot_table.c:	It contains the procedures of type globus_slot_yyy, which constitute the interface between resource manager and slot manager. The interface provides the calls to the database.
slot_table.h :	is the header file, which contains defines and functions prototypes.
garadbutility.c	It contains all the utilities needed to connect to the database, to disconnect, to get the environment variables, to print the information log message from the database and to evaluate the time of a reservation as requested by some procedures of slot_table.c.
garadbutility.h:	is the header file, which contains defines and functions prototypes.
slot_errors.h:	this header file contains the enum which defines the errors of type: GLOBUS_SLOT_MANAGER_ERROR_something. For database management the following error has been added: GLOBUS_SLOT_MANAGER_GENERIC_DB_ERROR.

3.2 Procedure description

3.2.1 globus_i_slot_table_activate ()

It contains some initializations needed before creating any slot tables.

Input: nothing;

Output: nothing;

Return: GLOBUS_SLOT_MANAGER_ERROR_NONE if the activation of the module and the connection to the database are executed with success, otherwise the correspondent error code.

Actions:

It activates the module and connects to the database.

In this procedure are called the:

-globus_module_activate (GLOBUS_COMMON_MODULE), for common module of the whole globus toolkit;

-ldbConnect ()

3.2.2 globus_slot_init ()

It contains some initializations needed before creating any slot tables.

Input: slot_table_filename (*char);

Output: nothing;

Return: GLOBUS_SLOT_MANAGER_ERROR_NONE if OK, otherwise if not OK.

Actions:

To maintain unchanged the interface, the name of the old file used to store the reservations is assigned to the variable:

_slot_table_filename.

3.2.3 globus_i_slot_table_deactivate ()

It cleans up the module and disconnects from the database.

Input: nothing;

Output: nothing;

Return: GLOBUS_SLOT_MANAGER_ERROR_NONE always.

Actions:

It deactivates the module and disconnects from the database.

In this procedure are called the:

-globus_module_deactivate (GLOBUS_COMMON_MODULE), for common module of the whole globus toolkit;

-ldbDisconnect ()

3.2.4 globus_slot_table_create ()

It creates and initializes a new record into RESOURCE table.

The couple (Unique_text_id, Resource_type) is unique in the whole table and identifies an inserted record. A new line is inserted into this table only if the same couple of values is not already present.

Input: total_quantity:

flags: int;

unique_text_id:

slot_table_id a pointer to slot_table_id_t, that represents the integer to give in output in order to contain the identifier of the inserted record.

Output: slot_table_id see description above for this parameter.

Return: GLOBUS_SLOT_MANAGER_ERROR_NONE if the verify or the insertion of the record into the database is executed with success, otherwise the code of relative error.

Actions:

It locks all tables in modality exclusive.

It calls the utility procedure get_resource_type which finds the resource type through the old file name used in case of slot table stored on a file. The use of the old file name is necessary in order to preserve the interface with the other modules.

It makes a Sql select, to find if the actual resource is already present, and in that case gets its identifier (Resource_id primary key into RESOURCE table)

It makes a Sql insert, with the parameters passed in input, if the actual resource is not already present in the table. This behaviour is the same as in the correspondent interface used for storing into a file. After the insertion of the new record, the procedure gets its identifier recovered thought a select of last insert id.

It unlocks all tables.

In this procedure are called the:

- IPCLOCK ();

- get_resource_type ();

-SQLExecDirect (**selectHstmt**, selsql, SQL_NTS) execution of sql statement for a SELECT from RESOURCE to verify the content of RESOURCE table;

-SQLExecDirect (**insertHstmt**, inssql, SQL_NTS) execution of sql statement for INSERT into RESOURCE of the new line. After that is necessary the execution of an SQLExecDirect (selectHstmt, "**SELECT LAST_INSERT_ID ()**", SQL_NTS) that select the last record inserted into the table and so the Resource_id is retrieved and provided as output parameter.

