GridCompute Documentation

Release 0.2

Boris Dayma

September 07, 2014

Contents

GridCompute is an open-source cross-platform tool that implements quickly distributed computing over a local grid.

The objective is to distribute resource intensive calculations over a local network to greatly reduce time needed, with a minimal work required for implementing it.

It can be particularly useful for engineering companies or universities.

Please see below useful links.

Documentation http://gridcompute.readthedocs.org

Source code http://github.com/borisd13/GridCompute

Binaries http://github.com/borisd13/GridCompute/releases

For any question, please contact the author **Boris Dayma** at boris@dayma.cc

CHAPTER 1

Contents

1.1 Overview

1.1.1 Description

GridCompute is a cross-platform tool that implements quickly distributed computing over a local grid.

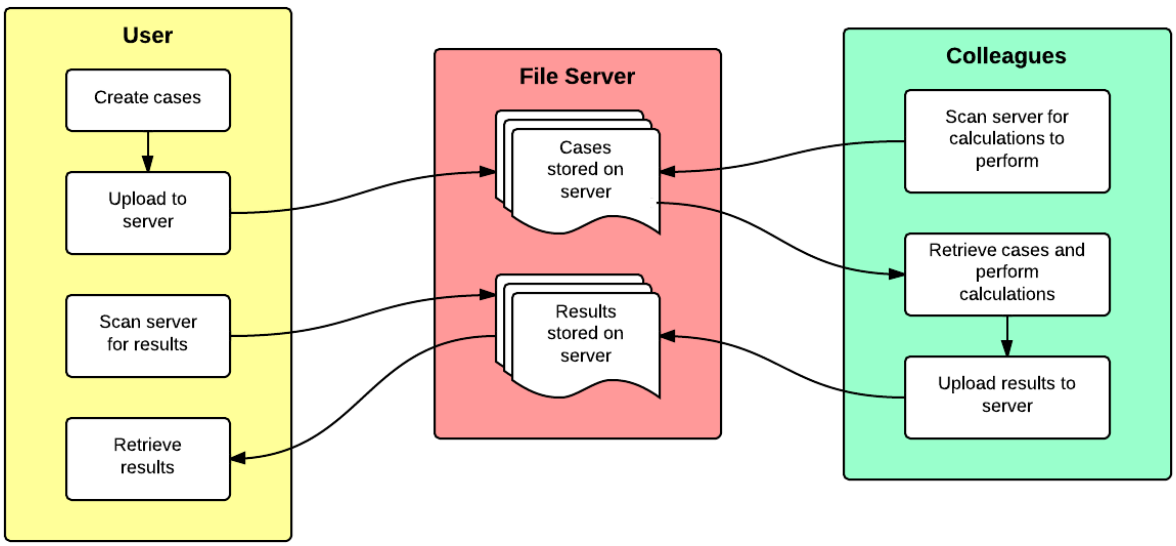
Gridcompute is easy to setup. You just need a shared folder and a database server. It can be adapted to any application through the use of python scripts.

1.1.2 Simple Workflow

The applicaiton follows the workflow:

- send calculations to the network
- scan network for calculations to perform, execute them and send results to network
- retrieve results

A very simplified workflow is represented herebelow:



For more details, refer to *Specifications*.

1.1.3 Applications of interest

Software can be used easily if:

- Calculations performed can be divided into multiple independent calculations, even if a final step is required to merge all results.
- Calculations can be started through the use of an external command line or python script.
- Calculations do not interfere with user interface, ie graphic, mouse, keyboard...

1.1.4 Interface functionalities

GridCompute interface has following functionalites:

- Submit cases to the network.
- Specify number of processes my computer can use to process cases from the network.
- Modify the number of processes during execution to pause, resume or cancel processes.
- Monitor status of user cases.
- Monitor status of user processes.
- Create a report of all the cases from same user group present on the network.
- Detect failed processes and retry to launch them.
- Specify which machines can perform calculations depending on application required.

1.2 Tutorial

Note: This section only intends to give a quick presentation of GridCompute through the use of a very simple demo application *Random Counter*.

This application takes a file as input, performs a countdown from a random number, and returns the time needed for that countdown, along with the name of the input file. This output is then added to a file in the home directory of the person who requested the process.

1.2.1 Create GridCompute database

GridCompute requires a MongoDB server to store all cases related information.

Create a MongoDB server

In this section, we are going to detail two possible scenarios: using a private server or using a specialized host.

Option 1: Create a private MongoDB server

In this example, we are going to assume that you want to use a Ubuntu server in a virtual machine created with VirtualBox. You can adapt it to install on a different OS and use of a virtual machine is only optional.

1. Install VirtualBox.
2. Create a virtual machine in VirtualBox with at least 512MB RAM and 8Go drive.
3. Install Ubuntu server. **Activate option "Open-SSH Server" during installation** for convenient access to your server.
4. Set up port forwarding in VirtualBox:
   - host=22 (or 2222 for Ubuntu host), guest=22 if you want to access to your machine by ssh
   - host=27017, guest=27017 for access to Mongo database
5. From host, connect to guest through ssh.

