Spequlos documentation

INRIA

-

Contents

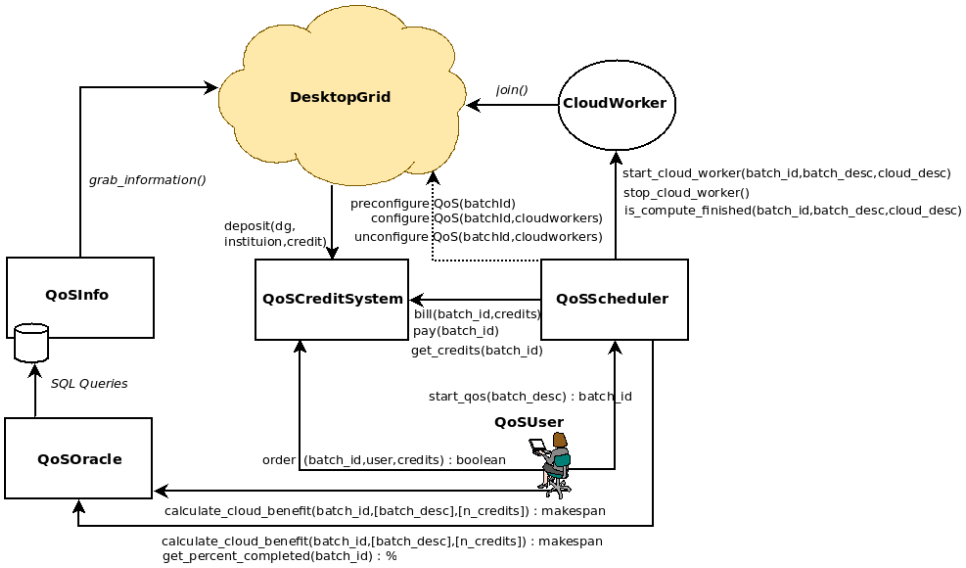

Figure 1: SpeQulos modules overview

1 SpeQuloS overview

1.1 SpeQulos presentation

The goal of the SpeQulos infrastructure is to bring QoS to Desktop Grid (DG) users. At the user point of view, the QoS denotes the time needed for a batch or Bag of Task (BoT), which is a set of individual work units or jobs, to be executed in the DG, since its submission to the delivery of the results of the last BoT job. This time is called makespan. The main objective of SpeQulos is to decrease a BoT makespan, according to user needs.

To decrease a BoT makespan, SpeQulos adds computing resources dedicated to its execution. SpeQulos uses computer nodes hosted in the cloud to provide those resources. It creates some DG workers (or Cloud Workers), makes them join the DG where the BoT is being processed and stop them when necessary.

As cloud resources are limited, SpeQulos provides a framework to share those resources between the DG users. Cloud resource usage is associated to a credit cost. Credits are earned by users' institution. When a user needs to improve the QoS for one of his BoT, he needs to spend some credits for this task.

To enable a smart use of the cloud resources, SpeQulos gather information about the QoS in a DG. This information is used by the SpeQulos oracle to evaluate the current state of the DG QoS and to anticipate the benefits of allocating cloud resources. A DG user can then query the oracle to know if it worth to spend some credits to improve the QoS of one of his BoT. SpeQulos also uses the oracle internally to only use cloud resources when needed.

1.2 SpeQulos modules

The SpeQulos system can be divided in several sub-modules (see figure 1):

- The Credit System module: Its goal is to manage all credit-oriented operations.
- The Information module: Its goal is to gather and maintain QoS related information about DG and their BoTs.
- The Oracle module: Its goal is to make some predictions on the current QoS in a DG, and on improvements that can be expected by spending credits.
- The Scheduler module: Its goal is to manage the cloud resource usage, and to supervise BoTs for which QoS has been requested.

