HPC Workflows using Slurm

Introduction to Slurm Scheduling
Working with Python on Aristotle

Vassilis Asteriou
April 2023

In this talk

Introduction to Slurm

Slurm Examples

Working with Python on Aristotle

Introduction to Slurm

HPC Scheduling

Why is scheduling needed on the “Aristotle” cluster?

1. Share finite resources among multiple users
2. Manage allocation of resources in a distributed heterogeneous environment
3. Bookkeeping, efficieny monitoring, statistics

Slurm Workload Manager

- Slurm is a scheduling and workload management system for HPC environments
- Functions of Slurm
  1. Allocates and manages exclusive users access to cluster resources
  2. Provides a framework for job tracking and parallel job execution
  3. Arbitrates contention by queuing pending work

`https://slurm.schedmd.com/quickstart.html`

Using Slurm to Access HPC Resources

1. Users can schedule work to be executed on the cluster by submitting **jobs** to Slurm.
2. Jobs submissions include a user-defined specification of the **resources** required for the workload associated to the job.
3. Slurm will **queue** jobs and schedule them for execution when the requested resources become available.

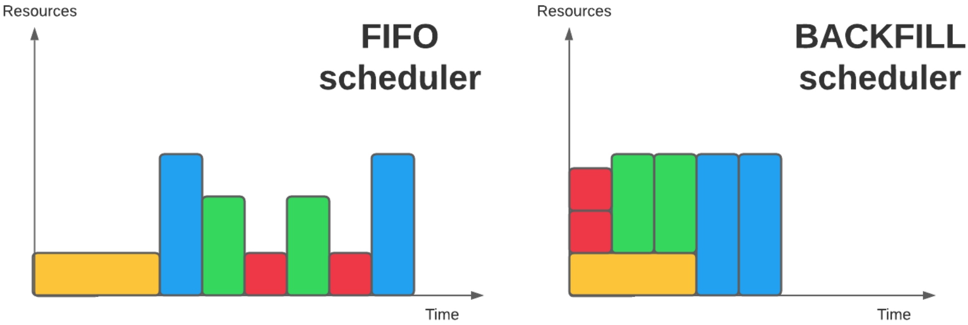
Slurm's Scheduling Algorithm

- Multifactor Priorities
  1. Age: time a job has spent in queue
  2. Size: quantity of resources requested (e.g. CPU cores, time)
  3. Fair share: decreases per user priority proportional to recently allocated resources
- Backfill Scheduling
  1. Highest Priority First Scheduling
  2. Start lower priority jobs *only* if it does not impact the *expected start time* of any higher priority jobs

Links:

- Fair Share `https://slurm.schedmd.com/fair_tree.html`
- Backfill `https://slurm.schedmd.com/sched_config.html#backfill`

Slurm's Scheduling Algorithm



Slurm Examples

Getting to know Slurm

- `sinfo` Show status of available partitions
- `sinfo -N --long` Show node status
- `squeue` Show status of running and queued jobs
  - `-u <username>` Filter results for one user
  - `-p <partition>` Filter results for one partition

Example 1: A test job

Steps:

1. Create a submission script
2. Submit job to Slurm
3. Monitor job execution
4. Get job results

Related docs `https://hpc.it.auth.gr/jobs/serial-slurm/`

Example 1: A test job

Submission script

```bash
#!/bin/bash

#SBATCH --time=10:00
#SBATCH --partition=testing

echo "Hello from $(hostname)"
sleep 30
echo Bye
```

Example 1: A test job

Job submission

1. `sbatch <submission-script.sh>`
2. Use `man sbatch` for more options

Job monitoring

1. `squeue -j <jobid>`
2. `tail -f slurm-<jobid>.out`
3. `sacct -j <jobid>`

Cancel jobs

1. `scancel <jobid>`

Example 2A: More CPU Cores

```
#!/bin/bash

#SBATCH --partition=testing
#SBATCH --time=10:00
#SBATCH --cpus-per-task=4

stress --cpu ${SLURM_CPUS_PER_TASK} --timeout 60
```

CPU Efficiency: `seff <jobid>`

Example 2B: MPI Parallelization

- Message Passing Interface (MPI) is a system for distributed parallel application development
- Gromacs is a molecular dynamics simulation tool
- Gromacs supports MPI Parallelization and Gromacs jobs can benefit from increasing CPU core count
- For scalable use cases, increasing cpu cores reduces job walltime

`srun [...]` launch Slurm-managed MPI process Documentation

`https://hpc.it.auth.gr/applications/gromacs/`

Example 2B: MPI Parallelization

```
#!/bin/bash

#SBATCH --partition=rome
#SBATCH --time=10:00
#SBATCH --nodes=1
#SBATCH --tasks-per-node=32
#SBATCH --cpus-per-task=1

module load gcc/9.4.0-eewq4j6 openmpi/4.1.2-akxtzzl
module load gromacs/2022-47qrtrj

srun gmx_mpi mdrun -ntomp 1 -s ../benchMEM.tpr
```

Example 3: More Memory

Default memory allocation

- Proportional to number of requested cores

Example:

- `batch` partition nodes have 20 cores and 128G of RAM each
- a job for partition `batch` with 10 cores will request by default 64GB of RAM.