   On Ubuntu: `ssh -p 2222 GUEST_USERNAME@HOST_MACHINE_NAME`
   On Windows: you can use a program such as putty
6. Update Ubuntu packages and reboot.

```
>>> sudo apt-get update
>>> sudo apt-get upgrade
>>> sudo sudo reboot
```

7. Install MongoDB per instructions from MongoDB tutorial.
8. In file */etc/mongod.conf*, comment line *bindip [...]*
9. Run `sudo service mongod start`
10. Create a new database called *gridcompute*

```
>>> mongo
>>> use gridcompute
```

11. Add the user you will use from *gridcompute*. For example, for user *Group1* with password *gridcompute*:

```
>>> db.createUser({user:"Group1", pwd:"gridcompute", roles:["readWrite"]})
>>> quit()
```

12. Set up MongoDB server.

Option 2: Use a specialized host

MongDB server instance can be hosted by a specialized website such as Compose or Mongolab.

1. Register to the host of your choice.
2. Create a new database **named "gridcompute"**.
3. Add the user you will use from *gridcompute*. For example, user *Group1* with password *gridcompute*.
4. Set up MongoDB server.

Set up MongoDB server

The main purpose is only to define what version of gridcompute can work with the database.

Note: This section is optional. If you don't set up the database, a warning message will appear to notify you that the program version is not controlled by the server but the program will still work.

---

1. Open script *source/admin/database_management.py*.
2. Edit the variables under `if __name__ == "__main__"`.
3. Run the script with **python 3**.

1.2.2 Set-up of GridCompute

Before starting the program, you should follow these steps:

1. Download latest version of GridCompute binary. Choose the version associated to your OS.
2. Create a shared folder accessible (read and write) to every person that will use the program.
3. Copy folder *template/Shared_Folder* from source code to the shared folder you are going to use with *GridCompute*.
4. At the root folder of *GridCompute* executable, there should be a file *server.txt*. Open it and copy the path to *GridCompute* shared folder. Ex: `\\Server010\...\Shared_Folder`.
5. Edit *settings.txt* from the template folder with applicable parameters (refer to *settings.txt specifications*)
6. Edit *Software_Per_Machine.csv* from the template folder (for more details refer to *Software_Per_Machine.csv specifications*):

- Add the name of your machine in the first column
- Enter *1* for application *Random Counter* on the row corresponding to your machine name
- Save as csv

1.2.3 Test GridCompute

The following section will give you a brief overview of the program.

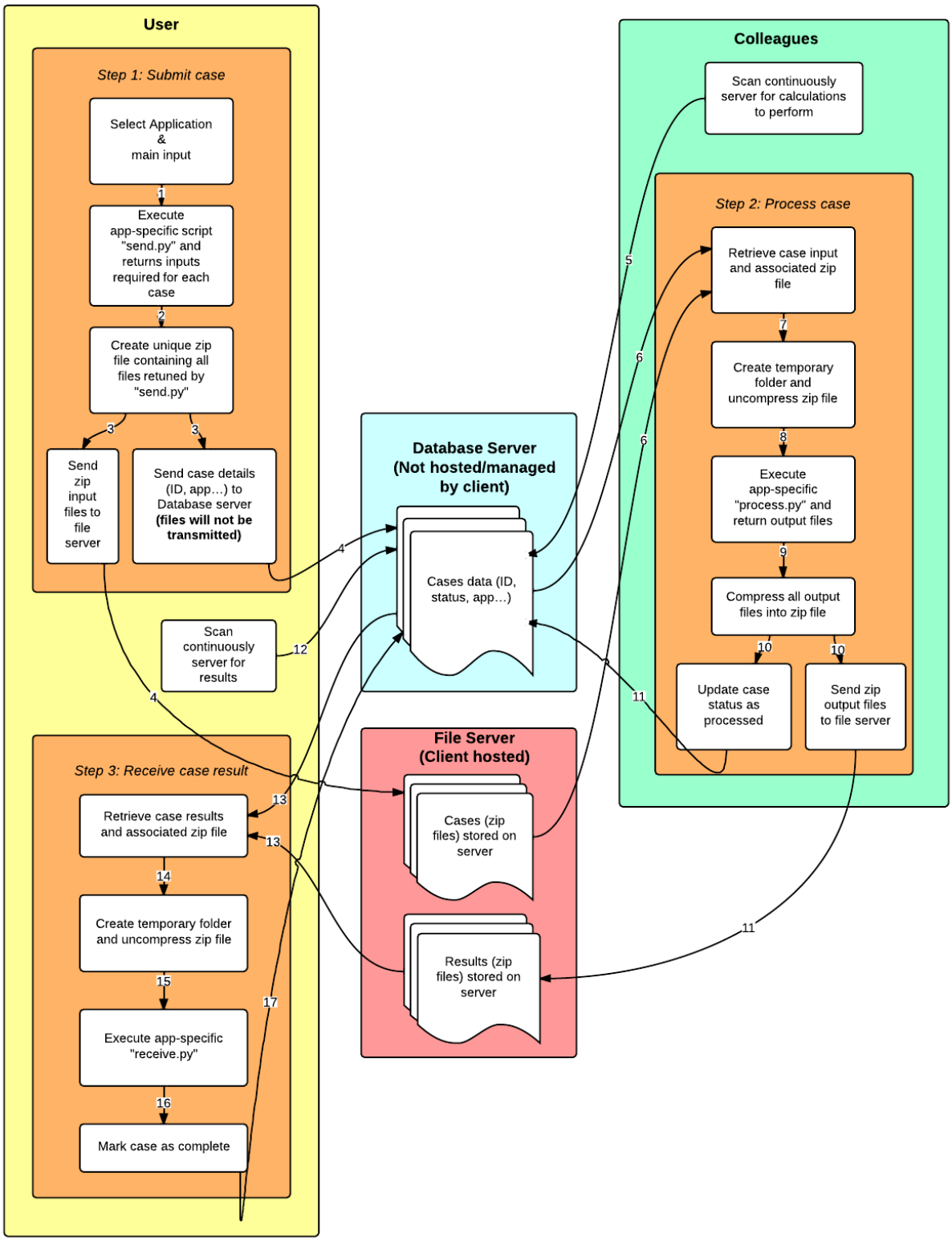
1. Run *GridCompute* executable. You will see the main window of the program.
2. Check at the bottom of the application that you can run *Random_Counter* as selected in *Software_Per_Machine.csv*.
3. Click on the application checkbox and select *Random Counter*.
4. Click on *add cases* and select 10-20 files (they will not be modified).
5. Check that cases have been added to the interface.
6. Click on *submit list to server* and confirm.
7. Click on the tab *my cases* and button *refresh now*. You will see the status of all your cases.
8. Increase the number of processes allowed to run.
9. While process are running, click on *refresh now* to see their status.
10. Go to the tab *my processes* to see what is happening on your computer.

