Hessisches Kompetenzzentrum für Hochleistungsrechnen

Scheduler Exercise

TECHNISCHE UNIVERSITÄT DARMSTADT

Logging into your HPC cluster

1. If you are using a Windows machine, make sure that you have an ssh client (preferably MobaXTerm) installed. Depending on the local cluster access policies, you may have to use your university's VPN to get access.
2. The exercises should have been provided beforehand as a zip archive. Copy them to your cluster home directory from your local machine using `scp` or `rsync` on Linux/Mac. On Windows, you can use WinSCP, FileZilla, or MobaXTerm's builtin file browser.
3. Unzip the file with `unzip exercises.zip` and verify if the files have been decompressed to the 'exercises' folder.

Adapting the scripts to your cluster location

1. Every cluster has its own policies, parameter defaults, and assumptions. The SGE versions are configured to run on Marburg's MaRC2 cluster, while the SLURM versions have been tested on Darmstadt's Lichtenberg cluster. If you are working on a different cluster, other or additional parameter restrictions may apply. For the Hessian clusters, these should be listed on the corresponding Cluster Reference Sheet, available at `https://www.hkhlr.de/en/material`.
2. If you are unsure, try to submit the job `simple.slurm` (SLURM) or `simple.sge` (SGE) and note any error messages. The exact steps to submit these jobs are described in Exercise 1. SLURM's error messages are usually quite straightforward if there are mandatory parameters missing.
3. If you have to make any additions to simple.slurm (or `simple.sge`, respectively) to make it run correctly, remember to also apply these changes to the other example scripts.

1 Hands-on Exercises

1.1 A Simple Job Script

You can do a lot of complex work inside job scripts, but to start out, we will use a very bare-bones template. If you start to work on a new system, it is often preferable to keep the first tests very simple, which eases debugging them and lets you use them as templates for more complex tasks later on.

1. Take a look at the script to find out what it is supposed to do. Use `nano`, `less`, `vim`, or any other text editor / textviewer you prefer.:

   SLURM
   ```
   nano simple.slurm
   ```
   SGE
   ```
   nano simple.sge
   ```

2. Check if there are any parameters whose purpose is not clear to you. If you do not find them on the Cluster Reference Sheet, try the manual page of your clusters job submit command, i.e.:

   SLURM
   ```
   man sbatch
   ```
   SGE
   ```
   man qsub
   ```
   Hint: On manpages, you can forward-search for a term by typing `/<searchterm>`. There are more options for searching, which you can find on the manpage for `less`, the default display application for manpages.

3. If everything looks okay to you, submit the job:

   SLURM
   ```
   sbatch simple.slurm
   ```
   SGE
   ```
   qsub simple.sge
   ```
   And track its state via `squeue` (SLURM) or `qstat` (SGE).

4. Once the job is finished, check its generated output file(s) to verify it ran without error.

1.2 Task Arrays

Task arrays are an effective and simple way to start a swarm of similar jobs through a single job script. Although it is not applicable in all situations, any workload that can be transformed into a task array can make a project much more efficient.

1. As in the previous exercise, take a look at the example script first before you submit them, and make sure to note what each line and parameter is supposed to do. The script is named `array.slurm` (SLURM) and `array.sge` (SGE).

2. Note that some scheduler directives are commented out by adding an extra `#`. This is to show different ways of defining task array indices and additional directions like how many tasks should be allowed to run concurrently. To test these different settings, try commenting out (adding a second '#') and commenting in (removing the second '#') different array directives. Check how many overall tasks are running, how many are concurrently running, and which indices they have.

1.3 Dependencies

Dependencies are great to set up job pipelines without having to wait for the first job to have finished. This can save significant amounts of time if, for example, you have to partition a big calculation into several shorter steps to fit within the cluster's maximum runtaim constraints.

1. As in the previous exercises, inspect the job files for this example and figure out what each line is suppoed to do. The files are `dependency_1.slurm` and `dependency_2.slurm` or their SGE versions ending on `.sge`.

2. You have to start the first job, note its job ID, and then start the second job depending on the first. Check the job status (`squeue` / `qstat` to verify that the second job does not start until the first job has finished):

   SLURM
   ```
   sbatch dependency_1.slurm
   Submitted batch job <jobID>
   sbatch -d afterok:<jobID> dependency_2.slurm
   ```

   SGE
   ```
   qsub dependency_1.sge
   Your job <jobID> ("<jobName>")has been submitted
   qsub -hold_jid <jobID> dependency_2.sge
   ```

   Note: `-d` is the short form of `--dependency` for SLURM, both commands can be used interchangeably.

2 Noon Break Exercise: Putting It All Together

With the help of the slides and previous examples, design three job scripts that do the following:

1. The first script should be a task array with indices 2000, 5000, and 8000. It should execute `./square.sh <task_ID>` and write its output into task-specific output files named `task_<task_ID>.out`. Also add a `sleep 60` command so it takes a bit of time to finish.

2. The second script should also be a task array, each task reading from a single previous task's output file. There are multiple ways to do this in bash, for example:
   ```
   input=`cat task_${SLURM_ARRAY_TASK_ID}.out`
   ```
   (for SGE, use `${SGE_TASK_ID}` instead). You can then use the 'input' variable to execute
   ```
   ./add_self.sh $input
   ```
   Again, add a `sleep 60` command. Reminder: You will need SLURM's `--dependency=aftercorr` or SGE's `-hold_jid_ad` command for this to work, and the second array must have *exactly* the same task indices as the first one. Let the second task array write its outputs in files named `task2_<task_ID>.out`

3. The third script should be a single job that reads all of the second job's output files and makes a sum. You can just execute `./summarize.sh` within the job file, as long as the output files of the previous job are in the `task2_<task_ID>.out` format.

4. When submitting, chain all of the three jobs through dependency directives so you can submit them back-to-back without having to wait for them to finish.

5. Augment the `dependency_wrapper.sh` script so you can submit all three jobs in one go.

6. If you get stuck or keep getting error messages, try to figure out what is wrong with the help of the course materials, and also the links to further documentation below!

Online Resources

- HKHLR Reference Sheets for Hessian HPC Clusters:
  `https://www.hkhlr.de/en/material`
- Cluster-specific links to local documentation for the Hessian HPC Clusters:
  `https://hkhlr.de/en/cluster`
- SLURM manpage for the sbatch command:
  `https://slurm.schedmd.com/sbatch.html`
- SGE manpage for the qsub command:
  `http://gridscheduler.sourceforge.net/htmlman/htmlman1/qsub.html`