Running and managing jobs

Intro

For deploying and managing jobs on Isabella computer cluster, SGE or Son of Grid Engine is used and job managment system JMS. In this document use of SGE ver. 8 is described.

Running jobs

User applications (in continuation **jobs**) which are run with SGE system have to be described with the **startup shell script**. Withing starting script, alongside the usual commands, **SGE parameters** are stated. It is possible to state the same parameters outside of starting script, during job submission.

Job run starts with **qsub** command:

```
qsub <SGE_parameters> <name_of_starting_script>
```

> SGE also has **graphical interface** or **GUI** for access to whole system functionality. GUI starts with **qmon** command. Use of GUI will not be described because there is no instruction manual within it (*Help* button).

Describing jobs

The SGE system language is used to describe jobs, and the **job description file (startup script)** is a standard ***shell*** script. The header of each script lists the **SGE parameters** that describe the job in detail, followed by the normal commands to execute the user application.

Startup script structure:

my_job.sge

```
#!/bin/bash

#$ -<parameter1> <value1>
#$ -<parameter2> <value2>

<command1>
<command2>
```

The job described by this start script is submitted with the command:

```
qsub my_job.sge
```

The qsub command returns a job ID that is used to monitor the job's status later:

```
Your job <JobID> ("my_job") has been submitted
```

Basic SGE parameters

```
-N <job_name> : the job name that will be displayed when retrieving job information
-P <project_name> : the name of the project to which the job belongs
-cwd : defines that the directory where the startup script is located is the working directory of the job
```

> The default working directory of the job is $HOME.

```
-o <file_name> : the name of the file where the standard output is saved
-e <file_name> : the name of the file where the standard error is saved
-j y|n : allows merging standard output and standard error into the same file (default value is n)
```

> If standard output and error are not explicitly specified, they will be saved to the files:
>
> 1. If job name is not defined:
>   `<working_directory>/<script_name>.o<job_id>`
>   `<working_directory>/<script_name>.e<job_id>`
> 2. else:
>   `<working_directory>/<script_name>.o<job_id>`
>   `<working_directory>/<script_name>.e<job_id>`
>
> The -o and -e parameters can point to a directory:
>
> **`#$ -o outputDir/`**
> **`#$ -e outputDir/`**
>
> In this case, SGE will create standard output and standard error files in the outputDir directory named `<job_name>.o<JobID>` and `<job_name>.e<JobID>`
>
> **Important:** **`outputDir`** must be created manually before submitting the job.

```
-M <emailAddress>[,<emailAddress>]…        : list of email addresses to which job notifications are sent

-m [a][b][e][n]        : defines in which case mail notifications are sent:
                                        b - start of job execution,
                                        a - job execution error,
                                        e - completion of job,
                                        n - do not send notifications (default option)

-now y|n : the value of y defines that the job must be performed immediately. For interactive jobs, this is the
default value.
                   If SGE cannot find free resources, the job is not queued but ends immediately.

-r y|n : whether the job should be restarted in case of a runtime error (default value is n)

-R y|n : the value of y defines that SGE will reserve nodes when deploying (important for multiprocessor jobs)

-l <resource>=<value>[,<resource>=<value>...] : defines the resources that the job requires. See Resources for
details.

-pe <parallel_environment> <range> : parameter is used for parallel jobs.
     The first parameter defines the module that runs the requested form of parallel job.
     The second parameter defines a specific number of processors or a range in the form <N>,[<Ni>,...]<S>-<E>,
[<Si>-<Ei>,] whichp arallel job demands. For more details see Parralel jobs

-q <queue_name>[,<queue_name>...] : job queue in which job is being prepared. This option can also be used to
request a specific node, such as requesting a local job queue (eg a12.q@sl250s-gen8-08-01.isabella).

-t <start>:<end>:<step> : the parameter defines that it is a job array. For details, see Job arrays.

-v <variable>[=<value>][,<variable>[=<value>]...] : ption defines that SGE sets the environment variable when
executing the job. This parameter is useful when the application uses special environment variables, because
SGE does not set them by default when starting the job.

-V : SGE passes all current environment variables to the job.
```

Note: spaces are not allowed when listing parameter values (eg -l or -q).

Detaljan popis i informacije o parametrima moguće je dobiti naredbom **`man qsub`**.