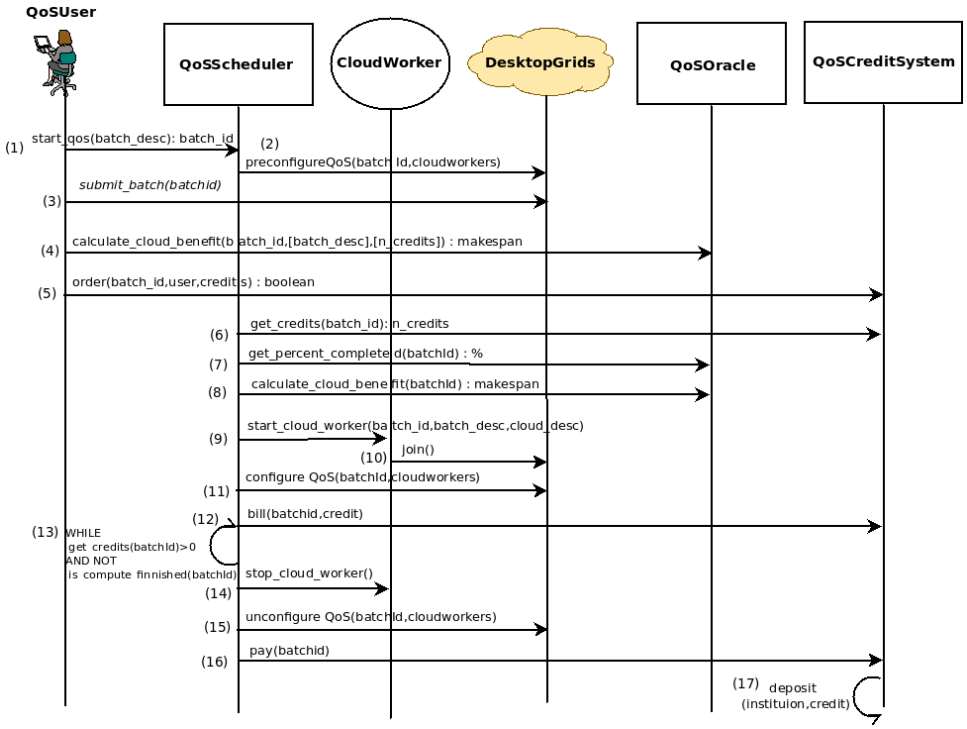

Figure 2: SpeQulos modules interaction during a typical use case scenario

As SpeQulos can be used in various contexts, its deployment is flexible. Each module has been designed to be independent from others. Hence, each module can be run on different computers. Communications between modules use web services.

SpeQulos does not depend on a particular DG technology. As the interfaces between SpeQulos and the DG is relatively small, only a small set of functions are specifics to a DG technology and need to be implemented accordingly. SpeQulos does not depend on a particular Cloud technology either. As with DG, the interface with the Cloud technology is small. Moreover, existing software such the EDGI's 3G bridge or the libcloud API are used as abstraction layer to use the various Cloud technologies transparently.

See section 3 for a detailed description of the SpeQulos modules.

1.3 Use case scenario

Figure 2 is a sequential diagram that represents a typical use case scenario of SpeQulos. The progression of the scenario is represented vertically, and the various function calls between SpeQulos modules are represented by arrows. Following is a description of the various steps of this scenario:

- The first step of the scenario is the request for QoS by a user, for one of its BoT, to the Scheduler (1). The scheduler returns to the user a BoT identifier (batch_id) that must be for the other transactions related to this BoT. The scheduler also adds the BoT to the list of its monitored BoTs. The scheduler may have to pre-configure the targetted DG to accept QoS (2). Then, the user submits its BoT to the DG, as he normally does, except that he needs to use the batch_id given by the scheduler (3).

- At any moment, the user can request the oracle to calculate the benefits of using cloud resources to improve the QoS of his BoT (4). According to the oracle answer, the user can order some QoS to the credit system (5). By doing this, he declares to the credit system that he is ready to spend a number of credits to improve the QoS of the BoT. The credit

system then verify that there is enough credits on the user's institution account to allow the order, and then allocate the credits to the user's BoT

- The scheduler periodically asks the credit systems if there is some credits allocated for the BoTs it manages (6). If some credits are available, and if the BoT completion is advanced enough (7), the scheduler asks the oracle what benefits for BoT's QoS can be expected by using cloud resources (8).
- If it worth to spend those credits, the scheduler starts a cloud worker (9). The cloud worker joins the DG, as any else worker would do (10). The scheduler then configures the DG to ensure that the cloud worker will only compute the jobs associated to the BoT for which credits have been spent (11).
- At each fixed period of time, the cloud resource usage must be billed (12). For each cloud worker started, the scheduler reports to the credit system the corresponding credits spent. If all the credits allocated to the BoT have been spent, or if the BoT computation finished (13), the cloud worker must be stopped (14), and the cloud worker assignment to BoT must be unconfigured in the DG (15).
- The scheduler then asks to the credit system to pay for the resources usage (16). The credit system then closes the order relative to the BoT. If the BoT computation was completed before all the credits have been spent, the credit system transfer back remaining credits to the user's institution account (17)

This scenario will be illustrated by a step by step example in section **??**.

2 SpeQulos Usage

In this section, we will present how a DG user can use SpeQuloS.

2.1 Getting SpeQuloS

SpeQuloS can be downlaoded at the address:

`http://graal.ens-lyon.fr/~sdelamar/spequlos-0.2.tar.gz`

2.2 Installing

The instructions to install SpeQuloS are detailed in the document `install.txt` in the `docs` directory of the SpeQuloS package.

2.3 Registering a BoT

To be supported by SpeQuloS, an user must register a BoT to SpeQuloS Scheduler. This registration means that the user MAY requieres some QoS for one BoT, and must be performed BEFORE the BoT submission to the DG.

The DG that will be used to execute the BoT must have been registered to the Scheduler during the SpeQuloS installation.

To register a BoT, the user must access to a Web page at the following address:

`http://<address of spequlos server>/spequlos/scheduler/start_qos.py`

Where <`address of spequlos server`> is the host address where the scheduler has been installed. For instance:

`http://localhost/spequlos/scheduler/start_qos.py`

A Web form page is displayed. In the dg_name field, the user has to enter the DG identifier where the BoT will be executed. This identifier is the same that has been used during DG registration.

To see which DG are registered, the user can access to the page:

```
http://localhost/spequlos/scheduler/admin_dg.py
```

Once the user submits the form, the server returns something like this:

```
Use this identifier for your batch:
batch_64c6f809-84d3-4e41-988a-e5d458f0ef4b
```

for an XWHEP DG, or

```
Use this identifier for your batch:
34
```

for a BOINC DG.

The given identifier is the BoT identifier that the user must to "tag" his BoT. This tag will later allow SpeQuloS to track the BoT execution in the DG.