11. *Random Counter* demo application has been set-up so that outputs are added to the file *gridcompute_output.txt* present in your home folder.

1.3 Specifications

1.3.1 Detailed workflow

The sequence of events is represented with numbered arrows herebelow:


User
Step 1: Submit case
Select Application & main input
1
Execute app-specific script "send.py" and returns inputs required for each case
2
Create unique zip file containing all files retuned by "send.py"
3
3
Send zip input files to file server
Send case details (ID, app…) to Database server (files will not be transmitted)
4
Scan continuously server for results
12
4
Step 3: Receive case result
Retrieve case results and associated zip file
13
13
14
Create temporary folder and uncompress zip file
15
Execute app-specific "receive.py"
17
16
Mark case as complete
Database Server (Not hosted/managed by client)
Cases data (ID, status, app…)
File Server (Client hosted)
Cases (zip files) stored on server
Results (zip files) stored on server
Colleagues
Scan continuously server for calculations to perform
5
Step 2: Process case
Retrieve case input and associated zip file
7
6
Create temporary folder and uncompress zip file
6
8
Execute app-specific "process.py" and return output files
9
Compress all output files into zip file
10
10
Update case status as processed
11
Send zip output files to file server
11

1.3.2 Architecture of shared folder

In the same folder as main script or program, there is a file named "server.txt" which contains only the network path of main shared folder between all users used by GridCompute (folder that contains "Settings", "Cases" and "Results"). Ex: `\\Server010\\Folder\\GridCompute`.

This shared folder has following architecture:

- *Settings* folder (to be at least read-only for users, read-write for administrators)
  - Software_Per_Machine.csv: This file contains machine names and corresponding applications installed (using unique ID per application). The program looks for the *1* in the matrix.

    Ex:

    ```
    Machine name,Software 1,Software 2,Software 3
    Machine 1,1,1,1
    Machine 2,0,1,0
    Machine 3,0,0,0
    ```

    Note: This file is only used to detect what machine can run *process* functions. All machines can submit or receive cases.

  - settings.txt: contains parameters of database on each line under the form: `parameter name: value`. Parameters to define are the following:
    - mongodb server: address of the mongo instance including connection port containing gridcompute database

      Ex: `mongodbserver.com:888` or `10.0.0.1:888` or `Machine123:888`
    - user group: Login used to connect on mongo database.
    - password: Password used to connect on mongo database.
    - instance: Data instance to consider. Example: `0` or `debug`.
  - *Applications* folder
    - One file per application with unique ID (ex: *Software 1*).

      Warning: The folder name cannot contain a dot.

      It contains:

      - send.py: Defines how to select input and send calculations.
      - process.py: Will run when analysis are executed based on input received and will create output files.
      - receive.py: Will run when output files are present on server.