SGE environment variables

Within the startup script it is possible to use SGE variables. Some of them are:

```
$TMPDIR : the name of the directory where temporary files can be saved (/scratch)
$JOB_ID : SGE job identifier
$SGE_TASK_ID : task identifier of the job arrays
$SGE_O_HOST : address of the computer from which the job was started
$SGE_O_PATH : the original value of the PATH environment variable when starting the job
$SGE_O_WORKDIR : the directory from which the job was started
$SGE_STDOUT_PATH : file where standard output is saved
$SGE_STDERR_PATH : file where standard error is saved
$HOSTNAME : the address of the computer on which the script is executed
$JOB_NAME : job name
$PE_HOSTFILE : the name of the file in which the addresses of the computers are listed
$QUEUE : the name of the queue in which the job is executed
```

Types of jobs

Serial jobs

The simplest form of SGE jobs are batch jobs that require only one processor to run. For them, it is usually not necessary to specify any special parameters, but only the name of the program is specified.

Examples of use:

1. An example script without additional parameters:

```
#!/bin/bash

date
```

2. Example of a simple script with parameters:

```
#!/bin/bash

#$ -N Date_SGE_script
#$ -o Date_SGE.out
#$ -e Date_SGE.err

date
```

3. Example of running a program from the current directory:

moj_program.sge

```
#!/bin/bash

#$ -N myprog
#$ -P local
#$ -o myprog.out
#$ -e myprog.err
#$ -cwd

myprog
```

Parallel jobs

To start parallel jobs, it is necessary to specify the desired parallel environment and the number of processor cores required to perform the job.

The syntax is:

```
#$ -pe <type_of_parallel_job> <N>,[<Ni>,...]<S>-<E>,[<Si>-<Ei>,]
```

Examples of use:

1. The job requires 14 processor cores to run:

```
#$ -pe *mpi 14
```

2. The number of allocated processor cores can be between 2 and 4:

```
#$ -pe *mpi 2-4
```

3. The number of allocated processor cores can be 5 or 10:

```
#$ -pe *mpi 5,10
```

4. The number of allocated processor cores can be 1 or between 2 and 4:

```
#$ -pe *mpi 1,2-4
```

More information about the parallel environments available on Isabella can be found on this page : Job queues and parallel environments.

Since the user does not need to know in advance how many processor cores will be allocated, SGE sets the value of the $NSLOTS variable to the number of allocated cores.

Running parallel jobs itself is specific because there are tools for running sub-jobs (eg. mpirun) that do the scheduling of sub-jobs on nodes themselves. When SGE assigns nodes to a parallel job, it saves the **list of nodes** in the $TMPDIR/machines file which is passed as a parameter to the parallel jobs (eg mpirun, mpiexec, pvm...) inside the job description script.

An **example** of a start script for starting one type of parallel job:

```
#!/bin/bash

#$ -N parallel-job
#$ -cwd
#$ -pe *mpi 14

mpirun_rsh -np $NSLOTS -hostfile $TMPDIR/machines ...
```

Job arrays

SGE enables multiple starting of the same job, the so-called job arrays. Sub-jobs within an array are called tasks and each task gets its own identifier. When running jobs, the user specifies a range of job identifier values using the **-t** parameter:

```
#$ -t <start>:<end>:<step>
```

The value **<start>** is the identifier of the first task, **<end>** the identifier of the last task, and **<step>** is the value by which each subsequent identifier between <start> and <end> is incremented. SGE stores the identifier of each task in the variable **$SGE_TASK_ID**, with which users can assign different parameters to a particular task. Tasks can be serial or parallel jobs.

Examples of use

1. Example script to run a job array of 10 batch jobs:

```
#!/bin/bash

#$ -N job_array_serial
#$ -cwd
#$ -o output/
#$ -j y
#$ -t 1-10

./starSeeker starCluster.$SGE_TASK_ID
```

2. Example script to run a job array of 10 parallel jobs:

```
#!/bin/bash

#$ -N job_array_parallel
#$ -cwd
#$ -o output/
#$ -j y
#$ -t 1-10

mpiexec -machinefile $TMPDIR/machines ./starseeker starCluster.$SGE_TASK_ID
```

Interactive jobs

SGE enables the launch of interactive jobs. The **qrsh** command is used to run jobs interactively.

It is recommended to use this form of jobs in the case when it is necessary to **compile or debug applications on nodes**.

