FUNDAMENTALS OF JOB MANAGEMENT

Ryan DeRue, Senior Computational Scientist

PURDUE UNIVERSITY | Rosen Center for Advanced Computing

Job Management

Outline

What to expect from Fundamentals of Job Management

Objectives

- Spend some time talking about developing robust workflows
- Discuss a few of the tools available on our clusters for analyzing resource utilization over the duration of a job
- Spend some time discussing Slurm's mechanisms for job management and building pipelines as it relates to creating workflows
- Look at a few of Slurm's other constructs related to job/task management

Job Management

Healthy Computational Workflows

Workflows

What is a “Workflow”?

- A workflow is a of set of steps taken to achieve some objective
- An example scientific workflow might look like:
  - Staging input data to a faster file system
  - Pre-processing that input data
  - Performing a simulation using that input data
  - Post-processing the output data
  - Archiving the output data to permanent storage
- Crucially, a workflow should be easily repeatable

How to Establish a Robust "Workflow"?

- You need to understand the computational resource requirements for each of the steps in your workflow
- We want to encapsulate the workflow into code where applicable. This enables automation and provides a degree of self-documentation
- As much as possible, we want to automate our workflow by building pipelines where each discrete computational step automatically triggers the next

Job Management

Job Resources

Gauging Resource Requirements

- Frequently we are asked, “How can I know how much memory to request for my job ahead of time?”
  - You probably can’t
- Probing resource requirements
  - Request a lot of resources once and use accounting information to inform your subsequent requests
    - Jobs with less resources not only “cost” less but are also start more quickly because of Slurm backfilling
- This requires telemetry tools to monitor resource consumption

Tools for Monitoring Resource Consumption

- Slurm Accounting Data
  - Slurm maintains a database of data about previous jobs
    - We can query Slurm with the appropriate sacct commands
  - Use the jobinfo command which makes these calls for you
    - Syntax: jobinfo <job_id>
  - Useful for post-mortem analysis
- Resource Monitoring Software
  - There is also software installed on the system which can monitor utilization in real-time (nvidia-smi, top/htop, etc.)
  - RCAC deploys a wrapper to this software as a module named “monitor”

Job Resources

The Monitor Utility

- Brief reproducible example:
  1. Use “module load monitor” to expose the tool to your shell**
     1. Use ”man monitor” to consult the available options
  2. Use “monitor cpu percent --all-cores” to begin logging cpu utilization to standard output

- Typically, you will want to run this monitor utility alongside your application code and redirect the output to a separate log file
  - Options also exist to change how frequently telemetry data is sampled as well as the format of the data

Typical Job Script Layout

```
#!/bin/bash

<Slurm options>

module load monitor
monitor cpu memory > cpu-memory.log &
CPU_PID=$!

<Application code>

kill -s INT $CPU_ID
```

** On Purdue Community Clusters such as Negishi, you Will first need to load the utilities module as well

How Should Resource Utilization Data Inform my Request?

- Scenario: “I see very low utilization across my cores, but my memory utilization is close to the amount of memory I requested. What is going on?”
  - On Purdue’s community clusters as well as on Anvil, allocation of memory and cores are made proportional to one another.
    - You can query this ratio by examining the value of “DefMemPerCPU” for the partition you are submitting to using “scontrol show partition <partition_name>”
  - In this case a reduction in CPUs would likely cause a job to fail with an Out of Memory (OOM) error, however you have identified an opportunity for parallelism!

How Should Resource Utilization Data Inform my Request?

- Scenario: “I see I am using about 8GB of memory, and I am running a serial application.”
  - Using the DefMemPerCPU value we examined in the last scenario, I can tailor my request to use slightly above 8GB of memory and reap the scheduling benefits of small resource requests.

Job Management

Slurm Job Dependencies

Slurm Job Dependencies

Job Dependencies

- A method of adding control flow logic to a batch workflow made up of many steps
- Job dependencies allow us to queue jobs to start conditionally rather than as soon as resources are available
- Syntax:
  - `--dependency=<type>:job_id[:job_id]`

Type	Holds Job Until
`after`	After job_id starts
`afternotok`	After job_id terminates with an error
`afterok`	After job_id terminates with an exit code of zero
`afterany`	After job_id terminates with any status
`singleton`	After “my” jobs with the same name terminate

Table outlining some of the most common dependency types used. For a complete list, see the documentation.