2.4 Submitting a BoT

Describing how to submit a BoT to a DG is out of scope of this document. However, user have to remember to tag its BoT with the identifier given by the scheduler during BoT registration.

For an XWHEP DG, user has to use the `--xwgroup <uid>` option to pass this tag during submission, where <uid> is the XWHEP identifier refering to the XWHEP group which name is the tag.

For a BOINC DG, user has to use the `-batch <tag>` option of the `create_work` command, where <tag> is the tag given by the Scheduler.

2.5 Monitoring BoT QoS

At any time, an user can get informed of the expected QoS of a BoT by querying the Oracle. This feature is still experimental. To allow prediction, the BoT must be under execution on a DG monitored by the Information module.

To get informed of a BoT QoS, the user must access to a Web page at the following address:

```
http://<address of spequlos server>/spequlos/oracle/calculate_cloud_benefit.py
```

Where <address of spequlos server> is the host address where the Oracle has been installed. For instance:

```
http://localhost/spequlos/oracle/calculate_cloud_benefit.py
```

A Web form page is displayed. Only the `batch_id` field should be useful to the user. In the `batch_id` field, the user has to enter the BoT identifier (given by by the Scheduler during BoT registration). The Oracle will display a Web page containing:

```
----------------------------------------
BatchId: xxxxxx
----------------------------------------
Computing cloud benefits\dots
T1
T2
T3
N
----------------------------------------
```

With:

- T1 the expected makespan, or BoT completion time, for the BoT associated to the identifier given, without tail effect
- T2 the expected makespan for the BoT, with the tail effect
- T3 the expected makespan for the BoT, if some Cloud resources are used
- N is the expected cost of using Cloud resources, in credits

2.6 Allocating resources to enable QoS

If the user decides to, he can allocate some Cloud resources to support QoS for one of its BoT. In SpeQuloS, Cloud resources usage are associated to a credit cost. To enable QoS for a BoT by provisionning Cloud resources, the user has to allocate some credits to this BoT.

To enable QoS for a BoT, the user must ask the SpeQuloS CreditSystem to create an "order" for this BoT, and to allocate some credits from the user account to this order. Users' accounts are called Institution account. Several users can share the same institution account. For the order to be created, there must be more credits on the user's institution account than on requested on for the order.

As the CreditSystem is still under development, it is preconfigured with an unique user (identified by 0) with an institution account (identified by 0) with 1000 credits. SpeQuloS users should use this CreditSystem user to enable QoS for their BoT.

To create an order, the user must access to a Web page at the following address:

`http://<address of spequlos server>/spequlos/creditsystem/order.py`

Where <`address of spequlos server`> is the host address where the CreditSystem has been installed. For instance:

`http://localhost/spequlos/creditsystem/order.py`

A Web form page is displayed. In the `batch_id` field, the user has to enter the BoT identifier (given by by the Scheduler during BoT registration) for which he wants to provision Cloud resources. In the user_id field, the user should enter 0 (zero), which refers to the preconfigured CreditSystem user. In the credit field, the user should enter the number of credits he wants to spend for this BoT (curently, the cost of one hour of Cloud usage is 15 credits).

After submission, the CreditSystem will display a web page containing:

`True`

if the order is accepted (if user's has enough credits on his account) or

`False`

otherwise.

Once the order has been created, the SpeQuloS Scheduler will be able to start some resources from Cloud to support the BoT execution. If some credits have not been spent at the end of the BoT execution, they will be back to the user's account.

3 Modules description

In this section, each module will be presented in details. For each module, a description of the database used will be given, as well as a presentation of its main functions.

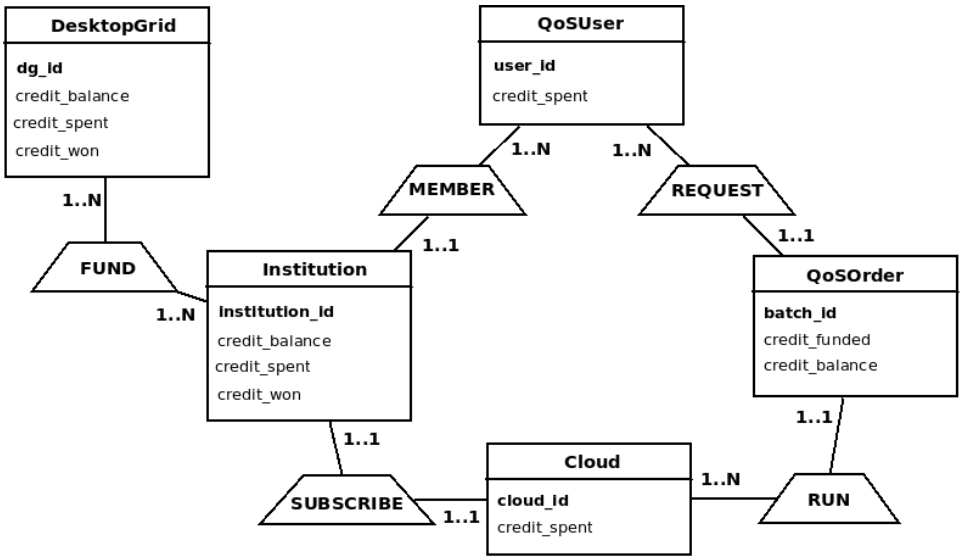

Figure 3: The Creditsystem module database concept map

3.1 The Creditsystem module

The credit system module is in charge of all credit-oriented operations. It stores the information about the credit accounts for the institutions, clouds, DG and orders given by users for their BoTs. It interacts with the users to receive orders, and with the scheduler to exchange information about credit spent for BoTs by spending some cloud resources. It also receives some deposit requests from DG administrators to fund an institution account.

