Array Jobs

A common problem that users have to solve in an HPC environment is how to best run the same script/application numerous times with only small changes between each run. For example, you may have 100 data sets that you wish to run through a single application on the cluster. The *naive solution* is to manually or programmatically generate 100 shell scripts and then submit them all to the queue.

An alternative solution exists on the HPCC clusters - array jobs. The advantages of using array jobs are:

1. You put considerably less burden on the head nodes.
2. You only have to write a single shell script.
3. You don't have to worry about generating, submitting or deleting hundreds or thousands of shell scripts.
4. If you submit an array job and then realize you made a mistake, you only have one job ID to cancel (scancel), instead of trying to remove hundreds of jobs.

The fact is, there are absolutely no disadvantages to using array jobs. Submitting an array job to do 1000 computations is equivalent to submitting 1000 separate scripts, but is much less work for you and the system.

Keep in mind that this guide will assume that you are knowledgeable in the process of writing submission scripts and submitting normal jobs to the HPCC clusters. If you don't feel confident in performing those tasks, please first read the Job Submission Guide.

Table of Contents

Writing and Submitting an Array Job

The process for writing and submitting an array job is near identical to the process used to write and submit standard jobs on the HPCC Clusters - see Job Submission Guide. The primary difference is with the addition of a few new "environment variables" and a new argument added to your job submission script.

Scheduler Parameters for Array Jobs

Similar to submitting a standard job, every array job that runs requires a submission script that contains the parameters for the job scheduler as well as the commands you wish to run on the partition. Running an array job requires you to add an additional parameter (**-a | --array**) to the submission script. The -a parameter will look as follows:

```
#SBATCH -a <start_num>-<end_num>:<increment>
```

Such that

- <start_num>
  - Is an integer that is >0
  - Defines the task id for the first task in the array job.
- <end_num>
  - Is an integer that is >start_num
  - Defines the task id for the last task in the array job.
- increment
  - Is an integer that is >0
  - Defines the increment between task ids.

For example:

```
#SBATCH -a 1-37:6
```

Would set the first task id as 1 and then run every 6th task id that is <=37. In this case that would be tasks 1,7,13,19,25,31 and 37.

Note: *You are not required to provide an <increment> value. If one is not provided it will default to 1.*

Environment Variables for Array Jobs

When writing a submission script, you may have noticed the use of some environmental variables that would exist once the job has been submitted. These included variables such as $SLURM_JOB_ID and $SLURM_JOB_NAME which would correspond to the ID given to the job by the scheduler and the name of the job defined by the -J or --job-name option, respectively. While writing submission scripts for array jobs, the scheduler will also define a few additional variables that your script can make use of. These new variables are defined below:

- $SLURM_ARRAY_JOB_ID
  - This will be the ID of your first submitted array job, or you can refer to this as the main ID for your entire array job.

- $SLURM_JOB_ID
  - This will be the ID of each of your job in the array.
- $SLURM_ARRAY_TASK_ID
  - When the code in your submission script is executed on a compute node, this will be the task ID associated with the currently running task.
- $SLURM_ARRAY_TASK_MIN
  - This environment variable contains the integer value defined by <start_num> in your -a parameter, which is the lowest index value set in the array.
- $SLURM_ARRAY_TASK_MAX
  - This environment variable contains the integer value defined by <end_num> in your -a parameter, which is the highest index value set in the array.
- $SLURM_ARRAY_TASK_COUNT
  - This environment variable contains the total number of tasks in the job array.

Filename Parameters for Array Jobs

There are two options available to specify the output and error files for your array job in the submissions script:

- %A
  - This will be the value of $SLURM_ARRAY_JOB_ID
- %a
  - This will be the value of $SLURM_ARRAY_TASK_ID

For example:

```
#SBATCH -o array-%A_%a.out
#SBATCH -e array-%A_%a.err
```

Would set the output filename and error filename to “array-30_1.out” and “array-30_1.err” respectively if your job array ID was 30 and the current task ID was 1.

Nocona Array Job Submission Tutorial

Submitting an array job on Nocona can be done using the following steps:

Step 1. Log on to login.hpcc.ttu.edu using your eRaider account and password.