- IPCUNLOCK ();

3.2.5 globus_slot_add ()

It inserts a new reservation (slot) into the SLOT table of the database after the check that the slot can fit.

If duration = INDEFINITE_DURATION the value NULL is inserted in the fields Duration and End_time of the table SLOT.

Input: table_id: Resource_id of table RESOURCE;

start_time: when the reservation should start;

duration: how long the reservation lasts;

quantity: how much quantity the reservation wants;

call_callback: boolean for calling the callback;

slot_id: pointer to an integer which contains the identifier of the inserted record to give in output.

Output: slot_id: primary key of SLOT table provided by slot_id parameter.

Return: GLOBUS_SLOT_MANAGER_ERROR_NONE if OK, GLOBUS_SLOT_MANAGER_ERROR_SLOT_DOESNT_FIT if slot not inserted in SLOT table, otherwise the code of relative error.

Actions:

It locks all tables in modality exclusive.

It makes a Sql select, to find the total quantity, from RESOURCE table.

For the resource involved in a new reservation, for each instant present in the duration of this reservation, the quantity demand is checked through the addition of the new requested quantity to the others quantity already reserved. If the result doesn't exceed the total quantity second per second, we say that the slot can fit.

To know the duration interval where the check of "slot can fit" has to be performed, it is necessary to find the variable latest_time (how far we have to look into the future to see if a slot can fit) through an algorithm, if the duration = INDEFINITE_DURATION and by latest_end_time = start_time + duration if the duration is not indefinite.

In case of INDEFINITE_DURATION, this algorithm is implemented by the utility ldbCalculate_latest_end_time defined in the file garadbutility.c. To calculate the latest end time we choose the biggest value between the End_time that terminates as last among the present reservations and the Start_time, which begins as last.

In ldbCalculate_latest_end_time procedure there is a Sql select from SLOT table to have a MAX (End_time) and MAX (Start_time) of the all reservations already present in the SLOT table.

Finally, if the slot can fit, a Sql insert, is called in order to add the slot. The parameters that are inserted are the ones passed in input and the Duration and the End_time modified as described above.

In SLOT table are inserted other two parameters calculated run time: Reported_start and Reported_end (both Boolean). They depend on the local time and determinate that the reservation has already started/finished.

Then the procedure unlocks all tables.

In this procedure are called the:

-IPCLOCK ();

-SQLExecDirect (**selectHstmt**, selsql, SQL_NTS) execution of sql statement for a SELECT from RESOURCE to retrieve Total_quantity;

-ldbCalculate_latest_end_time (table_id, &latest);

-SQLExecDirect (**selectHstmt**, selsql, SQL_NTS) execution of sql statement for a SELECT from SLOT to have instant for instant SUM (Quantity);

-SQLExecDirect (**insertHstmt**, inssql, SQL_NTS) execution of sql statement for INSERT into SLOT the new reservation and after that is necessary the execution of a SQLExecDirect (selectHstmt, "**SELECT LAST_INSERT_ID ()**", SQL_NTS) that select the last record inserted into the table. So the ouput parameter slot_id is retrieved;

-IPCUNLOCK ();

3.2.6 globus_slot_delete ()

It deletes a reservation from SLOT table and DIFFSERV table.

Input: table_id: Resource_id of table RESOURCE;

slot_id: Slot_id of the record to delete from SLOT table;

Output: nothing;

Return: GLOBUS_SLOT_MANAGER_ERROR_NONE if OK, otherwise the code of the correspondent error.

Actions:

It locks all tables in modality exclusive.

It makes a Sql select of Call_callback of the record to delete from SLOT table, to activate, if it is necessary the callback related to this slot.

It builds a Sql delete of the slot_id provided in input from SLOT table.

It makes a Sql delete from DIFFSERV table of the record identified through the slot_id passed in input.

It unlocks all tables.