3.1.1 Database

The Creditsystem's database, represented in figure 3, includes the following tables:

- QoSUser: Stores the credits spent by each users participating in SpeQulos, for statistics.
- Institution: Stores the credits balance for each institution, as well as the credits already won and spent, for statistics.
- DesktopGrid: Stores the credits balance for each DG, as well as the credits already won and spent, for statistics.
- Cloud: Stores the credits spent by each clouds participating in SpeQulos, for statistics.
- QoSOrder: Stores the credit balance for each order created by user for his BoT, as well as the credit originally funded.

3.1.2 Module's main functions

Bill

Web service to bill credits spent for a batch

- File: bill.py
- Arguments - Using the HTTP GET method:
  - batch_id – The batch identifier, as a string
  - credit – The number of credits spent, as a positive integer

  → If some arguments are missing, a web form will be displayed.
- Displays a Web page containing:

  - True if the bill is accepted
  - False if the bill is denied

- Notes:
  - The bill will be denied if all the credits allocated to a batch's order have already been spent.
  - A database configuration file named "db.cfg" must be present in the directory.

Deposit

Web service to deposit credits from a desktop grid to an institution account

- File: deposit.py
- Arguments - Using the HTTP GET method:
  - dg_id – The desktop grid identifier, as an integer
  - institution_id – The institution identifier, as an integer
  - credit – The number of credits to transfer, as a positive integer

  → If some arguments are missing, a web form will be displayed.
- Displays a Web page containing:
  - True if the deposit is correctly completed
  - True if the deposit is denied
- Notes:
  - A deposit will be denied if the DG credit balance is lower than the credit to transfer.
  - A database configuration file named "db.cfg" must be present in the directory.

Get_credits

Web service to get the number of credits in a batch order

- File: get_credits.py
- Arguments - Using the HTTP GET method:
  - batch_id – The batch identifier, as a string

  → If some arguments are missing, a web form will be displayed.
- Displays a Web page containing:
  - N where N is the number of credit in the order of the given batch_id
- Notes:
  - A database configuration file named "db.cfg" must be present in the directory.

Order

Web service to place an order for a batch

→ An order must be placed when a user decide to allocate some credit to one of his batch to enhance its QoS.

- File: order.py
- Arguments - Using the HTTP GET method:
  - batch_id – The batch identifier, as a string
  - user_id – The user identifier, as an integer
  - credit – The number of credits to allocate to the order, as a positive integer
  - → If some arguments are missing, a web form will be displayed.
- Displays a Web page containing:
  - `<div class="qos">True</div>` if the order is accepted
  - `<div class="qos">False</div>` if the order is denied
- Notes:
  - An order will be denied if there are not enough credits on the user's institution account.
  - A database configuration file named "db.cfg" must be present in the directory.

Pay

Web service to pay a batch order

→ When an order have to be closed (for instance, when the batch processing is finished), the pay operation must be called to acknowledge end of resources consumption.

- File: pay.py
- Arguments - Using the HTTP GET method:
  - batch_id – The batch identifier, as a string
  - → If some arguments are missing, a web form will be displayed.
- Displays a Web page containing:
  - `<div class="qos">True</div>`if the pay operation is correctly completed
- Notes:
  - A database configuration file named "db.cfg" must be present in the directory.

3.2 The Info module

The information module gathers and stores the information on DG relevant for QoS. Currently, it stores the information of batches completion history (in terms of number of completed jobs of a batch, compared to number of uncompleted jobs, as a function of time). It provides this information to the oracle.

Figure 4: The Info module database concept map

3.2.1 Database

The Info's database, represented in figure 4, includes the following tables:

- BatchCompletionHistory: Stores the number of completed and uncompleted jobs for one batch (denoted as batch_id), at some time (denoted as ts)
- WorkersHistory: Stores the number of workers and cloudworkers at some time (denoted as ts)

3.2.2 Module's main functions

Grabber_BOINC

Tools to grab batch completion from a BOINC DesktopGrid

→ It inserts the information in the QoSInfo database table "BatchCompletionHistory".

- File: grabber_BOINC.py
- Usage: ./grabber_BOINC.py URL
- Command line argument:
  - URL : The URL of the BOINC batch completion history wrapper page (i.e. http://<BOINC server address/BOINC-project_ops/BOINC-wrapper.php)
- Notes:
  - A database configuration file named "db.cfg" must be present in the directory.

Grabber_XWHEP

Tools to grab batch completion from an XWHEP DesktopGrid

→ It inserts the information in the QoSInfo database table "BatchCompletionHistory".

- File: grabber_XWHEP_bots.py
- Usage: ./grabber_XWHEP_bots.py URL
- Command line argument:
  - URL : The URL of the XWHEP batch completion history wrapper page (i.e. http://<XWHEP server address/spequlos/XWHEP-wrapper.php)
- Notes:
  - A database configuration file named "db.cfg" must be present in the directory.

