qpy - the **P**radipta's and **Y**uri's **PY**thon **Q**ueue system

User, Administrator and Developer manual

version X.X

Conceived and created by Pradipta Kumar Samanta and Yuri Alexandre Aoto

With the kind help and support of Andreas Köhn and Arne Bargholz

October 3, 2018

Contents

1 For users

QPY is used to submit jobs from a main server and execute them in different nodes, according the availability or the choices given by the user. It can handle jobs running in both single and multi processors and it has several options to make a friendly user interaction. If you are using QPY for the first time, please read carefully the section 1.1, *Basics*, just bellow. For a complete list of commands and their options, see the section 1.2, *Commands*.

1.1 Basics

1.1.1 Installation

The main instalation will probably be done by your system administrator. You might have to do, however, the following thing:

Put the following lines in your ∼/.bash_profile or ∼/.bashrc:

```
1 export PATH=<qpy_dir>:$PATH
2 source <qpy_dir>/bash_completion.sh
```

Your system administrador should tell you what is the <qpy_dir> directory. (S)He will might also send you a few files to be placed in your QPY directory, ∼/.qpy/. It's in this directory that

1.1.2 Initialisation

The first QPY command one should run is:

```
1 $ qpy restart
```

This command will set the background environment for using QPY. It should also be executed when some new update is done (you will be informed) or if the master node crashes.

1.1.3 Basic concepts and usage

Every job submmited by QPY has an ID and a status. The ID, a fixed integer for a particular job, is unique and identifies your job in the pool of jobs. The status are:

- queue - the job is waiting for allocation
- running - the was allocated and is running
- done - the job have run and is finished

- killed - the job had its execution killed by the user
- undone - the job have not been executed (it was killed while in the queue)

When you just submit a job, it has the 'queue' status. When a node is allocated for it, it is executed and the status changes to 'running'. If the job terminates normally, the status changes to 'done'. If you kills the job (that is, stops its execution), the status changes to 'killed'. If you kills a job that is still on the queue, it will never be executed and the status changes to 'undone'.

You can submit a job using the sub command:

```
1 $ qpy sub <job's execution command>
```

This means that you put, on your queue of jobs, a new job that has to be executed with the command ¡job's execution command¿. QPY will handle it for you from here on!

By the way, all commands in QPY are like this:

```
1 $ qpy <cmd> [options]
```

where ¡cmd¿ is one of the QPY commands and [options] are the otions for that command. Of course, these options depend on the actual command ¡cmd¿, some of them are optional, and so on. See below for a complete list of commands and the explanation of their several options.

Now that your job is on the list, you probably want to be able to: - check the status of your job (or jobs); - check the situation of the nodes; - eventually kill a unsatisfactory job.

This can be done by the following commands:

```
1 $ qpy check
```

This shows a list of all your jobs. If you are interested only on a portion of these jobs, say, the ones that are currently running, try:

```
1 $ qpy check running
```

Another useful QPY command is:

```
1 $ qpy status
```

It will show all the available node, users, how many jobs each one is running and so on. If your jobs are for too long on the queue, this command should provide an indication why.

Sometimes we are not so happy with the development of a job and we want to terminate its execution, without waiting to its normal termination. You can do this with:

```
1 $ qpy kill <job_ID>
```

where ¡job_ID¿ is the ID number of the job. Attention! This is not reversible. QPY will not ask you if you are sure to kill the job, so use the command with care.

There are several commands and options in QPY. Every command in QPY can be completed automatically by using the TAB key. It also gives the possible next arguments which, we hope, would be very helpful for all the users :) For instance, if you type:

```
$ qpy kill ? <TAB> <TAB>
```

where `<TAB> <TAB>` indicates that you should type the `TAB` key two times, you will receive a brief explanation on the command sub.

1.2 Commands

The followings are all the commands of QPY. The user can run any of these commands as

```
$ qpy <cmd> [options]
```

where `[options]` depends on the actual command `<cmd>`.

1.2.1 restart

The user can start QPY background environment with this command. This command also works as an easy way to finish current session of QPY and start a new, which is most useful when the user wants to update the QPY to its latest version.

- Options: there are no options.
- Examples:

  Restart the QPYbackground environment:

  ```
  $ qpy restart
  ```

1.2.2 finish

If you feel like done using QPY, please finish the existing QPY environment which is running in background using this command. If somehow the qpy-master crashes while using, please remove the '.port' file from the QPY directory before starting a new QPY session.

- Options: there are no options.

- Examples:

  Finishes the background QPY environment:

```
1 $ qpy finish
```

1.2.3 sub

Submit any executable after the command 'qpy sub'. The output and the error file of running that particular script will be written in the files 'job_¡job_ID¿.out' and 'job_¡job_ID¿.err' respectively.