Unlike using ssh, this lets SGE know that the nodes are busy and not run other jobs on them. When executing the command interactively, it is necessary to specify the full path to the command. If the SGE currently has no free resources and the job is to be left waiting in the queue, it is necessary to specify the "-now n" parameter. Otherwise, SGE will immediately end the job execution with the message:

```
Your "qrsh" request could not be scheduled, try again later.
```

Examples of use:

1. Direct access to the command line of the test node:

   ```
   qrsh
   ```

2. Interactive command execution:

   ```
   qrsh /home/user/moja_skripta
   ```

3. Interactive application execution with graphical interface:

   ```
   qrsh -DISPLAY=10.1.1.1:0.0 <moja_skripta>
   ```

Advanced job descriptions

Saving temporary results

It is not recommended to use the $HOME directory to save temporary results generated during job execution. This reduces the efficiency of the application and burdens the front end and the cluster network.

SGE creates a directory on the disk on the work nodes (/scratch) for each individual job, of the form /scratch/<jobID>.<taskID>.<queue>. The address of this directory is saved by SGE in the variable $TMPDIR.

For higher execution speed, using a temporary directory on the /scratch disk is also recommended for jobs that often require random access to data on disk, such as TensorFlow and PyTorch jobs.

If there are indications that the created temporary files will **exceed 500 GB**, the temporary data should be saved to the **/shared disk** (described below).

The temporary directory on the /scratch disk is deleted automatically at the end of the job execution.

Examples of use:

1. An example of simple use of the $TMPDIR variable:

```
#!/bin/bash

#$ -N scratch_1
#$ -cwd
#$ -o output/scratch.out
#$ -j y
#$ -l scratch=50

cd $TMPDIR
pwd > test
cp test $SGE_O_WORKDIR
```

2. An example of copying data to a scratch disk:

```
#!/bin/bash

#$ -N scratch_2
#$ -cwd
#$ -o output/scratch.out
#$ -j y
#$ -l scratch=50

mkdir -p $TMPDIR/data
cp -r $HOME/data/* $TMPDIR/data

python3.5 main.py $TMPDIR/data
```

If the **temporary data exceeds 500 GB**, it is necessary to use **/shared**. Unlike scratch, the directory on shared must be created manually and there is no automatic directory removal.

Example of use:

```
#!/bin/bash

#$ -N shared
#$ -cwd
#$ -o output/shared.out
#$ -j y

mkdir -p /shared/$USER/$TMPDIR
cd /shared/$USER/$TMPDIR

pwd > test
cp test $SGE_O_WORKDIR
```

Resources

When starting jobs, the user can describe in more detail which conditions must be met for the job. For example, it is possible to require only a certain architecture of the worker node, the amount of memory or the execution time. Specifying required resources allows for better job scheduling and gives jobs a higher priority (more on the Priorities page on Isabella).

The required resources are specified using the **-l** parameter:

```
#$ -l <resource>=<value>
```

Resources which **define job requirements**:

```
arch : node architecture (eg. lx26-x86, lx25-amd64)
hostname : node address (eg. c4140-01.isabella)
```

Resources that **place real limits on jobs**:

```
vmem : amount of virtual memory (format: <num>K|M|G)
rss : amount of real memory
stack : stack size
data : total amount of memory (without stack)
fsize : total file size
cput : processor time (format: [<hours>:<min>:]<sec>)
rt : real time
scratch : space on the scratch disk expressed in GB
```


The values of these resources should be carefully defined (eg take 50% higher values than expected). In case of exceeding, the job will be stopped with the "segmentation fault" signal.

Values cannot be changed for active jobs.

The **resource values** defined in the job start script are set per process. For example, if a user on one node requires 3 processor cores, the values of all requested resources will be multiplied by 3.

Example of use:

1. Example of a job that requires 20 CPU cores and 10 GB of RAM per process (the job will be allocated a total of 200 GB of RAM):

   ```
   #$ -pe *mpi 20
   #$ -l memory=10
   ```

2. The job requires 100 GB of scratch space:

   ```
   #$ -pe *mpisingle 4
   #$ -l scratch=25
   ```

Job queue

SGE supports multiple job queues, for different job types and maximum job duration, for different job types (interactive, vSMP jobs, ...) etc.

The desired job queue can be defined with the **-q** parameter:

```
#$ -q <queueName>
```

Popis dostupnih redova poslova na Isabelli može se pronaći na Job queues and parallel environments.

Job Status Notifications

SGE supports sending e-mail notifications about job status changes.

The **-M parameter** determines the email address to which job status notifications will be sent:

```
#$ -M <email_address>
```

The -m parameter determines in which cases the notification will be sent:

```
#$ -m [b][a][e][n]
```

- b - start of execution
- a - execution error
- e - completion of execution
- n - do not send notifications (default option)

Example of use:

```
#$ -m ae
#$ -M my@mail.com
```

Notification will be sent to the address **moj@mail.com** when the work is **interrupted** or **completed**.