Grabber_plainfile

Tools to grab information from a plain file and store it into the QoSInfo database

3.3 The Oracle module

The oracle module makes some prediction about QoS in the DG. Using the information provided by the Info module, the oracle's module computes an estimated makespan for a batch. The makespan estimations are performed with and without the use of cloud resources, to emphasize the benefit of using the cloud. The oracle module interacts with the users to help them to decide if it worth to spend credits for their batches, and with the scheduler, which verifies if it is really useful to use some cloud resources for a batch which has some credits allocated to it. The oracle module does not use a database.

3.3.1 Module's main functions

Calculate_cloud_benefit

Web service to know the benefit of using cloud resource to improve a batch's QoS

→ The function displays the various makespan information for the batch, to help the user/scheduler to decide to use QoS resources or not.

- File: calculate_cloud_benefit.py
- Arguments - Using the HTTP GET method:
  - batch_id — The batch identifier, as a string
  - current_time (Optional, for testing) — See info.get_elapsed_time
  - use_last_db_entry (Optional, for testing) — See info.get_elapsed_time

  → If the batch_id argument is missing, a web form will be displayed.
- Displays a Web page containing:

```
<div class="qos">T1</div>
<div class="qos">T2</div>
<div class="qos">T3</div>
<div class="qos">N</div>
```

With:

- T1 the expected makespan for batch without tail effect

  - T2 the expected makespan for batch under tail effect
  - T3 the expected makespan for batch using cloud resources
  - N is the cost of using cloud resources (n_jobs/10, will probably change)

- Notes:
  - A database configuration file named "info_db.cfg" must be present in the directory to connect to the SpeQuloS Info module database.

Get_elapsed_time

Web service to know the time elapsed since submission of a batch

- File: get_elapsed_time.py
- Arguments - Using the HTTP GET method:
  - batch_id — The batch identifier, as a string
  - current_time (Optional, for testing) — Retrieves batch's elapsed time since submission at specified fake "current_time", as an integer denoting the number of seconds since 1970-01-01 00:00:00 UTC.
  - use_last_db_entry (Optional, for testing) — If set to 1, use the timestamp found in the last database entry related to this batch as a fake current time.

  → If the batch_id argument is missing, a web form will be displayed.
- Displays a Web page containing:
  - `<div class="qos">N</div>` with N the time elapsed in seconds.
- Notes:
  - A database configuration file named "info_db.cfg" must be present in the directory to connect to the SpeQuloS Info module database.

Get_n_jobs

Web service to know the number of jobs in a batch

- File: get_n_jobs.py
- Arguments - Using the HTTP GET method:
  - batch_id — The batch identifier, as a string

  → If the batch_id argument is missing, a web form will be displayed.
- Displays a Web page containing:
  - `<div class="qos">N</div>` with N the number of jobs.
- Notes:
  - A database configuration file named "info_db.cfg" must be present in the directory to connect to the SpeQuloS Info module database.

Get_percent_completed

Web service to know the percentage of completion of a batch

→ The percentage of completion of a batch is the percentage of his jobs already completed compared to its total number of jobs.

- File: get_percentage_completed.py
- Arguments - Using the HTTP GET method:
  - batch_id — The batch identifier, as a string
  - current_time (Optional, for testing) — Retrieves batch percentage completion at specified fake "current_time", as an integer denoting the number of seconds since 1970-01-01 00:00:00 UTC.

  → If the batch_id argument is missing, a web form will be displayed.
- Displays a Web page containing:
  - `<div class="qos">F</div>` with F the completion ratio between 0 and 1
- Notes:
  - A database configuration file named "info_db.cfg" must be present in the directory to connect to the SpeQuloS Info module database.

Get_percent_distributed

Web service to know the percentage of distribution of a batch

→ The percentage of distirubtion of a batch is the percentage of its jobs already sended to workers compared to its total number of jobs.

- File: get_percentage_completed.py
- Arguments - Using the HTTP GET method:
  - batch_id — The batch identifier, as a string
  - current_time (Optional, for testing) — Retrieves batch percentage distribution at specified fake "current_time", as an integer denoting the number of seconds since 1970-01-01 00:00:00 UTC.

  → If the batch_id argument is missing, a web form will be displayed.
- Displays a Web page containing:
  - `<div class="qos">F</div>` with F the completion ratio between 0 and 1
- Notes:
  - A database configuration file named "info_db.cfg" must be present in the directory to connect to the SpeQuloS Info module database.

3.3.2 Internal functions

Compute_T_cloud

compute_T_cloud(x, elapsed_time, percent_completed): Computes T(x) for cloud

→ As we don't have any information on cloud performance, we currently use the formula: `T(x) for Cloud == T(x) for DG / 3`

- Arguments:
  - x – The percentage of batch completion
  - elapsed_time – The batch's elapsed time since its submission, as an integer
  - percent_completed – The batch completion ratio, as a float comprised between 0 and 1
- The function returns the number of seconds since jobs submission to compute x percent of the batch's jobs in the cloud, as an integer

Compute_T_dg

compute_T_dg(x, elapsed_time, percent_completed): Computes T(x) for DG, for arbitrary x by extrapolating T(percent_completed) == elapsed_time

→ T(x) denotes the time elapsed since batch submission when x percents of its jobs have been completed. To compute this time for arbitrary X with assume constant rate for jobs completion, i.e.:

```
T(x)/x == T(percent_completed)/percent_completed == elapsed_time/percent_completed
```