      More details are provided in Application-specific scripts.
- *Cases* folder: Contains one folder per user and inside, one folder per machine (so that user can see easily his files) storing each case as a zip file that has all input files/folders required.
- *Results* folder: Contains one folder per user and inside, one folder per machine storing each result as a zip file that has all output files/folders required.

Note: A template folder is present in source code in *template* folder and can be used to set up the shared folder.

1.3.3 Architecture of Mongo Database

GridCompute communicates with a mongo database that contains all the details on cases. Following entries are present:

- collection *cases*
  - _id: Unique Object Id based on timestamp of the case.
  - user_group: User group. Ex: `ENGINEERING DEPARTMENT`.
  - instance: Instance used to isolate grids. Ex: `0` or `debug`.
  - status: Current status of the case. It can be `to process`, `processing`, `processed` or `received`.
  - last_heartbeat: Timestamp of last heartbeat sent to notify the database that the process is still alive.
  - application: Application associated to the case.
  - path: Path on file server refering to input/output case.
  - origin: Machine/User who submitted the case to the database.
    - machine
    - user
    - time
      - start: Time the case has been submitted to server.
      - end: Time the results have been retrieved from server.
  - processors
    - processor_list: List of Machine/Users who tried to process the case (some attempts to process may have failed).
      - machine
      - user
    - time (start and end) for the last attempt to process
      - start: Time of the last attempt to process.
      - end: Time the process returned.
- collection *versions* (optional)
  - _id: Versions of program recognized by database.
  - status: Can be either `allowed`, `warning` or `refused`.
  - message: Message to be displayed when status is not `allowed`.

1.3.4 Application-specific scripts

Applications can easily take advantage of distributed computing by creating 3 scripts, as detailed in following sections.

Note: Some examples are present in *template/Shared_Folder/Settings/Applications*.

send.py

This script is executed when submitting cases to server. It takes as input a file selected by the user and returns one or several cases to submit to the server.

send.**select_input_files**(*filepath*)
Submit a case to the grid.

This function returns, from a selected file, one or several cases to run. Each case can be made of several input files.

Parameters **filepath** (*str*) – Path of the file selected.

Returns str list: A list (or tuple) of cases. Each case is a list (or tuple) of input files required to process a case.

process.py

This script is used to process cases. Its input is the ordered list of files submitted in *send.py* script. At the end of execution, a list of output files is returned, which is submitted to the server.

process.**process_case**(*input_files*)
Process a case and return its results.

This function process a case from the grid and returns a list of output files that are sent back to the server. Process is executed in a temporary folder where all files are copied.

Parameters **input_files** (*str list*) – ordered list (or tuple) of input files path.

Returns str list: An ordered list (or tuple) of output files to return to the server.

receive.py

This script is used to receive cases that have been processed, ie to specify what we want to do with the output files returned from *process.py* script.

receive.**receive_case**(*output_files*)
Receive a case from the grid.

This function receives a case that has been processed on the grid. Process is executed in a temporary folder where all files are copied.

Parameters **output_files** (*str list*) – ordered list (or tuple) of output files path.

Returns None.

1.3.5 Main code layout

For details on *GridCompute* source code layout, refer to *Source code layout*.

1.4 Development

This section details how to use source code for developers. You can get latest code and binaries at: http://github.com/borisd13/GridCompute

1.4.1 Requirements

The following is required to use the source code:

- python version 3.4.0 or superior
- tkinter (should be included in python installation)
- pip for python3 (should be included in python installation)
- pymongo version 2.7.1 or superior
- psutil version 2.1.1 or superior

Install requirements on Ubuntu

Enter below commands in prompt:

```
>>> sudo apt-get install python3-pip
>>> sudo pip3 install pymongo
>>> sudo pip3 install psutil
>>> sudo apt-get install python3-tk
```

Install requirements on Windows

Follow below instructions:

1. download and install python 3.4.1 (64 bit version). Make sure to enable "Add python.exe to Path"
2. download and install pymongo
3. download and install psutil

1.4.2 Build application

GridCompute is meant to be run easily on computers that don't have any python environment. This is achieved by creating binaries of the application with the help of cx_Freeze.

Note: Building the application is not required for executing the program.

To build the application:

1. Use the OS you plan to build the application for.
2. Install cx_Freeze from source.

   Warning:
   On Ubuntu, you might need to apply fix for issue 32.
   On Windows, you may have to install Visual Studio C++ 2010.

3. Go to the main folder of GridCompute source code and run `python setup.py build`.

   Warning: On Ubuntu, you will also need to apply fix for issue 95.

1.4.3 Build documentation

Documentation is generated through the use of sphinx. To build documentation, you need to install following dependencies:

```
>>> pip3 install sphinx
>>> pip3 install sphinxcontrib-napoleon
>>> pip3 install sphinx-rtd-theme
```

You can then go to *docs* folder and run `make html`.

1.4.4 Execute application

From source, run **python 3** on *main.py*.