Submitting a serial job is the default one for this command. To submit a job that uses multi-processor, please add the information of number of processors (N) needed either by putting '-n N' between 'qpy sub' and the executable, or by adding '#QPY n_cores=N' in the script running. For the first case, do not put the option after the executable as it would then be considered as an option corresponding to the executable.

QPY also check if enough memory is available in a node before starting a job there. The defualt memory it checks for availibility is 5 GB. If a job needs to use more memory (M) please add this information either by putting '-m M' between 'qpy sub' and the executable, or by adding '#QPY mem=M' in the script running.

- Options: the executable and its arguments; optionally, the the following flags BEFORE the executable:

  -n ¡n cores¿ the number of cores -m ¡memory¿ requested memory, in GB

- Examples:

  Executes the command 'hostname'. Allocates one cores and 5 GB of memory for it (default values):

```
1 $ qpy sub hostname
```

  Executes the script './script.sh'. Allocates three cores and 10 GB of memory for it:

```
1 $ qpy sub -n 3 -m 10 ./script.sh
```

  Executes the command 'ls -ltr':

```
1 $ qpy sub ls -ltr
```

1.2.4 check

Check the status of all the submitted jobs. The command 'qpy check' gives the list of all jobs submitted. One can also check jobs of specific status which are: 'running', 'done', 'killed', 'undone' and 'queue' adding the keyword after 'qpy check'

- Options: a status a job ID list
- Examples:

1.2.5 kill

Kill a particular jobs using the command 'qpy kill [jobid]'. One can also kill several jobs by providing a list of jobid's or a range of jobid's separated by '-'. Kill 'all'/'running' jobs or jobs in 'queue' by replacing the [jobid] with these keywords.

- Options: a status ('queue' or 'running')

  a job ID list

  'all'
- Examples:

1.2.6 status

This command is mainly useful for the 'multi-user' environment. It writes elaborately the number of jobs running for each users and in each of the available nodes.

- Options: There are no options.
- Examples:

1.2.7 config

This command gives informations about some of the variables in QPY. Feel free to check. It is quite self explanatory. One of the advanced feature of this command is to configure the format that 'qpy check' would follow. The default format that 'qpy check' uses to write all the jobs is '

It might seem little complicated, but it follows mainly these rules:

- %j : job ID

- %s : job status
- %c : command you used to submit the job
- %d : working directory of your job
- %n : node allocated for your job
- %N : number of cores for your job.
- %R : running time of the job. (time in queue if the job is in queue)
- %Q : actual time when the job is submitted
- %S : actual time when the job has started
- %E : actual time when the job has finished

The user can also change this format to a minimalistic one, by 'qpy config checkFMT %j:%s' or anything using the notations defined above. To check the current format, use the command 'qpy config checkFMT' and to change it to the default one, use 'qpy config checkFMT default'.

There are some new options available to format 'qpy check' and have some information about the timing of the jobs. These are:

Bash scripts, generally, stops if changes are made in it while it is still running. So it is better to have QPY running a copied script which allows to change and reuse the original bash script. 'qpy config' provides another advanced option to do this by copying the script to the local folder ' /.qpy/scripts' and use it later while running the actual calculation. This is very useful while working with a bash script. To use this, set it up with 'qpy config copyScripts true', where by default 'copyScripts' is set to 'false'.

The same thing can also be done for a specific job if the submitted scipt has the line '#QPY cpScript true' or if the job is submitted using the command 'qpy sub -c [script_name]'. If the 'copyScripts' is set True, then the reverse can be done by adding the line '#QPY cpScript false' in the script or by submitting the script using the command 'qpy sub -o [script_name]'.

- Options:
- Examples:

1.2.8 ctrlQueue

ctrlQueue is a command to control and move the jobs in the queue. Consider the situation where you have a long list of jobs in the queue but you have some job(s) of top priority that needs to be done as soon as possible. The command ctrlQueue can be used in this situtation to move the priority job(s) up in the queue-list. These are what you have to follow:

'qpy ctrlQueue pause' : This command needs to be used prior to moving the jobs. This command ensures that no jobs from the queue can be started further.

'qpy ctrlQueue jump [job-id-1] [job-id-2] [job-id-3] ... [job-id-n] [job-id-pos]' : This is the main command for moving the jobs. Here all the jobs (1 to n) are moved in the queue to be placed after [job-id-pos] in the queue-list.

'qpy ctlrQueue continue' : This command remove the pause and let QPY to continue putting jobs from queue to the nodes.

- Options:
- Examples:

1.2.9 clean

It cleans the list of jobs that one can see using 'qpy check'. The user can clean some specific jobs by adding the job id or the specific status of the job. The jobs that can be cleaned are 'done', 'killed' and 'undone'.

- Options: a status ('done', 'killed' or 'undone') a job ID list 'all'
- Examples:

1.2.10 tutorial

Shows a tutorial.