In this procedure are called the:

-IPCLOCK ();

-SQLExecDirect (**selectHstmt**, selsql, SQL_NTS) execution of sql statement for a SELECT from SLOT to retrieve Call_calback;

-SQLExecDirect (**deleteHstmt**, delsql, SQL_NTS) execution of sql statement for a DELETE from SLOT where Slot_id is equal to slot_id in input;

-SQLExecDirect (**deleteHstmt**, delsql, SQL_NTS) execution of sql statement for a DELETE from DIFFSERV where Slot_id is equal to slot_id in input;

-IPCUNLOCK ();

-The callback now is called, if the slot has been deleted with success and its Call_callback indicates that is necessary (value equal to TRUE);

3.2.7 globus_slot_modify ()

It attempts to modify a reservation. If the modify fails, the old reservation will still hold.

Input: table_id: Resource_id of table RESOURCE;

slot_id: Slot_id of the record to be modified in the SLOT table;

start_time: when the reservation should start;

duration: how long the reservation lasts;

quantity: how much quantity the reservation wants;

Output: nothing;

Return: GLOBUS_SLOT_MANAGER_ERROR_NONE if OK, otherwise the code of the correspondent error.

Actions:

It locks all tables in modality exclusive.

It makes a Sql select of Call_callback of the record to modify from SLOT table, to activate, if it is necessary the callback related to this slot.

It is necessary to verify also if the slot can be modified and this depends on its quantity. The steps to follow are the same as described in the case of slot addition in order to understand if the slot can fit.

The control is performed without including the slot to modify.

In ldbCalculate_latest_end_time_others procedure there is a Sql select from SLOT table to have a MAX (End_time) and MAX (Start_time) of the reservations already present in the SLOT table excluding the slot to modify.

If the slot can be modified, a Sql update is called. In SLOT table other two parameters are modified run time: Reported_start and Reported_end (both Boolean). They depend on the local time and determinate that the reservation has already started/finished.

It unlocks all tables.

In this procedure are called the:

-IPCLOCK ();

-SQLExecDirect (**selectHstmt**, selsql, SQL_NTS) execution of sql statement for a SELECT from SLOT to retrieve Call_calback;

-SQLExecDirect (**selectHstmt**, selsql, SQL_NTS) execution of sql statement for a SELECT from RESOURCE to retrieve Total_quantity;

-ldbCalculate_latest_end_time_others (table_id, &latest, slot_id);

-SQLExecDirect (**selectHstmt**, selsql, SQL_NTS) execution of sql statement for a SELECT from SLOT, exception for slot_id to modify, to have instant for instant SUM (Quantity);

-SQLExecDirect (**updateHstmt**, updsql, SQL_NTS) execution of sql statement for UPDATE into SLOT the relative reservation of the slot to modify;

-IPCUNLOCK ();

-Callback now is called, if the slot can be modified with success and its Call_callback value is TRUE.

3.2.8 globus_slot_properties ()

Given a reservation (slot_id), this procedure retrieves information about it. If you don't care about any particular property, just pass in GLOBUS_NULL for the property pointer.

Input: table_id: Resource_id of table RESOURCE;

slot_id: Slot_id of the reservation in SLOT table;

Output: start_time: when the reservation should start;

duration: how long the reservation lasts;

quantity: how much quantity the reservation wants;

object_id: this parameter is always NULL;

Return: GLOBUS_SLOT_MANAGER_ERROR_NONE if OK, otherwise the code of the correspondent error.

Actions:

It locks all tables in modality exclusive.

It makes a Sql select of Start_time, Duration, and Quantity of the reservation given in input (slot_id) from SLOT table in order to retrieve the requested data.

It unlocks all tables.

In these procedures are called the:

-IPCLOCK ();

-SQLExecDirect (**selectHstmt**, selsql, SQL_NTS) execution of sql statement for a SELECT from SLOT to retrieve Start_time, Duration, and Quantity;

-IPCUNLOCK ();

3.2.9 globus_slot_bind_object ()

It attachs some extra data to a reservation that has already been created. All data needs to be encoded to a string. This procedure is invoked only by the diffserv_manager for the network reservation.