From executable, run *GridCompute* executable.

1.5 Source code layout

The source code of *GridCompute* is split into following modules:

1.5.1 main.py

This module is the start-up script used to launch *GridCompute*.

It first ensures that there is only one instance running, then creates the interface and initializes all parameters required to run the program. To finish, it runs the main loop associated to interface.

`main.`**`ensure_single_instance`**(*current_pid*, *event_queue*)
Ensure that only one instance of the program is running.

Check current pid against a configuration file that will contain pid of other running instance if any.

Parameters

- **current_pid** (*int*) – pid of current process.
- **event_queue** (*Queue*) – queue of events to process by gui.

`main.`**`start`**(*gui*, *current_pid*)
Start the program.

Ensures there is only one instance, initialize all server related parameters and populate interface.

Parameters

- **gui** (*GUI*) – GUI instance handling program interface.
- **current_pid** (*int*) – pid of main thread.

1.5.2 g_config.py

This module gathers configuration variables used in the other modules.

It is used to share global variables and edit configuration variables easier.

g_config.**program_name**
Name of the program.

g_config.**version**
Current version.

g_config.**author**
Author of the program.

g_config.**copyright**
Copyright.

g_config.**title_windows**
Title displayed on windows.

g_config.**max_number_process**
Maximal number of parallel process that can be executed on this computer.

g_config.**gui_refresh_interval**
Refresh rate of interface in milliseconds.

g_config.**db_connect_frequency**
Time in seconds before database is accessed again if no case/result is found.

g_config.**db_heartbeat_frequency**
Frequency in seconds that heartbeat are sent on running processes to notify database that they are still alive.

g_config.**db_heatbeat_dead**
Time in seconds without heartbeat after which we consider a process is dead.

g_config.**daemon_pause**
Time in seconds between each process of daemons.

g_config.**log_path**
Path of the log file.

g_config.**pid_file**
Path of the file keeping pid of the program to ensure there is only one single instance running.

1.5.3 g_interface.py

This module contains all GUI functionalities.

It generates the main interface and handles any event that needs to be communicated to the user.

class g_interface.**GUI**
Class handling GridCompute interface.

It contains every parameter and module required for GUI. At creation, it displays a progress bar.

Parameters

- **root** – main Tk instance
- **application** – application selected in gui
- **app_combobox** – widget associated with application
- **cases_dict** (*dict*) – dictionary where key is the id of tree item and value is the list of input files associated to the case
- **cases_refresh_label** – label identifying time of refresh of "my cases"
- **cases_status_label** – label identifying status of refresh of "my cases"

- **dedicated_process** – number of dedicated process selected in gui
- **event_queue** – queue of events to process by gui
- **log** – gui element associated to logging
- **my_process_pid_id** (*dict*) – dictionary of pid to gui id for treeview in "my processes"
- **progress_bar** – progress bar in progress window
- **progress_label** – text present on progress window
- **progress_window** – top level window showing progress of current task set to None if progress window not existing or closed
- **server** – Server instance containing main functionalities
- **tree_cases** – widget containing cases to submit
- **tree_my_cases** – widget containing "my cases"
- **tree_my_process** – widget containing "my processes"

add_cases()
Display a window to select cases to submit to server and adds them on the list.

askokcancel(*msg*)
Ask a question to user.

Parameters **msg** (*str*) – Message to be displayed.

Returns True if user enters "ok", False otherwise.

Return type bool

create_progress_window(*progress_mode*, *progress_text*, *progress_max=100*)
Create a progress window.

The progress window can be controlled afterwards through the event_queue parameter. Refer to function "handle_next_event".

Parameters

- **progress_mode** (*str*) – "determinate" if progress level evolution needs to be controlled or "indeterminate".
- **progress_text** (*str*) – text displayed initially on the progress window.
- **progress_max** (*int*) – argument equal to maximal progress value when complete.

create_report()
Create a full report from database containing all cases from same "user group". Ask user where he wants to save the report.

error(*msg*)
Display error on screen.

Parameters **msg** (*str*) – Message to be displayed.

exit_program()
Communicate to processes to exit program.

handle_next_event()
Handles the next event from communicated to the GUI through event_queue variable.

Each element in event_queue is a dictionary. Action performed depends on value of *type* key:

•log_file_only: log a message in the log file, no display in gui

•warning: display a warning

•info: display an information in gui, optionally creating an info box

•error: display an error

•critical: display an error and exit program

•change progress max: change value of progress window corresponding to completion

•change progress: modify progress bar level and text

•close progress: close progress window

•add case: add a case in "send cases" tab

•submitted case: show a case as submitted

•terminate process?: ask user if he wants to terminate all processes

•send answer through the connection pipe to daemon process

•add my case: add a case in "my cases" tab

•add my process: add a process in "my processes" tab

•remove my process: remove a process from "my processes" tab

•change my process: change the status of a process in "my processes" tab

•populate: GUI can be fully populated

•exit: close main window

Raises **queue.empty: event_queue is empty, there are no more events to process.** –

info(*msg*)
Display info on screen

Parameters **msg** (*str*) – Message to be displayed.

link_server(*server*)
Associate GUI to a Server instance.