- Options:
- Examples:

2 For administrators

The duties of the administrator of QPY are basic to maintain the program "qpy-multiuser" running and add/remove new users and machines. This section explains how this is done.

2.1 Installation

We assume that you have a copy of QPY, otherwise you wouldn't be reading this manual. To set up a new QPY environment, create the following directory in the administrator's home directory:

```
1 $ mkdir ~/.qpy-multiuser/
```

And create the following files:

`~/.qpy-multiuser/distribution_rules`

`~/.qpy-multiuser/allowed_users`

`~/.qpy-multiuser/nodes`

`~/.qpy-multiuser/multiuser_connection_address`

The content of these files are described in section 2.6. Create them according to your local needs. After this, start the qpy-multiuser with:

```
1 $ python ${QPY_DIR}/qpy-access-multiuser.py start
```

2.2 Add new user

As QPY usually runs across several machines, make sure that `ssh` keys are set up!

To add a new user in QPY, first add his name in the file

`~/.qpy-multiuser/allowed_users`

After, you have to send the following files to the new user:

`~/.qpy-multiuser/multiuser_connection_address`

`~/.qpy-multiuser/multiuser_connection_port`

`~/.qpy-multiuser/multiuser_connection_conn_key`

These should be placed in the user's QPY directory:

```
${USER}/.qpy/multiuser_connection_address
${USER}/.qpy/multiuser_connection_port
${USER}/.qpy/multiuser_connection_conn_key
```

In addition, if you have a dedicated machine to run QPY, add this file to the user's QPY directory (note that the above filenames start with `multiuser_*` while this filename starts with `master_*`):

```
${USER}/.qpy/master_connection_address
```

with the address of the machine where his qpy-master is supposed to run. If this file is not present, QPY runs locally.

2.3 Add and remove a node

To add a new node, add a new line in the file

```
~/.qpy-multiuser/nodes
```

with the information about the node (see section 2.6 for a description of this file). Then run:

```
$ python ${QPY_DIR}/qpy-access-multiuser.py nodes
$ python ${QPY_DIR}/qpy-access-multiuser.py distribute
```

2.4 Update version

When a new version of QPY is released, depending where the new changes have been done, you might have to restart the qpy-multiuser:

```
$ python ${QPY_DIR}/qpy-access-multiuser.py finish
$ python ${QPY_DIR}/qpy-access-multiuser.py start
```

And/or ask all the users to restart their qpy-masters. Whatever is the case, this will be informed in the release's notes.

2.5 Commands

This is a full list of the commands that are available to the administrator:

```
1 $ python ${QPY_DIR}/qpy-access-multiuser.py start
```

Starts a new qpy-multiuser instance. Run this if you are starting QPY for the first time, if a update is needed or if the machine qhere QPY runs has crashed.

```
$ python ${QPY_DIR}/qpy-access-multiuser.py finish
```

Finishes the qpy-multiuser program

```
$ python ${QPY_DIR}/qpy-access-multiuser.py nodes
```

Loads the content of the `nodes` file.

```
$ python ${QPY_DIR}/qpy-access-multiuser.py distribute
```

Distribute the cores among the users.

```
$ python ${QPY_DIR}/qpy-access-multiuser.py variables
```

Lists several internal variables of qpy-multiuser. Used mainly to check if something is going wrong.

```
$ python ${QPY_DIR}/qpy-access-multiuser.py status
```

Show the current status of the users, nodes and cores.

```
$ python ${QPY_DIR}/qpy-access-multiuser.py saveMessages
```

Saves messages from the internals of qpy-multiuser, that will be shown in the "variables" command. Mainly for debugging purposes.

ATTENTION!

The next commands are not supposed to be used in a normal run. Use them only if you know exactly what you are doing.

```
$ python ${QPY_DIR}/qpy-access-multiuser.py __user <user_name> <address> <port> <conn_key>
```

Adds a new user to qpy-multiuser.

```
$ python ${QPY_DIR}/qpy-access-multiuser.py __req_core <user_name> <jobID> <n_cores> <mem> <que
```

Requires a core from qpy-multiuser.

```
$ python ${QPY_DIR}/qpy-access-multiuser.py __remove_job <user_name> <job_ID> <queue_size>
```

Removes a job from qpy-multiuser.

2.6 Files

This is a list of the files in the qpy-multiuser directory.

`~/.qpy-multiuser/distribution_rules`

This file defines how the cores are distributed to the users. The basic syntax is one of the following:

```
1 even minimum <n_cores>
```

This means an even distribution among the users, with at least `n_cores` granted for each.