Monitoring and management of job performance

Display of job status

SGE's **qstat** command is used to display job status. The command syntax is:

```
$ qstat <options>
```

By executing the **qstat command without additional options**, a printout of all current user jobs is obtained:

```
[tsmolcic@teran ~]$ qstat
job-ID  prior   name       user         state submit/start at     queue                          slots ja-task-
ID
------------------------------------------------------------------------------------------------------------------
-
 131982 0.00001 md5-10k.0  tsmolcic     r     05/28/2019 06:24:06 gpu.0.q@c4140-03.isabella          1
 131988 0.00001 md5-10k.2  tsmolcic     r     05/28/2019 09:29:26 gpu.2.q@c4140-01.isabella          1
 131990 0.00001 md1-5k.3   tsmolcic     r     05/28/2019 09:29:41 gpu.3.q@c4140-01.isabella          1
 131991 0.00001 md10-15.4  tsmolcic     r     05/28/2019 10:18:08 gpu.1.q@c4140-01.isabella          1
 131994 0.04119 md1kp.5    tsmolcic     qw    05/27/2019 13:14:14                                   20
 131996 0.00005 md1-5k.6   tsmolcic     qw    05/27/2019 13:16:30                                    1
```

Some of more important options:

```
-s [r|p|s|h] : filter jobs by state: r - active, s - stopped, h - stopped in queue, p - queued
-j [job_ID] : detailed display of job status (if job_ID is not specified, information on all active jobs is
displayed)
-f : detailed information about node load and job allocation to nodes is displayed
-F : detailed display of node data
-u <user_name> : only jobs from user <user_name> are displayed (* - for all users)
-q [queue_name] : command retrieves job queue information only
-g c : retrieve summary information about job queues: total queue load, total number of nodes in the queue,
number of busy ones
and free nodes and the number of nodes in special states
-g t : slave and master tasks are marked in the jobs view
-l <resource>=<value> : filter jobs by resources
-ext : additional information about jobs
-pri : detailed data on job priorities
```

The options of the qstat command can be combined, which is especially important when it is necessary to filter through several hundred jobs.

Examples of use:

1. View all jobs in the queue

```
$ qstat -u "*" -s p
```

2. Display of all user jobs that are currently running and required a16-mpi parallel environment

```
$ qstat -u "*" -s r -pe a16-mpi
```

3. Display of all jobs and nodes user pero

```
$ qstat –s r –f –u pero
```

4. Display of all jobs currently running on nodes with graphical processors:

```
$ qstat –u "*" -q gpu.*.q
```

Display of working nodes

To print the number of processors, cores and the amount of working memory per node with use of the following command:

```
$ qhost
```

The values of the available resources per work nodes can be printed with the following command:

```
$ qhost -F vendor,scratch,memory
```

Job management

The job can also be managed after launch.

While a job is in the queue, it is possible to temporarily stop its execution with the following command:

```
$ qhold <ID_posla>
```

A stopped job can be requeued with the following command:

```
$ qrls <ID_posla>
```

Greater control over the job is provided by the qmod command. The command makes it possible to temporarily stop active jobs by sending a SIGSTOP signal to the job. The job will enter the idle state (T) but will not make resources available (memory, file descriptors). The command also enables saving the state of the process to disk (checkpointing) for jobs that have this option. Furthermore, using the qmod command, the user can stop the active process and return it to the job queue.

The command parameters are:

```
-c : the command performs a process state saving
-f : parameter defines to execute the command whether it is possible or not
     (useful when requeueing jobs marked with the -r n parameter).
-r: the command stops the job and puts it back in the queue.
-s : the command stops the execution of the active process.
-us : command resumes execution of stopped active job.
```

The job is completely stopped or unqueued with the command:

```
$ qdel <job_ID>
```

It is possible to stop all user jobs:

```
$ qdel -u <username>
```

Force stop should be used for stuck jobs or jobs in loop:

```
$ qdel -f <job_ID>
```

Access to information about completed jobs

The **qacct** command is used to retrieve information about completed jobs. The syntax is:

```
$ qacct <ptions>
```

Most common example:

```
$ qacct -j <job_ID>
```

Prints all the information about the completed work.