Parameters **server** – Server instance.

open_about()
Display program information.

open_help()
Display a help message.

open_license()
Display licenses used by the program.

populate()
Populate GUI during loading of program.

refresh()
Refresh GUI at regular intervals.

refresh_my_cases()
Refresh the list of "my cases".

Access database and display details of user cases that have not been received yet.

remove_selected()
Removes selected cases to submit from the list.

submit_to_server()
Send all cases (not yet submitted) from the list to the server.

warning(*msg*)
Display warning on screen

Parameters **msg** (*str*) – Message to be displayed.

g_interface.**init_log**()
Initialize logging in file.

Create necessary directory structure and remove previous log file.

g_interface.**write_log**(*log_message*, *gui_log=None*)
Write log messages in log file and in GUI.

Parameters

- **log_message** – message to be displayed.
- **gui_log** – widget receiving log message. None if message not to be displayed in gui.

1.5.4 g_server_management.py

This module contains all server related functionalities.

It creates server variables associated to file server and mongo database and handle the communication with them.

class g_server_management.**Server**(*event_queue*, *dedicated_process*)
Class handling server-related functionalities.

File server path is taken from "server.txt" present at root of program. Settings are taken from "settings.txt" present on the file server.

Server properties are set at initialization.

Parameters

- **event_queue** – queue of events to process by gui
- **gui_dedicated_process** – number of dedicated process selected in gui
- **daemon_dedicated_process** – number of dedicated process communicated to daemon process
- **exit_program** – variable scanned by daemons to know when to exit
- **gui_answer** – notify when gui answered a question
- **server_path** – path of file server as defined in "server.txt" file
- **app_path** – path of applications-specific scripts
- **settings_path** – path of settings.txt file
- **settings** – list of settings from settings.txt on file server including:
  - mongodb server: Address of the mongo instance including connection port containing "gridcompute" like `mongodbserver.com:888` or `10.0.0.1:888` or `Machine123:888`.
  - user group: Login used to connect on mongo database.

  - password: Password used to connect on mongo database.
  - instance: Data instance to consider like `0` or `debug`.
- **mongodb** – Connection to mongo database
- **server_functions** – Dictionary of application-specific scripts. The key is application name and value is a dictionary of following keys:
  - path: path of application folder
  - send, process, receive: booleans corresponding to existence of these functions
- **software_allowed_to_run** – set of applications that can run as defined in "Software_Per_Machine.csv"

access_mongodb()
Access the mongo database.

add_cases(*files_selected*, *application*, *keep_running=True*)
Add a list of cases to interface from files selected by user.

A file selected can correspond to one or several cases to run. Each case being a list of input files. Those cases are sent to the interface which displays the first input file of each case. The application specific "send.py" function is used on each file selected.

Parameters

- **files_selected** – Files selected by user
- **application** – Application associated to input files
- **keep_running** – Argument that can be associated to the state of a variable. When this variable is False, function ends. It is used to catch when user closes the progress window.

applications_with_process()
Return a dictionary of applications that have a process function.

The key is the application name and the value is the script path.

applications_with_receive()
Return a dictionary of applications that have a receive function.

The key is the application name and the value is the script path.

applications_with_send()
Return a dictionary of applications that have a send function.

The key is the application name and the value is the script path.

create_daemons()
Create daemons required in the application.

Two daemons are created:

•Daemon process: scans continuously database to check if there are new calculations to perform (if number of processes selected allows it).

•Daemon receive: scans continuously database to check if there are new results to receive.

create_report(*file_report*, *keep_running=True*)
Create a report of cases present on database.

Display cases and their details from database, limited to the ones from same "user group".

Parameters **keep_running** – Argument that can be associated to the state of a variable. When this variable is False, function ends. It is used to catch when user closes the progress window.

exit_processes()
Terminate program.

Ensure daemon process (and its child processes) and daemon receive terminate properly.

get_settings()
Initialize settings variable from "settings.txt" present on server.

get_software_allowed_to_run()
Return the list of software allowed to run on this machine per "Software_Per_Machine.csv" present on file server.

handle_software_permissions()
Verify that mongo database allows current version of program to run.

This is based on the collection "versions" present in mongo database. If "versions" collection is not present, a warning is displayed.

notify_number_process_daemon(**args*)
Notify the daemon process of a change of number of processes selected in GUI.

refresh_my_cases(*keep_running=True*)
Refresh "my cases" list.

Access mongo database to display all user cases that have not been totally processed yet (ie not received by user).

Parameters **keep_running** – Argument that can be associated to the state of a variable. When this variable is False, function ends. It is used to catch when user closes the progress window.

scan_applications()
Scan application specific scripts present on file server.

submit_to_server(*cases_to_submit*, *application*, *keep_running=True*)
Submit cases to the server.