- Arguments:
  - x – The percentage of batch completion
  - elapsed_time – The batch's elapsed time since its submission, as an integer
  - percent_completed – The batch completion ratio, as a float comprised between 0 and 1
- The function returns the number of seconds since jobs submission to compute x percent of the batch's jobs, as an integer

3.4 The Scheduler module

The scheduler module manages the batches and the cloud workers during the system execution. It stores the batches for which QoS has been request by users and the cloud workers available. When cloud resources are used, it assigns which cloud worker must worker for a batch. It interacts with users who request QoS management for their batches. It also interacts with the Credit System to account the credits spent and to verify their availability. It interacts with the oracle to evaluate if the use of cloud resource is relevant. It starts the cloud workers, to make them join the DG and stop them. It also interacts with the DG directly, to make a batch being computed by cloud workers which has been funded by its user.

3.4.1 Database

The Scheduler's database, represented in figure 5, includes the following tables:

- QoSBatches: Stores the information on batches currently managed by QoS, for which cloud resources may be assigned to.
- CloudWorkers: Stores the information on cloud worker available in the system. The cloud worker may be assigned to a batch ("execute" association)
- DesktopGrid: Stores the information on DG available in the system.

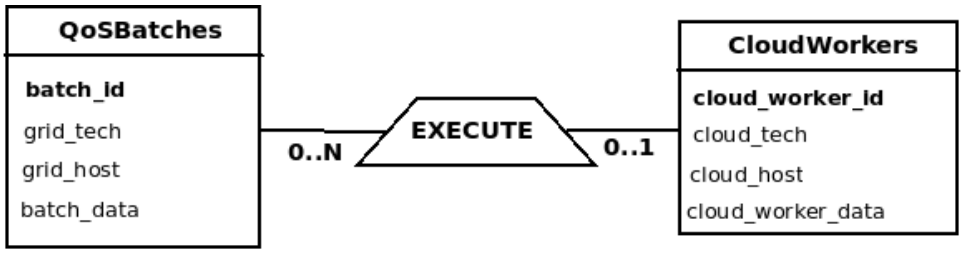

Figure 5: The Scheduler module database concept map

3.4.2 Module's main functions

Monitor_batches

Program that monitor QoS-enabled batches

→ This program manages batches previously created by the start_qos function by allocating cloud resources to batch being funded by users. It should be started periodically to perform monitoring operations.

- File: monitor_batches.py
- Notes:
  - A database configuration file named "db.cfg" must be present in the directory.
  - A file named "creditsystem.cfg" containing a line "address=<url>", with <url> the url to access to the Credit System web server, must be present in the directory.
  - A file named "oracle.cfg" containing a line "address=<url>", with <url> the url to access to the Oracle web server, must be present in the directory.
- Algorithm:

```
For each of batches, this function checks
  if the batch is not already computed by a CloudWorker,
                        (!!Could change in the future!!)
   and if we at 90% of the batch completion
   and if the oracle-predicted benefit of using cloud
                                              is enough,
   and if the batch order has more credits than the
                    credit cost predicted by the oracle,
   and if there is one cloud worker available,
      starting it and assigning the batch to it.
```

Monitor_cloud_workers

Program that monitor Cloud Worker currently running

→ This program manages Cloud Worker currently running, by billing the resources being spent, when stopping them if needed. It should be started periodically to perform monitoring operations.

- File: monitor_cloud_workers.py
- Notes:

  - A database configuration file named "db.cfg" must be present in the directory.
  - A file named "creditsystem.cfg" containing a line "address=<url>", with <url> the url to access to the Credit System web server, must be present in the directory.

- Algorithm:

```
For each of running cloud workers, this function checks
  if the worker finished his job,
      stop it
  else if there is still some credits in the batch order,
      bill the resource spent
  else
      stop the worker
  if the worker stopped and no other workers are working
                                      on the worker's batch,
      ask the Credit System to pay the order
```

Start_qos

Web service to enable QoS for a batch

→ The start_qos function is the first to be called, because this function will return the batch identifier that must be used for the other QoS related operations

- File: start_qos.py
- Arguments - Using the HTTP GET method:
  - dg_name – The grid technology used for the batch computation, as a string
  - → If some arguments are missing, a web form will be displayed.
- Displays a Web page containing:
  - `<div class="qos">batch_id</div>` with batch_id the batch identifier to use in future QoS operations
- Notes:
  - A database configuration file named "db.cfg" must be present in the directory.

3.4.3 Other functions

Functions used by the scheduler monitor_batches and monitor_cloud_workers programs

- File: scheduler_func.py

Configure_qos

configure_qos(batch_id, dg_type, dg_plugin_url, cw_id): Configure a Desktop Grid to start QoS

→ The configuration is a step performed once the requierment to start QoS are filled (ie. credit ordered, oracle agreement, cloudworker started, etc.)

- Arguments:
  - batch_id – The batch identifier, as a string
  - dg_type – The grid technology used for the batch computation, as a string. Currently, only BOINC needs some configuration

  - dg_plugin_url – The URL to the DG plugins, as a string.
  - cw_id – The cloud worker identifier, as an interger. Currently unused.