Input: table_id: Resource_id of table RESOURCE;

slot_id: Slot_id of the reservation to which the data are bounded;

information_to_store: the bound object is a pointer to char;

Output: nothing;

Return: GLOBUS_SLOT_MANAGER_ERROR_NONE if OK, otherwise the code of the correspondent error.

Actions:

It locks all tables in modality exclusive.

It makes a Sql select of slot_id from SLOT table, to verify that the slot which has to be bound is present into SLOT table.

It performs a sscanf to retrieve the single information from the information_to_store given in input.

To have the network reservation subtype (Foreground, Background, LowLatency, Invalid) the internal utility findSubtype_for_Bind is called. Aim of this procedure is to convert the subtype of a reservation from an integer provided by diffserver manager to a string requested in the database.

After that the procedure makes a Sql select of slot_id from DIFFSERV table. With this action we would check if the involved slot is present in table DIFFSERV; if it is already stored, this is a bind after a modify and an update is necessary. Otherwise an insert is performed.

Now a Sql update or a Sql insert will follow with the values previously calculated.

It unlocks all tables.

In this procedure are called the:

-IPCLOCK ();

-SQLExecDirect (**selectHstmt**, selsql, SQL_NTS) execution of sql statement for a SELECT from SLOT to verify if the Slot_id given in input is present;

-findSubtype_for_Bind(subtype) where the subtype retrieved from the information_to _store string is converted in the string used in the database;

-SQLExecDirect (**updateHstmt**, updsql, SQL_NTS) execution of the sql statement for UPDATE into DIFFSERV with the information to bound in the right format;

-or SQLExecDirect (**insertHstmt**, inssql, SQL_NTS) execution of the sql statement for INSERT into DIFFSERV with the information to bound in the right format;

-IPCUNLOCK ();

3.2.10 globus_slot_status ()

It finds out the status of a particular slot. In particular it calculates if a slot has begun or finished.

Input: table_id: Resource_id of table RESOURCE;

slot_id: Slot_id of table SLOT;

Output: status: Status of the reservation. It can be of type: SLOT_ENDED, SLOT_STARTED,

SLOT_NOT_STARTED

Return: GLOBUS_SLOT_MANAGER_ERROR_NONE if OK, otherwise the code of the correspondent error.

Actions:

It locks all tables in modality exclusive.

It makes a Sql select from SLOT table of Reported_start and Reported_end of the reservation given in input (slot_id), to verify if the reservation has ended, started or not started.

It unlocks all tables.

In this procedure are called the:

-IPCLOCK ();

-SQLExecDirect (**selectHstmt**, selsql, SQL_NTS) execution of sql statement for a SELECT from SLOT to retrieve Reported_start and Reported_end;

-IPCUNLOCK ();

3.2.11 globus_slot_get_object ()

Get extra data bound to a reservation that are stored in table DIFFSERV. All data needs to be encoded to a string. This procedure is invoked only by the diffserv manager for network reservations.

Input: table_id: Resource_id of table RESOURCE;

slot_id: Slot_id of the reservation to which the data are bounded;

Output: information_stored: additional information stored;

Return: GLOBUS_SLOT_MANAGER_ERROR_NONE if OK, otherwise the code of the correspondent error.

Actions:

It locks all tables in modality exclusive.

It makes a Sql select of slot_id from SLOT table, to verify that the requested slot is present into SLOT table.

Now, if the slot_id is present, it makes a Sql select from DIFFSERV to retrieve the needed information.

It's necessary to allocate the space where put the result encoded as string. String length used for bound object is 1024.

It is necessary to call an internal utility called findSubtype_for_get. Its aim is to convert the subtype of a reservation from a string read in database to an integer requested by diffserver manager. In particular in the database the enum begin from 1 and are strings. This utility returns the right enumerate, if the conversion is ok, otherwise returns –1 to indicate that there is an error.

