BgQmap Documentation

Release 0.3

BBGLab

Mar 26, 2018

Contents:

CHAPTER 1

General overview

1.1 BgQmap

BgQmap is a tool aimed to ease the usage of a cluster.

BgQmap contains 5 different tools:

- *run*: execute commands with extended resources
- *template*: create a jobs map file
- *submit*: submit jobs from a map file
- *reattach*: reattach to a previous bgqmap execution
- *info*: explore the metadata of your jobs

Currently, only SLURM is supported as workload manager.

1.1.1 Tools

bgqmap run Execute a command with more resources maintaining your working environment

```
bgqmap run -m <memory> -c <cores>  "<command>"
```

bgqmap template Create a *jobs map file* that works with **bgqmap submit**.

```
bgqmap template "<command with wildcards>" -f <jobs map file>
```

The file created uses the current loaded Easy Build modules and the current conda environment as jobs pre-commands[1] if not explicitly provided.

The job commands are all the combinations that result of the expansion of:

{{list,of,items}} comma separated list of items

[1] Commands executed before any actual job

{{file}} all lines in file

***, ?, [x-y]** wildcards in Python's glob module

Wildcards of the format `{{...}}` are expanded in a first phase and glob wildcards are expanded later on.

As additional feature, any of the above mentioned groups can be named `{{?<name>:...}}` and replaced anywhere using `{{?=<name>}}`.

Note: To name glob wildcards they should be solely in the group. E.g. `{{?myfiles:*}}`

---

bgqmap submit Execute all jobs from a *jobs map file*

```
bgqmap submit -m <memory> -c <cores>  <jobs map file> --logs <logs folder> --max-
↪running <#>
```

`bgqmap submit` has been implemented to submit a set of jobs to a cluster for execution and control them. It acts as a layer between the workload manager and the user preventing she/he from submitting a huge number of jobs at once (potentially blocking future users). The number of jobs that can be submitted to the workload manager is controlled by the *–max-running* flag.

Warning: If `bgqmap submit` is closed, jobs that have not been submitted to the workload manager will never be. Thus, it is recommended to run it inside a **screen**.

In addition, in the folder indicated to store the logs with the *–logs* flag the user can find important information about each job execution as well as the logs from STDOUT and STDERR.

Another feature of this tool is the possibility to group your jobs with the *–grouping* option. This option uses the value passed as the number of commands that fit in each job. Thus, several commands can be executed as part of the same job, one after another. This option can be interesting for "small" jobs as they use the same allocation. If any of the commands fail, the associated job will fail.

Finally, any job command can include two values that are substituted before execution:

${{JOB}} identifier of the job

${{LINE}} identifier of the line the job command has in the input file

---

Note: `${{JOB}}` is the same for all job commands within a group

---

bgqmap reattach Once a `bgqmap submit` execution is closed, you can reconnect from its logs directory

```
bgqmap reattach --logs <logs folder>
```

Note: If in the previous execution there were jobs that have not been submitted to the workload manager `bgqmap reattach` will start to submit them.

---

bgqmap info `bgqmap submit` generates a file for each job with metadata information. `bgqmap info` is designed to explore them and retrieve the requested data. Information is stored in json format and the user can request any fields:

```
bgqmap info --logs <logs folder> <field 1> <field 2>.<subfield 1> ...
```

In addition, the *–status* option can be used to filter the jobs by their status (completed|failed|other|pending|running|unsubmitted|all).

If you do not pass any field, then the return value is the input commands of the jobs.

1.1.2 Jobs map file

This file contains a list of the commands to be executed as well as commands to be executed before and after each job (e.g. loading Easy Build modules or conda environments). The format of the file is:

```
# command to be executed before any job
## parameters for all the jobs (e.g. cores=7, memory=16G)

job command
job command

# command to be executed after any job
```

1.1.3 License

LICENSE.

CHAPTER 2

run

The `bgqmap run` command is aimed to be use to execute a single command in a cluster with extended resources.

In certain cluster managers, you can ask for job resources to have a interactive console running on a worker node. Typically, the resources of such a job are quite limited, so few resources are taken even if people leave that console open.