Input files are zipped and copied to the file server. Cases are entered in mongo database.

Parameters

- **cases_to_submit** – List of cases. Each case is a list of input files associated to a case.
- **application** – Application associated to input files.
- **keep_running** – Argument that can be associated to the state of a variable. When this variable is False, function ends. It is used to catch when user closes the progress window.

g_server_management.**check_quit_program**(*exit_program*)
Function used by daemons to check if they need to terminate and kill their child processes.

Parameters **exit_program** – Variable scanned by daemons to know when to exit.

g_server_management.**launch_process**(*case*, *event_queue*, *server_path*, *settings*)
Launch one process from database.

Mongo database is updated when process finishes.

Parameters

- **case** – case to process obtained from mongo database.
- **event_queue** – queue of events to process by gui.
- **server_path** – path of file server as defined in "server.txt" file.
- **settings** – list of settings from settings.txt on file server.

g_server_management.**refresh_status_daemon_process**(*alive_process*, *dedicated_process*, *event_queue*, *exit_program*, *gui_answer*, *paused_process*)

Refresh status variables used by daemon process.

Parameters

- **alive_process** – List of processes alive.
- **dedicated_process** – Number of dedicated process selected in gui.
- **event_queue** – Queue of events to process by gui.
- **exit_program** – Variable scanned by daemons to know when to exit.
- **gui_answer** – Notify when gui answered a question.
- **paused_process** – List of processes currently on pause.

g_server_management.**return_module**(*application*, *function*)

import app-specific function send, process or receive

g_server_management.**run_daemon_process**(*applications_with_process*, *dedicated_process*, *event_queue*, *exit_program*, *gui_answer*, *server_path*, *settings*, *software_allowed_to_run*)

Run "daemon process" that launches new processes when possible.

Daemon process runs continuously and check on database if processes are available to launch when user allows it.

Parameters

- **applications_with_process** – dictionary of applications that have a process function
- **dedicated_process** – number of dedicated process selected in gui
- **event_queue** – queue of events to process by gui
- **exit_program** – variable scanned by daemons to know when to exit
- **gui_answer** – notify when gui answered a question
- **server_path** – path of file server as defined in "server.txt" file
- **settings** – list of settings from settings.txt on file server
- **software_allowed_to_run** – set of applications that can run as defined in "Software_Per_Machine.csv"

g_server_management.**run_daemon_receive**(*applications_with_receive*, *event_queue*, *exit_program*, *server_path*, *settings*)

Run "daemon receive" that retrieve back cases from server after they have been processed.

Daemon receive runs continuously and check on database if output files are available.

Parameters

- **applications_with_receive** – dictionary of applications that have a process function.
- **event_queue** – queue of events to process by gui.
- **exit_program** – variable scanned by daemons to know when to exit.
- **server_path** – path of file server as defined in "server.txt" file.
- **settings** – list of settings from settings.txt on file server.

1.5.5 setup.py

This script is used for building the executuble with cx_Freeze.

Please refer to the section "Development" of the documentation for building the application.

1.5.6 database_management.py

This module contains administrator functions for database management.

database_management.**set_up_mongodb_server**(*mongodb_server*, *login*, *password*, *versions*)

Sets up a mongodb server for GridCompute.

Mongo database "gridcompute" is initialized and the "versions" collection is created to specify the program versions that are authorized by the database.

The "gridcompute" database must be present on the server. Any collection in it will be removed.

Parameters

- **mongodb_server** – Address of the mongo instance including connection port containing *gridcompute* database like `mongodbserver.com:888` or `10.0.0.1:888` or `Machine123:888`
- **login** – Login used to connect on mongo database.
- **password** – Password used to connect on mongo database.
- **versions** – List of versions of gridcompute that the mongo database recognizes defined by:
  - _id: version number (ex: '0.1').
  - status: either "allowed", "warning" or "refused".
  - message: message to be displayed when status is not "allowed" like:

```
[{'_id':'0.1', status:"warning", message:"Beta version},
 {'_id':'1.0', status:"allowed"}]
```

1.6 License

1.6.1 GridCompute

```
Copyright 2014 Boris Dayma

GridCompute is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, version 3 of the License.

GridCompute is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with GridCompute.  If not, see <http://www.gnu.org/licenses/>.

For any question, please contact Boris Dayma at boris.dayma@gmail.com
```

1.6.2 Third-party tools

Please note that this program uses 3rd party tools whose licenses have been reproduced herebelow.