Examples of Using Job Dependencies to Support Workflows

- We can use job dependencies to break our workflow into steps, but submit these steps in a single script
  - This is useful because we can use fewer resources for tasks like data transfer
- Let's assume we have the following steps:
  1. Data transfer
  2. Data pre-processing
  3. Simulation
  4. Data post-processing
  5. Data archival

Slurm Job Dependencies

Examples of Using Job Dependencies to Support Workflows

stage_data.sub

```
#!/bin/bash
…

<directory Set up>

#Perform data transfer
source=/depot/labname/data/dataset1.tar.gz
target=$SCRATCH/experiment1/data/dataset1.tar.gz

#Extract dataset from the tarball
tar -xf $target --directory=$(dirname $target)
```

- This script could take quite awhile to run if the dataset is very large
  - By excluding the computationally intensive part of my workflow from this script, I am not charged for the large number resources required to perform that work for a step that requires very little resources.
  - I could submit this job with a request of only a single core or two.

Slurm Job Dependencies

Examples of Using Job Dependencies to Support Workflows

preprocess.sub

```
#!/bin/bash
…

#Activate my anaconda environment
module load anaconda
conda activate experiment1

raw_data_dir=$SCRATCH/experiment1/data/dataset1
clean_data_dir=$SCRATCH/experiment1/data/cleaned/

python preprocess.py $data_dir $clean_data_dir
```

- This script may require more resources than a simple data transfer, but will require less than my simulation step
  - Again, I save the difference in resources while this step runs.

Slurm Job Dependencies

Examples of Using Job Dependencies to Support Workflows

simulate.sub

```
#!/bin/bash
…

#Activate my anaconda environment
module load anaconda
conda activate experiment1

input_data=$SCRATCH/experiment1/data/cleaned/
output_data=$SCRATCH/experiment1/data/output/
python simulate.py $input_data
```

- This step represents the bulk of my computational work, and it will require the most amount of resources
  - However, I am charged for those resources *only* when I need them
- By breaking it up this way, if I encounter an error, I can debug my code and more easily re-run this section of my workflow while avoiding redundant work

Slurm Job Dependencies

Examples of Using Job Dependencies to Support Workflows

postprocess.sub

```
#!/bin/bash
…

#Activate my anaconda environment
module load anaconda
conda activate experiment1

input_dir=$SCRATCH/experiment1/data/output/
output_dir=$SCRATCH/experiment1/data/processed/

python postprocess.py $input_dir $output_dir
```

- This step is logically symmetric to the reasoning behind using a separate preprocessing step.
  - I can scale down my resource consumption immediately after my work is finished

Slurm Job Dependencies

Examples of Using Job Dependencies to Support Workflows

archive_data.sub

```
#!/bin/bash
…

<directory Set up>

#Define directories
output_dir=$SCRATCH/experiment1/data/
destination=/depot/labname/data/experiment1/

#Create tarball of output in permanent storage
tar -czvf $destination/experiment1_output.tar.gz \
$output_dir
```

- Because scratch storage is not backed up and is regularly purged, a robust workflow immediately backs up output to a more permanent location

Slurm Job Dependencies

Examples of Using Job Dependencies to Support Workflows

experiment1.sh

```
#!/bin/bash
#Define directory containing previous scripts
submission_script_dir=/depot/labname/data/experiment1/scripts/

#1. Stage Data
stage_id=$(sbatch -N 1 -n 1 $submission_script_dir/stage_data.sub)

#2. Preprocess Data
preprocess_id=$(sbatch --dependency=afterok:$stage_id -N 1 -n 4 $submission_script_dir/preprocess.sub)

#3. Run Simulation
simulate_id=$(sbatch --dependency=afterok:$preprocess_id -N 1 -n 128 $submission_script_dir/simulate.sub)

#4. Postprocess Data
postprocess_id=$(sbatch --dependency=afterok:$simulate_id -N 1 -n 4 $submission_script_dir/postprocess.sub)

#5. Archive Data
sbatch --dependency=afterany:$postprocess_id -N 1 -n 1 $submission_script_dir/archive_data.sub)
```

Slurm Job Dependencies

Other Ways to use job dependencies

- Restarting jobs from a checkpoint file that timed out
  - --dependency=notok:<job_id>
    - Be advised this will run for any non-zero exit code!
- Starting jobs after a job serving a database has started
  - --dependency=after:<job_id>+<time>
- Ensuring that jobs which require a license do not attempt to check out more copies of a license than you own
  - --dependency=singleton