What happens if a job tries to use more memory than it is allocated?

Example 3: More Memory

```
#!/bin/bash

#SBATCH --partition=testing
#SBATCH --job-name=memory
#SBATCH --time=4:00


./allocate-10gb
```

Example 3: More Memory

```
#!/bin/bash

#SBATCH --partition=testing
#SBATCH --job-name=memory
#SBATCH --time=4:00
#SBATCH --mem=11G

./allocate-10gb
```

Example 4: GPU jobs

```bash
#!/bin/bash

#SBATCH --partition=gpu
#SBATCH --gres=gpu:1
#SBATCH --cpus-per-task=20
#SBATCH --time=10:00

nvidia-smi
```

Example 5: Allocating Licenses

- ANSYS Fluent is a computational fluid dynamics (CFD) tool
- ANSYS Fluent is one example of a licensed software available for AUTH users
- A limited number of licenses are available for HPC usage.
- A user can request that a job be scheduled only if there are enough licenses available

Documentation `https://hpc.it.auth.gr/applications/fluent/`

Example 5: Allocating Licenses

```bash
#!/bin/bash

#SBATCH --job-name=FLUENT-2021R1-case
#SBATCH --partition=batch
#SBATCH --ntasks-per-node=20
#SBATCH --nodes=1
#SBATCH --licenses=ansys@ansys.it.auth.gr:16
#SBATCH --time=1:00:00

module load ansys/2021R1

fluent 3ddp -g -ssh -t$SLURM_NTASKS -i elbow_journal.in
```

Documentation

More info

1. `https://hpc.it.auth.gr/jobs/job-submission/`
2. `https://hpc.it.auth.gr/jobs/serial-slurm/`
3. `https://hpc.it.auth.gr/jobs/slurm/`

Working with Python on Aristotle

In this session

- Create a conda environment with PyTorch
- Add the environment to Jupyter server
- Submit a test PyTorch job on a GPU node

Conda environment

- Conda is an open-source package management system and environment management system.
- Miniconda is a free minimal installer for conda.
- Miniconda3 is available on Aristotle.
  `module spider miniconda3` See available versions.

Environment setup 1/4

```bash
#!/bin/bash

#SBATCH --partition=gpu
#SBATCH --gres=gpu:1
#SBATCH --cpus-per-task=20
#SBATCH --time=1:30:00

# Load Miniconda
module load gcc/9.4.0-eewq4j6 miniconda3
source "${CONDA_PROFILE}/conda.sh"
```

Environment setup 2/4

```
# Create pytorch environment
conda create -p ./test.env pytorch==1.12.0 torchvision==0.13.0 \
    torchaudio==0.12.0 cudatoolkit=11.3 \
    -c pytorch
```

Also see `https://pytorch.org/get-started/previous-versions/`

Environment setup 3/4

```
# Check if CUDA is installed properly
conda activate ./test.env

cat <<EOF | python -
import torch
print(torch.cuda.is_available())
print(torch.cuda.device_count())
EOF
```

Environment setup 4/4

```
# Install additional libraries
conda install -c conda-forge jupyter matplotlib tqdm
```

What is Jupyter

- Jupyter is an open-source web application that allows you to create and share notebooks
- Notebooks are documents that contain live code, equations, visualizations and narrative text
- Jupyter is available on Aristotle via Open OnDemand (`https://hpc.auth.gr`)

Setup environment in Jupyter

- Jupyter was installed in environment
- To add the new environment to Jupyter:

```
conda activate ./test.env
python -m ipykernel install \
    --user --name torch-env \
    --display "Torch Test Environment"
```

- Also see `https://hpc.it.auth.gr/applications/jupyter/#custom-python-virtual-environments`

A demo

- PyTorch is a deep learning framework
- The MNIST dataset contains 60,000 training images of *handwritten digits* from zero to nine and 10,000 images for testing.
- The MNIST digit classification problem is almost the *Hello, world!* of deep learning.

This demo is from `https://github.com/jiuntian/pytorch-mnist-example`

A demo

```bash
#!/bin/bash

#SBATCH --partition=gpu
#SBATCH --gres=gpu:1
#SBATCH --cpus-per-task=20
#SBATCH --time=10:00

module load gcc/9.4.0-eewq4j6 miniconda3
source "${CONDA_PROFILE}/conda.sh"
conda activate ../test.env

python pytorch-mnist.py
```

Thank you!