Other useful **qacct** command options are:

```
-j <job_id> : detailed description of individual jobs
-h <node_address> : statistics for individual nodes
-q <queue_name> : statistics for individual queues
-o <username> : statistics for individual users
-pe <parallel_env_name> : statistics for individual parallel environments
-slots <number_of_processors> : statistics for jobs for a specified number of processors
-P <project> : display the consumption summary of the defined project
```

Examples of use:

1. Detailed information about all jobs performed on the cluster (caution: large amount of data):

   ```
   $ qacct -j
   ```

2. Display of information about all jobs of the defined user:

   ```
   $ qacct -j -o <user>
   ```

   Display a summary of the consumption of computer resources of a defined user (if <user> is not defined, data for all users is displayed):

   ```
   $ qacct -o <user>
   ```

3. Display information about all jobs for the defined project:

   ```
   $ qacct -j -P <projekt>
   ```

   Display of the consumption summary of the defined project (if <project> is not defined, data for all projects are displayed):

```
$ qacct -P <project>
```

Cheatsheets

Linux cheatsheet

Navigating the file system

Command	Command description
`pwd`	Shows the user's current location. The location is displayed as an absolute path to the current directory.
`cd`	Changing the current directory (cd - change directory).
`cd -`	Return to previous directory.

Directory management

Command	Command description
`mkdir dir1`	Creates directory named dir1.
`mkdir -p /tmp/novi/dir1`	The -p option automatically creates the necessary subdirectories.
`rm -rf dir1/*`	Deletes all files and subdirectories inside the directory dir1, ie. leaves the directory dir1 empty.
`rm -rf dir1/`	Deletes all files and subdirectories including dir1.

Copying files and directories

Command	Command description
`cp dat1 dat2`	Copies the file dat1 to dat2 (dat1 is unchanged).
`cp dat1 dir/`	Copies the file dat1 to the directory dir.
`cp -r dir1/* dir2/`	Copies all files from directory dir1 to directory dir2 without directory dir1 itself.
`cp -r dir1/ dir2/`	Copies all files and sub-directories in the dir directory to the directory dir2, including the dir1 directory.

Move and rename files and directories

Command	Command description
`mv dat1 dat2`	Renames the file dat1 to dat2.
`mv dat1 dir1`	If dir1 is a directory name, moves the file dat1 to the directory dir1.

Password change

Command	Command description
`passwd`	Changing the password of the current user. The command first asks to enter the old password, and then asks to enter the new password (twice). Note: when entering the password, for security reasons, no text is printed in the terminal.

Auto-fill and search of command history

Command	Command description
[Tab]	Auto-fill of orders. When the user starts typing a command, eg. passwd, they can type the first few letters (eg. pass) and press the [Tab] key. The shell will then automatically complete the command or print all commands that start with the string pass. File names on the disk can be supplemented in the same way.
[Ctrl] + [r]	Search command history. In the terminal, hold down the [Ctrl] key and press the [r] key. You start typing letters from a command, and previous commands that contain the letters you type appear. If you want to cycle through all the commands that contain the typed letters, press [Ctrl] + [r] again.

SGE cheatsheet

Job submit

Command	Command description
`qsub`	Submits a job and returns job ID.

Checking job status

Command	Command description
`qstat`	Shows the status of jobs on the cluster for the current user.
`qstat -f`	Shows the status of jobs and nodes.
`qstat -s rpsh`	Shows: p - jobs waiting in the queue; r - active jobs; s - temporarily stopped active jobs; h - temporarily stopped jobs in the queue.
`qstat -g c`	Displays a summary of the state of individual job queues.
`qstat -j <job_id>`	Displays a detailed view of information about one job
`qstat -u <user>`	Displays jobs of a specific user ("*"- for all)
`qstat -pe <name>`	Displays jobs that use a defined parallel environment.
`qstat -q <queue>`	Displays jobs in a defined job queue.

Stopping jobs

Command	Command description
`qdel <job_id>`	The job is completely stopped or moved from the queue.
`qdel -u <user>`	All jobs of the default user are stopped.
`qdel -f <job_id>`	Force stop for stuck jobs.

Information about completed jobs

Command	Command description

`qacct -j <job_id>`	Detailed information about the job with ID <job_id>.
`qacct -j`	Detailed information about all jobs (a large amount of data).
`qacct -j -o <user>`	Display of information about all jobs of a defined user.
`qacct -o <user>`	Display of a summary of consumption of a defined user; if <user> is not defined, data for all users is displayed.
`qacct -slots [<count>]`	Summary display of all jobs that used the given number of processors; if <count> is not defined, data for all values is displayed.
`qacct -j -P <project>`	Display of information about all jobs for the defined project.
`qacct -P <project>`	Display of the consumption summary of the defined project; if <project> is not defined, data for all projects is displayed.
`qacct -j -q <queue>`	Display of information about all jobs for the defined job order.
`qacct -q <queue>`	Display of the consumption summary of the defined job order.