It operates a sprintf to assign the information retrieved from the select to the information_stored to provide in output.

It unlocks all tables.

In this procedure are called the:

-IPCLOCK ();

-SQLExecDirect (**selectHstmt**, selsql, SQL_NTS) execution of sql statement for a SELECT from SLOT to verify if the Slot_id given in input is present;

-SQLExecDirect (**selectHstmt**, selsql, SQL_NTS) execution of sql statement SELECT from DIFFSERV of the information related to the slot_id of the bound object.

-*information_stored = (char *) globus_malloc(1024) allocates the space for the string with information associated at bound object.

-subtype = findSubtype_for_get (subtype_string); where subtype_string is read from database and subtype is the correspondent integer to return to the diffserver_manager process.

-IPCUNLOCK ();

3.2.12 globus_slot_table_find ()

Given a Slot_id, the procedure verifies that it is present in SLOT table and returns the correspondent Resource_id.

Input: slot_id: Slot_id of table SLOT;

Output: nothing;

Return: table_id: Resource_id of table RESOURCE if the Slot_id is present in SLOT table, otherwise return –1 to indicate that there is an error.

Actions:

It locks all tables in modality exclusive.

It makes a Sql select of Resource_id from SLOT table for the slot_id given in input, to return the retrieved Resource_id.

It unlocks all tables.

In this procedure are called the:

-IPCLOCK ();

-SQLExecDirect (**selectHstmt**, selsql, SQL_NTS) execution of sql statement for a SELECT from SLOT to retrieve Resource_id;

-IPCUNLOCK ();

3.2.13 globus_slot_table_dispose ()

It deletes from RESOURCE, SLOT, and DIFFSERV the rows related to table_id in input.

Input: table_id: Resource_id of table RESOURCE and SLOT;

Output: nothing;

Return: GLOBUS_SLOT_MANAGER_ERROR_NONE if OK, otherwise the code of the correspondent error.

Actions:

It locks all tables in modality exclusive.

It makes a Sql select from RESOURCE table for the Resource_id given in input, to verify if the involved Resource is present in table RESOURCE

It builds a Sql delete from RESOURCE table concerning the row with the table_id provided in input.

Now for each Slot_id correspondent to the table_id in input the entry is deleted from the tables SLOT and DIFFSERV.

It unlocks all tables.

In this procedure are called the:

-IPCLOCK ();

-SQLExecDirect (**selectHstmt**, selsql, SQL_NTS) execution of sql statement for a SELECT from RESOURCE to verify if the Slot_id given in input is present;

-SQLExecDirect (**deleteHstmt**, delsql, SQL_NTS) execution of sql statement for a DELETE related to the row in the RESOURCE table where Resource_id is equal to table_id in input;

-In a cycle for each slot present in SLOT table which corresponds to the Resource_id in input makes:

-SQLExecDirect (**deleteHstmt**, delsql, SQL_NTS) execution of sql statement for the DELETE of the row from SLOT where Slot_id is equal slot_id of cycle;

-SQLExecDirect (**deleteHstmt**, delsql, SQL_NTS) execution of sql statement for the DELETE of the row from DIFFSERV where Slot_id is equal slot_id of cycle;

-End of cycle when there isn't any more slot_id in the SLOT table which belongs to the resource represented by table_id in input;

-IPCUNLOCK ();

3.2.14 globus_slot_get_quantity ()

It gets the total quantity from the RESOURCE table.

Input: table_id: Resource_id of table RESOURCE;

Output: quantity: total quantity available for the current resource;

Return: GLOBUS_SLOT_MANAGER_ERROR_NONE if OK, otherwise the code of relative error.

Actions:

It locks all tables in modality exclusive.

It makes a Sql select of Total_quantity from the RESOURCE table for the table_id given in input.

It unlocks all tables.