Step 2. Create a new directory then copy the tutorial job script for Nocona to your new folder.

```
mkdir arrayTest
cd arrayTest/
cp /lustre/work/examples/nocona/array.sh .
ls
```

You now have a copy of the array job submission script file for this particular tutorial. To read this file we can use the command *cat array.sh* which will print out the contents of this script.

```
login:/arrayTest$ cat array.sh
#!/bin/bash
#SBATCH -J ArrayTestJob
#SBATCH -p nocona
#SBATCH -o array-%A_%a.out
#SBATCH -e array-%A_%a.err
#SBATCH -N 1
#SBATCH -n 128
#SBATCH -t 01:00:00
#SBATCH --mem-per-cpu=3G
#SBATCH -a 1-37:6

#The variable $SLURM_ARRAY_TASK_ID is the ID for this task.
#The variable $SLURM_ARRAY_TASK_MIN is the ID for the first task.
#The variable $SLURM_ARRAY_TASK_MAX is the ID for the last task.
if [[ $SLURM_ARRAY_TASK_ID == $SLURM_ARRAY_TASK_MIN ]]; then
     position="first"
elif [[ $SLURM_ARRAY_TASK_ID == $SLURM_ARRAY_TASK_MAX ]]; then
     position="last"
else
     position="neither"
fi

#The variable $SLURM_ARRAY_JOB_ID is the ID for the entire array job
#The variable $SLURM_JOB_ID is the ID for each job in the array
echo "Array Job ID: $SLURM_ARRAY_JOB_ID"
echo "Job ID: $SLURM_JOB_ID"
echo "Task ID: $SLURM_ARRAY_TASK_ID"
echo "First or last: $position"
```

This script will request to run 7 tasks (task ids: 1, 7, 13, 19, 25, 31 and 37) and for each task the node will print out the array job ID, job ID, task ID, and whether the task was the first task, last task or neither.

Step 3. We will now submit the array job to the Nocona partition using the sbatch command.

```
sbatch array.sh
```

Step 4. Once your job has completed, list the directory and view some of the newly created output files. For each task ID, you will see the following files:

- array-<array_job_id>_<task_id>.out
    - This file will contain the output for your task.
- array-<array_job_id>_<task_id>.err
    - This file will contain any errors generated by your task.

Step 5. Edit the array.sh file and change the line *"#SBATCH -a 1-37:6"* to instead say *"#SBATCH -a 1-6"*. Now rerun steps 3 and 4 and see what has changed. Notice how the array job now only ran 5 array tasks with IDs 1,2,3,4, and 5. Keep in mind that if you do not specifically set an increment, it will default to an increment of 1.

Step 6. Congratulations, you have now successfully set up and run an array job on Nocona!

Matador Array Job Submission Tutorial

Submitting an array job on Matador can be done using the following steps:

Step 1. Log on to login.hpcc.ttu.edu using your eRaider account and password.

Step 2. Create a new directory then copy the tutorial job script for Matador to your new folder.

```
mkdir arrayTest
cd arrayTest/
cp /lustre/work/examples/matador/array.sh .
ls
```

You now have a copy of the array job submission script file for this particular tutorial. To read this file we can use the command cat array.sh which will print out the contents of this script.

```
login:/arrayTest$ cat array.sh
#!/bin/bash
#SBATCH -J ArrayTestJob
#SBATCH -p matador
#SBATCH -o array-%A_%a.out
#SBATCH -e array-%A_%a.err
#SBATCH -N 1
#SBATCH -n 40
#SBATCH -t 01:00:00
#SBATCH --mem-per-cpu=8G
#SBATCH -a 1-37:6
#SBATCH --gpus=2
```

```
#The variable $SLURM_ARRAY_TASK_ID is the ID for this task.
#The variable $SLURM_ARRAY_TASK_MIN is the ID for the first task.
#The variable $SLURM_ARRAY_TASK_MAX is the ID for the last task.
if [[ $SLURM_ARRAY_TASK_ID == $SLURM_ARRAY_TASK_MIN ]]; then
     position="first"
elif [[ $SLURM_ARRAY_TASK_ID == $SLURM_ARRAY_TASK_MAX ]]; then
     position="last"
else
     position="neither"
fi

#The variable $SLURM_ARRAY_JOB_ID is the ID for the entire array job
#The variable $SLURM_JOB_ID is the ID for each job in the array
echo "Array Job ID: $SLURM_ARRAY_JOB_ID"
echo "Job ID: $SLURM_JOB_ID"
echo "Task ID: $SLURM_ARRAY_TASK_ID"
echo "First or last: $position"
```

This script will request to run 7 tasks (task ids: 1, 7, 13, 19, 25, 31 and 37) and for each task the node will print out the array job ID, job ID, task ID, and whether the task was the first task, last task or neither.

Step 3. We will now submit the array job to the Matador partition using the sbatch command.

```
sbatch array.sh
```

Step 4. Once your job has completed, list the directory and view some of the newly created output files. For each task ID, you will see the following files:

- array-<array_job_id>_<task_id>.out
    - This file will contain the output for your task.
- array-<array_job_id>_<task_id>.err
    - This file will contain any errors generated by your task.

Step 5. Edit the array.sh file and change the line "*#SBATCH -a 1-37:6*" to instead say "*#SBATCH -a 1-6*". Now rerun steps 3 and 4 and see what has changed. Notice how the array job now only ran 5 array tasks with IDs 1,2,3,4, and 5. Keep in mind that if you do not specifically set an increment, it will default to an increment of 1.

Step 6. Congratulations, you have now successfully set up and run an array job on Matador!