`~/.qpy-multiuser/allowed_users`

A list with all the users that can use the current qpy environment:

```
1 <user_1>
2 <user_2>
3 <user_3>
```

`~/.qpy-multiuser/nodes`

A list with all the nodes available in the qpy environment. Each line has: hostname of the node, followed by the number of cores this node has (or is available to QPY) and, optionally, a "M", to indicate that that node has preference for multicore jobs.

```
1 <node_1> <n_cores> [M]
2 <node_2> <n_cores> [M]
3 <node_3> <n_cores> [M]
```

`~/.qpy-multiuser/multiuser_connection_address`

The address where the qpy-multiuser instance will run. If you do not set this, QPY automatically uses the local machine.

```
1 <machine_address>
```

The following files are generated and used by QPY, but the administrator should not alter or delete them. They have the information required to make all the message transfers.

`~/.qpy-multiuser/multiuser_connection_port`

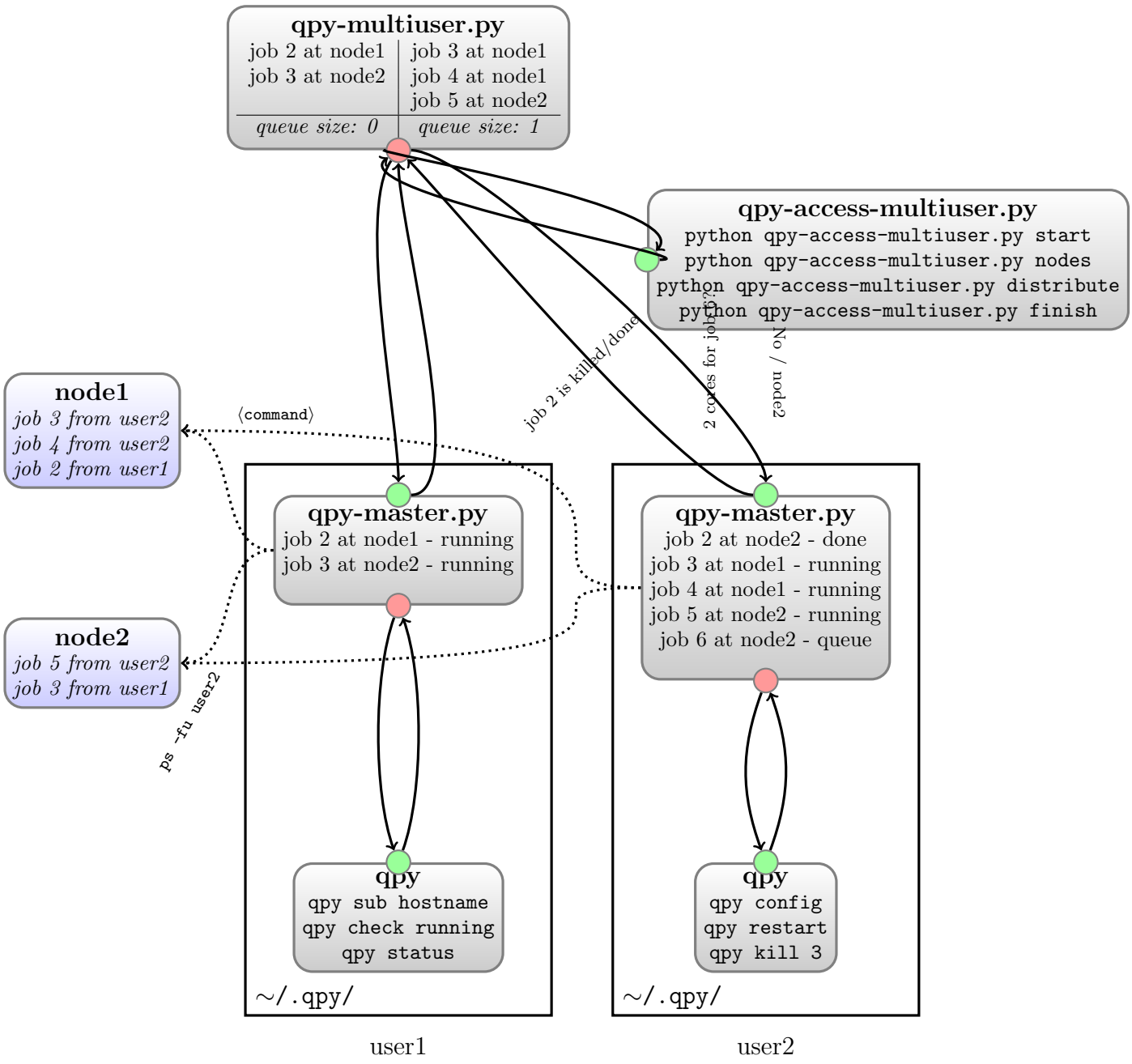
`~/.qpy-multiuser/multiuser_connection_key`

`~/.qpy-multiuser/connection_<user>_key`

`~/.qpy-multiuser/connection_<user>_key`

`~/.qpy-multiuser/connection_<user>_key`

3 For developers



Python's multiprocessing: Listener/Client connection