`bgqmap run` allows users to run one specific command as another job and then return so that resources are optimized and only taken for the time that the job requires them.

Note: `bgqmap run` keeps your working directory and environment variables for the execution.

Once the job finishes, `bgqmap run` will try to provide the user with some job statistics (if available) like the memory consumed or the elapsed time.

2.1 Usage

```
bgqmap run -m <memory> -c <cores> "<command>"
```

Usage: bgqmap run [OPTIONS] CMD

Execute CMD in shell with new resources

Options:

-c, --cores INTEGER Number of cores to use. Default: 4

-m, --memory TEXT Max memory. Default: 16G. Units: K|M|G|T. Default units: G

-h, --help Show this message and exit.

2.2 Examples

Usage example:

```
$ bgqmap run -c 6 -m 12G "sleep 5 && echo 'hello world'"
Executing sleep 5 && echo 'hello world'
salloc: Granted job allocation 31707
hello world
salloc: Relinquishing job allocation 31707
Elapsed time:  00:00:05
Memory  0G
```

Jobs that require more resources can be easily re-run:

```
$ python test/python_scripts/memory.py 10
1 Gb
2 Gb
...
8 Gb
Killed

$ bgqmap run -m 12 "python test/python_scripts/memory.py 10"
Executing python test/python_scripts/memory.py 10
salloc: Granted job allocation 36015
1 Gb
...
10 Gb
salloc: Relinquishing job allocation 36015
Elapsed time:  00:00:36
Memory  10G
```

CHAPTER 3

template

`bgqmap template` is a tool aimed to ease the creation of a *jobs file* to be used with **bgqmap submit**.

Features:

- find your current loaded *EasyBuild modules* and adds them to the output as pre-command
- find your current *conda environment* and add them to the output as pre-command
- the *job parameters* passed through command line are added to the generated jobs_file

3.1 Generating a file

By default, the output is printed to the standard output. You can provide a file with the `-f` flag or redirect the output to a file (`> file.txt`).

3.2 Using wildcards

`bgqmap template` accepts two types of wildcards:

- **user wildcards**: indicated with `{{...}}` They can contain:
  - list of `,` separated items: the wildcard is replaced by each item
  - file name: he wildcard is replaced by each line in the file
- **glob wildcards**: if any of `*`, `**`, `?` and `[x-y]` is found, it is assumed to be a *glob* wildcard and therefore expanded using the python glob module.

3.2.1 How it works

The expansion of wildcards is a two step process. First *user wildcards* are expanded and *glob wildcards* are expanded in a second phase. For the latter, any set of characters surrounded by blanks is analysed. If it contains one or more of the mentioned wildcards, a glob search is performed.

Note: Use \ before glob wildcards to avoid their expansion

3.2.2 Named groups

`bgqmap template` contains a special feature that allows the user to replace the values of **any of the wildcards** in different parts of the command.

To use this feature, the wildcard needs to be named using `{{?<name>:<value>}}` and it can be replaced anywhere using `{{?=<name>}}`.

The *name* can be anything, but a *glob wildcard* character. It cannot start with a number.

Tip: We recommend to limit the names to characters in a-z, A-Z and 0-9.

The *value* can be **anything** in a *user wildcard* or a *glob wildcard*.

Note: Even if it is possible to use a glob wildcard in a user wildcard (e.g. {{a,*.txt}}) we do not recommend this use for named groups as the result might differ from the expected.

Warning: As mentioned, in a user group you can place anything that is is a user or glob wildcard. Thus, `{{?group:*}}.txt` recognize the glob wildcard and and will do a glob search for all `.txt` files. On the other hand, `{{?group:*.txt}}` assumes it is a user wildcard (as it is not only a glob wildcard) and will try to open a file named `*.txt` which most likely will not exits and will fail.