In this procedure are called the:

-IPCLOCK ();

-SQLExecDirect (**selectHstmt**, selsql, SQL_NTS) execution of sql statement for a SELECT from RESOURCE to retrieve the Total_quantity;

-IPCUNLOCK ();

3.2.15 globus_slot_dump ()

It prints out a table, for debugging purposes.

Input: table_id: Resource_id of table RESOURCE.

Output: nothing.

Return: void.

Actions:

It locks all tables in modality exclusive.

Now for each Slot_id contained in the SLOT table correspondent to the table_id in input, it makes an Sql select of Slot_id, Quantity, Start_time, and End_time from SLOT table in order to print the requested information. If the End_time read from the database is NULL, (reservation with INDEFINITE_DURATION) the string “INDEF” is printed, otherwise is printed the value read from the database.

It unlocks all tables.

In this procedure are called the:

-IPCLOCK ();

-In a cycle for each slot present in SLOT table makes:

-SQLExecDirect (**selectHstmt**, selsql, SQL_NTS) execution of sql statement for a SELECT from SLOT to retrieve Slot_id, Quantity, Start_time, and End_time;

-IPCUNLOCK ();

3.2.16 globus_slot_dump_all ()

For MDS publishing purposes, it dumps the whole table.

Input: table_id: Resource_id of table RESOURCE;

Output: contents: is a pointer to struct slot_item_t;

size: is a pointer to long (int) for the number of row in table SLOT;

Return: GLOBUS_SLOT_MANAGER_ERROR_NONE if OK, otherwise the code of correspondent error.

Actions:

It locks all tables in modality exclusive.

Now for each Slot_id contained in the SLOT table correspondent to the table_id in input, it makes a Sql select of Slot_id to count how many rows are contained into SLOT table in order to return this value in “size”.

This parameter is also the largeness of the struct to return in the “contents”.

It’s necessary to allocate the space where put the result for struct slot_item_t. This operation is performed with a globus_malloc.

Now for each Slot_id contained in the SLOT table correspondent to the table_id in input, it makes a Sql select of Slot_id, Quantity, Start_time, End_time, and Duration from SLOT table, in order to return the information. If the End_time read from database is NULL (reservation with INDEFINITE_DURATION) for End_time and Duration is assigned the value -1, otherwise is assigned the value read from the database. In case of the value of quantity and/or start_time read from the database is/are NULL the correspondent value in struct is/are assigned to 0.

It unlocks all tables.

In this procedure are called the:

-IPCLOCK ();

-SQLExecDirect (**selectHstmt**, selsql, SQL_NTS) execution of sql statement for a SELECT from SLOT for the table_id given in input. After that a SQLFetch (selectHstmt) in the cycle retrieves the number of slot_id present in the SLOT table.

-ptr = (slot_item_t *) globus_malloc (count*sizeof(slot_item_t)). It allocates the memory for struct slot_item_t to be returned for printing the information.

-In a cycle for each slot present in SLOT table it makes an SQLExecDirect (**selectHstmt**, selsql, SQL_NTS) execution of sql statement to retrieve Slot_id, Quantity, Start_time, End_time, Duration;

-IPCUNLOCK ();

3.2.17 globus_slot_check_for_changes ()

This function can be called repeatedly, while it returns TRUE. If it returns TRUE, then the slot_id and change can be examined to see how a particular reservation has changed. But at the moment this function returns just pointers to void and updates the field status in the database for each slot.

Input: table_id: Resource_id of table RESOURCE;

Output: slot_id pointer to void;

change pointer to void;

changed_entry pointer a void;

next_entry pointer a void;

Return: GLOBUS_SLOT_MANAGER_ERROR_NONE if OK, otherwise the code of the correspondent error.

Actions:

It locks all tables in modality exclusive.

It makes a Sql update: Status becomes SLOT_ENDED, Reported_start and Reported_end become equal to 1 (TRUE) where Reported_end is 0 AND Duration is NOT NULL AND current time is greater (or equal) than Start_time+Duration.