Python

```
Copyright © 2001-2014 Python Software Foundation. All rights reserved.
Copyright © 2000 BeOpen.com. All rights reserved.
Copyright © 1995-2000 Corporation for National Research Initiatives. All rights reserved.
Copyright © 1991-1995 Stichting Mathematisch Centrum. All rights reserved.


PSF LICENSE AGREEMENT FOR PYTHON 3.4.1

This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and the Individual or Organ
Subject to the terms and conditions of this License Agreement, PSF hereby grants Licensee a nonexclus
In the event Licensee prepares a derivative work that is based on or incorporates Python 3.4.1 or any
PSF is making Python 3.4.1 available to Licensee on an "AS IS" basis. PSF MAKES NO REPRESENTATIONS OR
PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 3.4.1 FOR ANY INCIDENTAL, SPECIAL, O
This License Agreement will automatically terminate upon a material breach of its terms and condition
Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership,
By copying, installing or otherwise using Python 3.4.1, Licensee agrees to be bound by the terms and
```

Pymongo

```
Copyright 2008 - 2014, MongoDB, Inc

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.

You may obtain a copy of the License at
http://www.apache.org/licenses/LICENSE-2.0
```

Zlib

```
Copyright (C) 1995-2011 Jean-loup Gailly and Mark Adler

This software is provided 'as-is', without any express or implied
warranty.  In no event will the authors be held liable for any damages
arising from the use of this software.

Permission is granted to anyone to use this software for any purpose,
including commercial applications, and to alter it and redistribute it
freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not
   claim that you wrote the original software. If you use this software
   in a product, an acknowledgment in the product documentation would be
   appreciated but is not required.

2. Altered source versions must be plainly marked as such, and must not be
   misrepresented as being the original software.

3. This notice may not be removed or altered from any source distribution.
```

```
Jean-loup Gailly        Mark Adler
jloup@gzip.org          madler@alumni.caltech.edu
```

Cx_Freeze

```
* Copyright © 2007-2014, Anthony Tuininga.
* Copyright © 2001-2006, Computronix (Canada) Ltd., Edmonton, Alberta, Canada.
* All rights reserved.

NOTE: this license is derived from the Python Software Foundation License
which can be found at http://www.python.org/psf/license

License for cx_Freeze
---------------------

1. This LICENSE AGREEMENT is between the copyright holders and the Individual
   or Organization ("Licensee") accessing and otherwise using cx_Freeze
   software in source or binary form and its associated documentation.

2. Subject to the terms and conditions of this License Agreement, the
   copyright holders hereby grant Licensee a nonexclusive, royalty-free,
   world-wide license to reproduce, analyze, test, perform and/or display
   publicly, prepare derivative works, distribute, and otherwise use cx_Freeze
   alone or in any derivative version, provided, however, that this License
   Agreement and this notice of copyright are retained in cx_Freeze alone or in
   any derivative version prepared by Licensee.

3. In the event Licensee prepares a derivative work that is based on or
   incorporates cx_Freeze or any part thereof, and wants to make the derivative
   work available to others as provided herein, then Licensee hereby agrees to
   include in any such work a brief summary of the changes made to cx_Freeze.

4. The copyright holders are making cx_Freeze available to Licensee on an
   "AS IS" basis.  THE COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES,
   EXPRESS OR IMPLIED.  BY WAY OF EXAMPLE, BUT NOT LIMITATION, THE COPYRIGHT
   HOLDERS MAKE NO AND DISCLAIM ANY REPRESENTATION OR WARRANTY OF
   MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF
   CX_FREEZE WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. THE COPYRIGHT HOLDERS SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF
   CX_FREEZE FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS
   A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING CX_FREEZE, OR ANY
   DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material breach
   of its terms and conditions.

7. Nothing in this License Agreement shall be deemed to create any relationship
   of agency, partnership, or joint venture between the copyright holders and
   Licensee.  This License Agreement does not grant permission to use
   copyright holder's trademarks or trade name in a trademark sense to endorse
   or promote products or services of Licensee, or any third party.

8. By copying, installing or otherwise using cx_Freeze, Licensee agrees to be
   bound by the terms and conditions of this License Agreement.
```

```
Computronix® is a registered trademark of Computronix (Canada) Ltd.
```

Psutil

```
psutil is distributed under BSD license reproduced below.

Copyright (c) 2009, Jay Loden, Dave Daeschler, Giampaolo Rodola'
All rights reserved.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

 * Redistributions of source code must retain the above copyright notice, this
   list of conditions and the following disclaimer.
 * Redistributions in binary form must reproduce the above copyright notice,
   this list of conditions and the following disclaimer in the documentation
   and/or other materials provided with the distribution.
 * Neither the name of the psutil authors nor the names of its contributors
   may be used to endorse or promote products derived from this software without
   specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR
ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

 * Redistributions of source code must retain the above copyright notice, this
   list of conditions and the following disclaimer.
 * Redistributions in binary form must reproduce the above copyright notice,
   this list of conditions and the following disclaimer in the documentation
   and/or other materials provided with the distribution.
 * Neither the name of the psutil authors nor the names of its contributors
   may be used to endorse or promote products derived from this software without
   specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR
ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

Many thanks to below people who contributed to the development of the program:

- Bruno Guigon

- Vianney Da Costa

d

g

m

p

r

s