3.3 Usage

```
bgqmap template "<command>" -m <memory> -c <cores> -f <output file>
```

Usage: bgqmap template [OPTIONS] CMD

> Create a file template for execution with bgqmap.
>
> Conda environment and Easy build modules, if not provided are taken from the corresponding environment variables
>
> The commands accepts '{{...}}' and '*', '?', [x-y] (from glob module) as wildcards. If you want to use the value resulting of the expansion of that wildcard, name it '{{?name:...}}' and use it anywhere '{{?=name}}' and as many times as you want.
>
> First, items between '{{...}}' are expanded: If there only one element, it is assumed to be a file path, and the wildcard is replaced by any line in that file which is not empty or commented. If there are more ','

separated elements, it is assumed to be a list, and the wildcard replaced by each of the list members. If the inner value corresponds to one of the glob module wildcards, its expansion is postponed.

In a second phase, '*' wildcards are substituted as in glob.glob. Wildcards with are not in a named group are expanded first, and the latter ones are expanded in a final iterative process.

Options:

-f, --file PATH File to write the template to

-c, --cores INTEGER Number of cores to use

-m, --memory TEXT Max memory

-t, --wall_time TEXT Wall time

--conda-env TEXT Conda environment. Default: current environment

--module PATH Easy build modules. Default: current loaded modules

-h, --help Show this message and exit.

3.4 Examples

Easybuild modules and **conda environments** are recognized:

```
$ bgqmap template "sleep 5"
# module load anaconda3/4.4.0
# source activate test_bgqmap
sleep 5
```

Job parameters can also be added:

```
$ bgqmap template "sleep 5" -c 1 -m 1G
##cores=1, memory=1G
sleep 5
```

Using **user wildcards** with lists:

```
$ bgqmap template "sleep {{5,10}}"
sleep 5
sleep 10
```

Using **user wildcards** with files:

```
$ bgqmap template "sleep {{examples/input/sleep_times.txt}}"
sleep 5
sleep 10
```

Using **glob wildcards**:

```
$ bgqmap template "mypgrog --input examples/input/*.txt"
mypgrog --input examples/input/sleep_times.txt
mypgrog --input examples/input/empty.txt
```

Using **named wildcards**:

```
$ bgqmap template "myprog --input examples/input/{{?f_name:*}}.txt --variable {{?v_
↪name:a,b}} --output {{?=f_name}}_{{?=v_name}}"
myprog --input examples/input/sleep_times.txt --variable a --output sleep_times_a
myprog --input examples/input/empty.txt --variable a --output empty_a
myprog --input examples/input/sleep_times.txt --variable b --output sleep_times_b
myprog --input examples/input/empty.txt --variable b --output empty_b
```

CHAPTER 4

submit

`bgqmap submit` launches a bunch of commands to the workload manager for each execution. The commands to be executed come from a file with the following format:

```
# pre-command 1
# pre-command 2
...
# pre-command l
## job parameters

job 1
job 2  ## job specific parameters
...
job m

# post-command 1
# post-command 2
...
# post-command n
```

Job pre-commands Command to be executed before any job

Job parameters Resources asked to the workload manager (e.g. memory or cores)

Job command Bash command to be executed. One command corresponds to one job unless *groups* are made

Job post-commands Commands to be executed before any job

An example of such file:

```
# module load anaconda3
# source activate oncodrivefml
## cores=6, memory=25G

oncodrivefml -i acc.txt -e cds.txt -o acc.out
oncodrivefml -i blca.txt -e cds.txt -o blca.out
```

`bgqmap submit` is a tool that is not only intended to easy the job submission, but also tries to limit the jobs that one user submits at once, preventing that he/she takes the whole cluster.

4.1 General features

It uses the current directory as the **working directory** for the jobs.

The **command line parameters** override *general job parameters* but not *specific job parameters* (*see below*).

There is a limit of 1000 commands per submission without **grouping**. Grouping involves that a set of `x` commands is executed one after the other as part of the same job. If one fails, the job is terminated.

> **Warning:** *Job specific parameters* are ignored in grouped submissions.

If a job is killed due to high memory usage, it is **resubmitted** automatically (as long as bgqmap is active) requesting twice the memory that was requested in the previous execution. This self-resubmission feature will only be done twice.

4.2 How does it work?

4.2.1 Reading the jobs file

The lines at the head of the file with a single `#` are interpreted as *job pre-commands*.