It makes a Sql update: Status becomes SLOT_STARTED, Reported_start equal to 1 (TRUE) where Reported_End is 0 AND current time is greater than Start_time AND Duration is NULL or current time is lower than Start_time + Duration.

It unlocks all tables.

In this procedure are called the:

-IPCLOCK ();

-SQLExecDirect (**updateHstmt**, updsql, SQL_NTS) execution of sql statement for UPDATE into SLOT Status, Reported_start, Reported_end with SLOT_ENDED;

-SQLExecDirect (**updateHstmt**, updsql, SQL_NTS) execution of sql statement for UPDATE into SLOT Status, Reported_start, Reported_end with SLOT_STARTED;

-IPCUNLOCK ();

3.2.18 globus_slot_callback_register ()

It registers a callback that gets called whenever something changes in that table. Note that we start up a thread that can't be killed.

Input: callback: globus_slot_table_callback_t;

user_parameter pointer a void;

Output: nothing;

Return: GLOBUS_SLOT_MANAGER_ERROR_NONE if OK, otherwise the code of the correspondent error.

Relevant code of error in this case is GLOBUS_SLOT_MANAGER_ERROR_CANT_MAKE_THREAD.

Actions:

It locks all tables in modality exclusive.

Main action for this procedure is to create a thread for a callback given in input. To obtain this goal a globus_thread_create (&thread_id, NULL, callback_thread, _slot_callback_info) function is called. The third parameter callback_thread is a procedure called periodically for polling purpose which finds the changes happened to a slot table.

If the thread is created properly, the global variable _slot_callback_info->started_thread is assigned to GLOBUS_TRUE, else the following error is assigned: GLOBUS_SLOT_MANAGER_ERROR_CANT_MAKE_THREAD

It unlocks all tables.

In this procedure are called the:

-IPCLOCK ();

-globus_thread_create(&thread_id, NULL, callback_thread, _slot_callback_info);

-IPCUNLOCK ();

3.2.19 callback_thread ()

This function periodically polls for changes to a slot table.

Input: nothing;

Output: nothing;

Return: GLOBUS_SLOT_MANAGER_ERROR_NONE if OK, otherwise the code of the correspondent error.

Actions:

It makes a Sql select from SLOT table to retrieve Resource_id, Slot_id, and Call_callback of each record present in this table. If Reported_end = 0 and Duration is NOT NULL and current time is equal greater of the Duration + Start_time, the reservation of this resource has ended.

The main action for each of this record is the call of the _slot_callback_info->callback_function if the Call_callback read from database is TRUE and _slot_callback_info is different from NULL. This procedure is invocated with follow parameters:

_slot_callback_info->callback_function (resource_id,slot_id,SLOT_ENDED, _slot_callback_info->user_parameter).

For each slot the procedure makes a Sql update for Status = SLOT_ENDED, Reported_start = 1 and Reported_end = 1.

It makes a Sql select from SLOT table to retrieve Resource_id, Slot_id, and Call_callback of each record present in this table. If Reported_end = 0 and current time is greater of Start_time and Duration is NULL or current time is equal minor of the Duration+Start_time, the reservation of this resource has started but not finished.

The main action for each of this record is the call of the _slot_callback_info->callback_function if the Call_callback read from database is TRUE and _slot_callback_info is different from NULL. This procedure is invocated with follow parameters:

_slot_callback_info->callback_function (resource_id,slot_id,SLOT_STARTED, _slot_callback_info->user_parameter).

For each slot the procedure makes a Sql update for Status = SLOT_STARTED, Reported_start = 1 and Reported_end = 0.

These actions are inserted in a cycle controlled by a global variable _stop_threads.