Job Management

Slurm Job Arrays

Job Arrays

- A mechanism for launching many similar jobs at once all with identical parameters
  - Job arrays have many use cases, but one of the most common is parameter sweeps
  - Instead of having 100 workflow scripts testing different inputs, have 1 that launches 100 pipelines!
- Syntax:
  - `--array=<indices>`
  - Valid indices are 0 through `MaxArraySize -1`:
    - "`1-32`" Create 32 jobs with indices between 1 and 32
    - "`2,4,8,16,32`" Create 5 jobs with indices equal to 2,4,8,16,32 respectively
    - "`1-31:2`" Create 16 jobs with indices 1,3,5,7,...,31
  - You can limit the number of array jobs which are allowed to run at once by using the "%" character when specifying indices.
    - "`1-16%2`" Create 16 jobs, but only allow two to run at a time
  - You can query `MaxArraySize` using `scontrol show conf | grep MaxArraySize`

Job Arrays

- If all the jobs in the array are launched with identical parameters, how do I accomplish different work?
- Important environment variable: `$SLURM_ARRAY_TASK_ID`
  - This variable is equal to the index of the job in the array
    - We can have jobs fetch their input parameters based on this value
    - We can separate jobs into different directories named by this value

Environment Variable	Format Code	Description
`$SLURM_ARRAY_JOB_ID`	%A	JobID of overarching array submission
`$SLURM_ARRAY_TASK_ID`	%a	JobID of the current array index

Slurm Job Arrays

Example

array_submission.sub

```
#!/bin/bash
…
#SBATCH --array=1-10
#SBATCH --output=example_%a.out

input_dir=/depot/labname/data/experiment2/inputs
output_dir=$SCRATCH/experiment2/outputs/

#Create Job Array working directory
mkdir -p $output_dir/$SLURM_ARRAY_TASK_ID

#Launch application code
input_file=$input_dir/input_$SLURM_ARRAY_TASK_ID.in
./mdapplication $(cat $input_file)
```

```
/depot/labname/data/experiment2/inputs/
    -input_1.in
    -input_2.in
    -input_3.in
        …
    -input_10.in
```

```
--dt=0.0025 -x 10 -y 10 -z 10
```

Job Arrays

- Job arrays can also be used with job dependencies
  - The “types” of dependencies behave slightly differently with job arrays.
  - When dependent upon a jobID specifying a Slurm job array:
- An example of when this might be useful is when a workflow may make a comparative analysis of a parameter sweep
  - An “afterok” dependency could launch a job that plots the results of using different parameters after a post-processing step

Type	Holds Job Until
`after`	job begins after all tasks in the job array begin
`afternotok`	job begins after all tasks in the job array terminate *with at least one non zero exit code*
`afterok`	job begins after all tasks in the job array terminate with exit code zero
`afterany`	job begins after all tasks in the job array terminate

Job Management

Job Level Parallelism

Job Level Parallelism

Carving up a Slurm Allocation

- Up until now, every example I have shown has dedicated all the resources under the Slurm allocation to each command
- We can also execute several commands simultaneously each using a portion of the resources within a job script
- The “srun” command allows us to specify the resources we want to allocate to a particular task using the same syntax as “sbatch”
  - The constraint is that across all our tasks, we cannot allocate more resources than was given to us in our original request

```
#!/bin/bash
…
#SBATCH –N1 –n12

…

srun –n 6 ./scriptA &
srun –n 2 ./scriptB &
srun –n 2 –c 2 ./scriptC &
wait
```

Job Level Parallelism

Heterogeneous Jobs

- In the previous example, all jobs shared from the same pool of resources and had the same Slurm parameters
- It is possible to define a job which consists of many different job components each with its own Slurm parameters
  - Such a job is referred to as a heterogeneous job
- This works by defining different “hetgroups” within a jobscript and launching commands under the hetgroup you wish them to be a part of
- Heterogeneous jobs are a relatively recent addition to Slurm and have some nuances which can cause unexpected results so take care when using them.

```
#!/bin/bash
…
#SBATCH -N1 -p wholenode
#SBATCH -n128
#SBATCH hetjob
#SBATCH -p shared
#SBATCH -n 8
…

srun -hetgroup 0 -n 128 \
./largeScript &

srun -hetgroup 1 -n 2 \
./smallScript &

srun -hetgroup 1 -n 6 \
./mediumScript &
```

Job Management

Conclusion

Conclusion

Takeaways

- Developing workflows which can be broken into discrete steps is not only more maintainable but also more computationally efficient
- Slurm provides constructs like Job Dependencies and Job Arrays for the purpose of making the creation of these pipelines easier
- Constructs also exist for command level parallelism in both homogeneous and heterogeneous jobs using Slurm tooling

THANK YOU

Feel free to reach out to rderue@purdue.edu with questions.

Slides are posted at:
https://www.rcac.purdue.edu/training/jobmanagement