Any non-empty line that does not start with `#` is considered as a *job command*. If the job command contains `##` anything from there is interpreted as *specific job parameters*.

Any line starting with `#` after the first *job command* is interpreted as a *job post-command*.

Any line starting with `##` is assumed to contain the *general job parameters*. That is, the parameters (memory, cores. . .) for all the jobs.

> **Warning:** To increase readability, we highly recommend to place all *post-commands* at the end of the file and the *general parameters* right after the *pre-commands*.

4.2.2 Generating the jobs

Once the *jobs file* is parsed, the jobs are created. This process involves:

- creating an **output directory** and copying the *jobs file*

  > **Note:** If the output directory is not empty, `bgqmap` will fail

- each job receives and **id** that correspond to its line in the submit folder
- for each job command one file with the job **metadata** is created (named as <*job id*>.*info*)

  > **Warning:** To prevent lots of writing to the `.info` file, `bgqmap` only writes to disk on special cases, when explicitly asked or before exiting.

- for each job and a **bash script file** with the commands to be executed is created. The file is named as *<job id>.sh* and consists on:
  - all the *pre-commands*
  - all the *commands* in the group or a single command if not groups are made
  - all the *post-commands*

Note: The job commands can contain two wildcards that are expanded before job submission:

- ${JOBID}: identifier of the job (is the same for all the commands in a group)
- ${LINE}: identifier of the line of the job (unique for each command)

4.2.3 Running the jobs

- the jobs start to be submitted to the workload manager. Only certain amount of jobs are submitted according to the `-max-running` parameter. This parameter accounts for running and pending jobs.
- each job requests certain **resources** to the workload manager. The order of priority is: *command line parameters*, *general job parameters* from the *jobs file* and *default parameters*.

Note: If no grouping is perform and the job contains **specific job parameters** those have the highest priority.

- the job output to the standard error is logged in a file named as *<job id>.out* and the job output to the standard error is logged in *<job id>.err*.

4.3 Usage

```
bgqmap submit -m <memory> -c <cores> <jobs file>
```

Usage: bgqmap submit [OPTIONS] JOBS_FILE

Submit a set of jobs

The following values will be extended ${JOB}: for id ${LINE}: for line number in the input file

Options:

-l, --logs PATH Output folder for the bgqmap log files. Default a folder is created in the current directory.

-r, --max-running INTEGER Maximum number of job running/waiting. Default: 4.

-g, --group INTEGER Group several commands into one job. Default: no grouping.

--no-console Show terminal simple console

-c, --cores INTEGER Number of cores to use. Default: 2

-m, --memory TEXT Max memory. Default: 4G. Units: K|M|G|T. Default units: G

-t, --wall_time TEXT Wall time for the job. Default: no wall time.

-w, --working_directory TEXT Working directory. Default: current.

-h, --help Show this message and exit.

4.4 Examples

Using this jobs file:

```
sleep 5 && echo 'hello world after 5'
sleep 10 && echo 'hello world after 10'
```

Basic example:

```
$ bgqmap submit -m 1 -c 1 examples/input/hello.map --no-console
Finished vs. total: [0/2]
Job 0 done. [1/2]
Job 1 done. [2/2]
Execution finished
```

In the **output directory** of bgqmap, you can find a copy of the input file (as `bgqmap_input`) and for each job 4 up to for different files as explained above:

```
$ ls bgqmap_output_20170905
0.err  0.info  0.out  0.sh  1.err  1.info  1.out  1.sh  bgqmap_input
```

The **output directory** must not exist before the submission:

```
$ bgqmap submit examples/input/hello.map --no-console
BgQmapError: Output folder [bgqmap_output_20170905] is not empty. Please give a␣
↪different folder to write the output files.
```

Grouping reduces the number of jobs, but expecific job execution parameters are ignored:

```
$ bgqmap submit examples/input/hello.map -g 2 --no-console
Specific job execution parameters ignored
Finished vs. total: [0/1]
Job 0 done. [1/1]
Execution finished
```

The following examples make use of this other *jobs file*:

```
# module load anaconda3/4.4.0
## memory=8G
python memory.py 8
python memory.py 10 ## memory=10G
```