- The function returns True if the configuration succeed or None otherwise

Get_percent_completed

//get_percent_completed(batch_id): // *Remote call of Oracle's get_percent_completed*
The function calls and retrieves the result of the get_percent_completed function from the Oracle web server. It needs a file named "oracle.cfg" containing the line "address=<url>", with <url> the url to access to the Oracle web server, to be present in the directory.
Arguments: batch_id – The batch identifier, as a string
The function returns the batch completion ratio, as a float

Is_compute_finished

//is_compute_finished(cw_id, cw_tech, cw_host, batch_id, dg_type, dg_name): //

Preconfigure_QoS

preconfigure_QoS(batch_id, dg_type, dg_plugin_url): Preconfigure a Desktop Grid to be able to handle QoS

→ The preconfiguration is a configuration step that must be performed in the DG before the batch submission, i.e. when the user calls the start_qos function

- Arguments:
  - batch_id – The batch identifier, as a string
  - dg_type – The grid type used for the batch computation, as a string. Currently, only XWHEP needs some preconfiguration
  - dg_plugin_url – The URL to the DG plugins, as a string.
- The function returns some datas, as a string, specific to the DG technology, as output of the preconfiguration (for XWHEP its the group URI associated with the batch_id) or True if there is no specific data or None is the preconfiguration failed

Start_cloud_worker

start_cloud_worker(batch_id, dg_type, dg_name, batch_data, cw_id, cw_tech, cw_host): Start a cloud worker

→ According to the CloudWorker properties, such as the cloud technology it uses, this function load the appropriate implementation to start the CloudWorker. Then, according to the batch properties, such as the grid technologie it runs on, the function executes the appropriate commands on the CloudWorker to make it join the grid and participate to the batch computation

- Arguments:
  - batch_id – The batch identifier, as a string
  - dg_type – The batch grid technology, as a string. Currently, only "XWHEP" or "BOINC" are implemented.

  - dg_name – An unique name identifying the DG server. In the cloud worker, used to start the appropriate script (located to /root/spequlos/<dg_name>.sh)
  - batch_data – Some data associated to the batch, for DG technology-specific information, as a string (used to store the group URI associated to the batch in XWHEP), as a string
  - cw_id – The CloudWorker identifier, as an integer. Currently unused in this function.
  - cw_tech – The cloud technology used by the CloudWorker, as a string. Currently, only "EC2", "EUCA", or "DummyCloudWorker" are implemented
  - cw_host – The CloudWorker cloud host. Currently unused.
- The function returns some cloud technology specific data if the CloudWorker is correctly started (for EC2 or EUCA, the instance name is returned)

Stop_cloud_worker

stop_cloud_worker(cw_id, cw_tech, cw_host, cw_data): Stop a cloud worker

→ According to the CloudWorker properties, such as the cloud technology it uses, this function load the appropriate implementation to stop the CloudWorker.

- Arguments:
  - cw_id – The CloudWorker identifier, as an integer (currently unused in this function)
  - cw_tech – The cloud technology used by the CloudWorker, as a string. Currently, only "EC2", "EUCA", or "DummyCloudWorker" are implemented
  - cw_host – The CloudWorker cloud host (currently unused)
  - cw_data – Some data associated to the cloud worker, for technology-specific inforamtion (used to store the instance name for EC2 or EUCA)
- The function returns True if the CloudWorker is correctly stopped, or False otherwise

Unconfigure_QoS

//unconfigure_QoS(batch_id, dg_type, dg_plugin_url, batch_data, cw_id=-1, cw_data=''): *// Unconfigure a Desktop Grid when QoS operation finished*

→ The unconfiguration is a configuration step that must be performed in the DG when the QoS operations related to a batch is finnished, i.e. when the batch is completed or when no more resources are available to support QoS

- Arguments:
  - batch_id – The batch identifier, as a string
  - dg_type – The grid technology used for the batch computation, as a string. Currently, only XWHEP needs some unconfiguration
  - dg_plugin_url – The URL to the DG plugins, as a string.
  - batch_data – Some data associated to the batch, for DG technology-specific information, as a string (used to store the group URI associated to the batch in XWHEP), as a string
  - cw_id – The CloudWorker identifier, as an integer. Currently unused in this function.
  - cw_data – Some data associated to the cloud worker, for technology-specific information, as a string (used to store the instance name for EC2), as a string. Currently unused in this function.
- The function returns True if the unconfiguration succeed or None otherwise

3.5 Cloud Workers implementation

3.5.1 DUMMY Cloudworker implementation

Dummy Cloud Worker implementation

→ This Cloud Worker does nothing, it is only used for testing.

- File: dummy_cloudworker.py

3.5.2 G5K Cloudworker implementation

Cloud Worker implementation on G5K

→ Contains some function to start and stop Cloud Worker instance hosted on G5K, not using the libcloud API

- File: g5k_cloudworker.py
- Notes:
  - A file, named "g5k.cfg", with G5K credential (G5K_LOGIN and G5K_PASS), G5K site (G5K_SITE) and G5K instance information (G5_CW_IMAGE_ID) are needed to start the instances.
  - A SSH tunnel linking the port 22 of the G5K_SITE frontend to localhost on port 22222 must exists (`ssh -L 22222:frontend.G5K_SITE.grid5000.fr:22 G5K_LOGIN@access.G5K_SITE.grid5000.fr`)

3.5.3 EC2 Cloudworker implementation

Cloud Wroker implementation on Amazon EC2

→ Contains some function to start and stop Cloud Worker instance hosted on Amazon EC2, using the libcloud API

- File: ec2_cloudworkers.py
- Notes:
  - A file, named "ec2.amazon.cfg", with EC2 credential (EC2_ACCESS_KEY and EC2_SECRET_KEY) and EC2 instance information (EC2_CW_IMAGE_ID and EC2_CW_SIZE_ID) are needed to start the instances.
  - A file, named "id_rsa", containing the private key to connect with SSH to the cloud worker instance on Amazon EC2, must be present in the current directory.