In this procedure are called the:

-SQLExecDirect (**selectHstmt**, selsql, SQL_NTS) execution of sql statement for a SELECT from SLOT to retrieve Resource_id, Slot_id, Call_callback;

-_slot_callback_info->callback_function(resource_id,slot_id,SLOT_ENDED, _slot_callback_info->user_parameter);

-SQLExecDirect (**updateHstmt**, updsql, SQL_NTS) execution of sql statement for UPDATE into SLOT Status, Reported_start, Reported_end with SLOT_ENDED;

-SQLExecDirect (**selectHstmt**, selsql, SQL_NTS) execution of sql statement for a SELECT from SLOT to retrieve Resource_id, Slot_id, Call_callback;

-_slot_callback_info->callback_function(resource_id,slot_id,SLOT_STARTED, _slot_callback_info->user_parameter);

-SQLExecDirect (**updateHstmt**, updsql, SQL_NTS) execution of sql statement for UPDATE into SLOT Status, Reported_start, Reported_end with SLOT_STARTED;

3.2.20 globus_slot_pos_to_beginning ()

This function sets the current entry of the table SLOT. In particular some information about the first Slot_id recovered by SLOT is returned, where the Slot_id are in increasing order.

Input: table_id: Resource_id of table RESOURCE;

Output: entry: struct for Slot of table SLOT;

Return: GLOBUS_SLOT_MANAGER_ERROR_NONE if OK, otherwise the code of the correspondent error.

Actions:

It locks all tables in modality exclusive.

It makes a Sql select of Slot_id, Start_time, Duration, End_time, Quantity, Reported_start, Reported_end, and Call_callback from SLOT ordered by Slot_id. This action returns the first slot.

These values are assigned to the entry struct to give in output.

It unlocks all tables.

In this procedure are called the:

-IPCLOCK ();

-SQLExecDirect (**selectHstmt**, selsql, SQL_NTS) execution of sql statement for a SELECT from SLOT to get back Slot_id, Start_time, Duration, End_time, Quantity, Reported_start, Reported_end, and Call_callback;

-IPCUNLOCK ();

3.2.21 globus_slot_pos_to_next_entry ()

This function positions the current entry in the SLOT table to the next entry.

In particular it is returned some values about the first slot with the Slot_id recovered from the table SLOT greater than the Slot_id provided by the entry given in input.

These Slot_id are in increasing order.

Input: table_id: Resource_id of table RESOURCE;

entry: struct for Slot of table SLOT;

Output: entry: next struct for Slot of table SLOT;

Return: GLOBUS_SLOT_MANAGER_ERROR_NONE if OK, otherwise the code of the correspondent error.

Actions:

It locks all tables in modality exclusive.

It makes a Sql select of Slot_id, Start_time, Duration, End_time, Quantity, Reported_start, Reported_end and Call_callback from SLOT where Slot_id is greater than the Slot_id of the entry. This action returns the first Slot_id after the Slot_id given in input.

These values are assigned to the entry struct to give in output.

It unlocks all tables.

In thise procedure are called the:

-IPCLOCK ();

-SQLExecDirect (**selectHstmt**, selsql, SQL_NTS) execution of sql statement for a SELECT from SLOT to get back Slot_id, Start_time, Duration, End_time, Quantity, Reported_start, Reported_end, and Call_callback;

-IPCUNLOCK ();

3.2.22 globus_slot_get_current_entry ()

This function delivers the information of the current entry in the SLOT table.

Input: table_id: Resource_id of table RESOURCE;

entry: struct for slot of table SLOT;

Output: slot_id correspondent to the entry given in input;

quantity: how much quantity the reservation wants regarding the entry given in input;

Return: GLOBUS_SLOT_MANAGER_ERROR_NONE if OK, otherwise the code of the correspondent error.

Actions:

It locks all tables in modality exclusive.

It makes a Sql select of Slot_id, Quantity from SLOT where Slot_id is equal to the Slot_id of the entry given in input.

These values are assigned to the slot_id and quantity to be given in output.

It unlocks all tables.

In this procedure are called the:

-IPCLOCK ();

-SQLExecDirect (**selectHstmt**, selsql, SQL_NTS) execution of sql statement for a SELECT from SLOT to get back Slot_id and Quantity;

-IPCUNLOCK ();