The **working directory** is helpful when your jobs file does not contain the full path to your script

```
$ bgqmap submit examples/input/memory.map --no-console
Finished vs. total: [0/2]
Job 2 failed. [1/2]
Job 3 failed. [2/2]
Execution finished

$ bgqmap submit examples/input/memory.map -w test/python_scripts/ --no-console
Finished vs. total: [0/2]
Job 2 done. [1/2]
```

(continues on next page)

(continued from previous page)

```
Job 3 done. [2/2]
Execution finished
```

CHAPTER 5

reattach

In order to re-open a previous **bgqmap** submission, `bgqmap reattach` does so using the data provided by the output (logs files) of that **bgqmap submit** execution.

There are two main differences between the execution you run with **bgqmap submit** and the reattached one.

1. Once you reattach, the *maximum number of jobs* that can be running in the cluster is reset to its default value. This is done to prevent a huge number of jobs to be launched all at once.
2. If you have stopped the first execution, when reattaching, it will be *running* again and unsubmitted jobs will be launched.

5.1 Usage

```
bgqmap reattach [--logs <folder>]
```

Usage: bgqmap [OPTIONS] COMMAND [ARGS]...

Error: No such command "reattch". (/home/ireyes/anaconda3/envs/bgframework) ireyes@lopez003507:/home/ireyes/projects/framework/bgqmap2$ bgqmap reattach -h Usage: bgqmap reattach [OPTIONS]

Reattach to a previous execution in FOLDER

Default FOLDER is current directory

Options:

-l, --logs PATH	Output folder of the bgqmap log files. Default is current directory.
--force	Force reattachment
--no-console	Show terminal simple console
-h, --help	Show this message and exit.

5.2 Examples

```
$ bgqmap reattach -l examples/output/hello --no-console
Finished vs. total: [2/2]
Execution finished
```

CHAPTER 6

info

`bgqmap info` is a tool aimed to explore the metadata of your jobs or to get the commands of certain jobs.

6.1 Exploring the metadata

`bgqmap info` can explore the metadata of your jobs and retrieve the fields of interest.

The first column corresponds to the **job id** and each of the other columns correspond to the requested fields. *Missing fields* return and empty string.

To check what fields you can return, you can simply take a look at one of the `.info` files. To access nested elements use `.` to divide the levels (e.g. `usage.time.elapsed`).

The return data is *tab separated* or `|` *separated* if the *collapse flag* is provided.

6.1.1 Usage

```
bgqmap info -s <status> -l <bgqmap logs folder> <field 1> <field 2> ... <field n>
```

6.2 Filtering commands

`bgqmap info` can also return a subset of your commands file that corresponds to the ones whose job have a certain status.

This option is enabled when no fields are passed in the command line. Moreover, in this case, the *collapse flag* removes empty lines from the output.

6.2.1 Usage

```
bgqmap info -s <status> -l <bgqmap logs folder>
```

6.3 Usage

Usage: bgqmap info [OPTIONS] FIELDS

Search for FIELDS in the metadata files in FOLDER

FIELDS can be any key in the metadata dictionary. (nested keys can be accessed using '.': e.g. usage.time.elapsed). Missing fields will return an empty string. The return information is tab separated or 'l' separated if the collapse flag if passed.

If no FIELDS are passed, then the output corresponds to the input command lines that match that resulted in jobs with that status criteria. In this case, the collapse flag forces removes blank lines from the output.

Options:

-f, --file PATH File to write the output to

-s, –status [completed|failed|other|pending|running|unsubmitted|all|c|f|o|p|r|u|a] Job status of interest

--collapse Collapse the output

-l, --logs PATH Output folder of the bgqmap log files. Default is current directory.

-h, --help Show this message and exit.

6.4 Examples

Get the fields of interest from your jobs:

```
$ bgqmap info -s completed -l examples/output/hello usage.time.elapsed retries
id      usage.time.elapsed      retries
1       00:00:12        0
0       00:00:07        0
```

CHAPTER 7

Indices and tables

- genindex
- modindex
- search