3.5.4 EUCA Cloudworker implementation

Cloud Wroker implementation on Eucalyptus

→ Contains some function to start and stop Cloud Worker instance hosted on Eucalyptus, using the libcloud API

- File: euca_cloudworkers.py
- Notes:
  - A file, named "euca.cfg", with Eucalyptus host (EUCA_HOST and EUCA_PORT), credentials (EUCA_ACCESS_KEY and EUCA_SECRET_KEY) and instance information (EUCA_CW_IMAGE_ID and EUCA_CW_SIZE_ID) are needed to start the instances.
  - A file, named "id_rsa", containing the private key to connect with SSH to the cloud worker instance on Eucalyptus, must be present in the current directory.

4 Implementation details

SpeQulos is implemented using the Python programming language, and uses a MySQL database. The web server used for web services is Apache. In this section, we will present some details of the implementation.

4.1 Web services

We use CGI for all functions that need to be accessed remotely. The function arguments are passed through the GET method. If some arguments are missing, a web form will be displayed. Results returned by the function are displayed on an HTML page and are enclosed by ... tags (to be differenciated from other output).

All CGI related functions are defined in the tools/cgi_func.py file.

To query a remote function and to retrieve its results, we use the function call(url), defined in the tools/web_func.py file. This function returns a list of results as strings, for each elements enclosed by ... tags found in the web page accessed at the "url" argument.

For testing whitout deploying python files on a Web Server, it is needed to emulate the HTTP GET argument passing in the command line. For this, use the QUERY_STRING environment variable. Example: QUERY_STRING='batch_id=batch_0' ./get_credits.py

4.2 Implementation Tools

4.2.1 CGI tools

Help publishing results with CGI

→ This module contains various functions to help the publication of web page and services generated by CGI.

- File: cgi_func.py

Get_form_value

get_form_value(name='input'): Get the value from a web form using the HTTP GET method

- Arguments:
  - name — The name of the form, as described by the HTML "name" in the <form> tag. It will use "input" as name as a default argument
  - The function returns the form content
- Note: Special HTML character are escaped before being returned by this function

Print_footer

print_footer(): Print the footer of the HTML web page generated by CGI

Print_form

//print_form(inputs=['input']): // *Print a HTML code containing web forms*

→ The function prints a web form with several

```
<p>NAME <input type="text" name="NAME"/></p>
```

lines, with NAME for each line defined by argument

- Arguments:
  - inputs — A list of string, containing the name of the form, as described by the HTML "name" in the <form> tag. It will use a the ["input"] list as a default argument, displaying a single "input" form

Print_header

print_header(title="): Print the header of the HTML web page generated by CGI
- Arguments:
  - title — The window title, as a string. " is used as a default argument.

Print_message

//print_message(msg): *// Print a message on the Web page*
- Arguments:
  - msg — The message to print

Print_result

print_result(result): Print some information as it can be used as a result of the web function call

→ To be differentiated from other information on the web page, the result is enclosed by <`div class="qos"`>...</`div`> tags.

- Arguments:
  - result — The result to publish

4.2.2 CONF tools

Module for reading configuration files
- File: conf_func.py

Read_config

read_config(filename): Read a configuration file

→ This readers parse a configuration with pattern `option=value` on multiple lines.

- Lines starting with # or ; are ignored
- Arguments:
  - filename — The configuration file name, as a string
- The function returns a dictionnary containing the set of strings' (option,values) as read in the file

4.2.3 DB tools

Module for database related operation

- File: db_func.py
- Database used: MySQL

Connect

connect(params): Connect to a MySQL database

- Arguments:
  - params — The database configuration parameters, as a dictionary. It must contain the right values for database access in the keys: "host", "db", "user", "passwd".
- The function returns a MySQLdb.connections.Connection instance on the database connection

Disconnect

//disconnect(conn): // *Disconnect from a connected MySQL database*

- Arguments:
  - conn — The database connection, as a MySQLdb.connections.Connection

Read_db_config

read_db_config(filename): Read a database configuration file

- Arguments:
  - filename — The configuration file name, as a string
- The function returns a dictionary containing the keys: "host", "db", "user", "passwd", with the according values as read in the file.

4.2.4 LOG tools

Module for logging

- File: log_func.py

Get_spequlos_log

get_spequlos_log(name=None, html=True): Creates and configure logger for SpeQuloS logging

- Arguments:
  - name — The logger name as a string, should be the name of the module using the logger
  - html — If True (default), will print HTML log. Will print plain text otherwise
- The function returns a python logging.Logger instance

4.2.5 WEB tools

Module for web service operation

→ This module contains function to call and get results from a distant web page, generated by python CGI and publishing its results using the cgi_func module

- File: web_func.py

Call

//call(url): // Calls a distant function generated by python CGI and get its results.

- Arguments:
  - url — The URL of the distant function
- The function returns, as a list of strings, the results published on called web page and enclosed in `<div class="qos">...</div>` tags