UVA | SCHOOL of DATA SCIENCE

Slurm Introduction

UVA Rivanna

Robert Knuuti <ugq5zz@virginia.edu>

DS 6013 - Summer 2022
Instructor:
Dr. Judy Fox

https://docs.google.com/presentation/d/1mqP0QgYxbESCCDnlptTFp1HJvpAN_RW2tIX5-dDBSQ0/edit?usp=sharing

What is Slurm?

- Cluster and Job management ecosystem
- Provides a series of interactive and batch tools based on a reservation request
- The system ensures that users have confidence that they will have reserved access to system hardware, and to enforce limits on users to prevent abuse.
    - Slurm also provides a cost system based on Service Units that are spent with usage.

- **Fun Fact**: Slurm is what powers all self-service capabilities on Rivanna, if you've requested a Desktop, JupyterLab, or RStudio from the OnDemand portal, you've used a slurm script that the Rivanna team has written to setup these tools.

How to request an Allocation (CLI)

There's two ways

Interactive (ijob)

- You place you request and are placed in a queue.
- Your prompt will pause until the hardware needed for the time of your job can be made
- You are then presented with a prompt within the allocation for the duration of your request
- Once you exit, the allocation is revoked and you are returned to your normal prompt.

Batch (sbatch)

- You write a shell script that has annotations to specify your reservation needs and the commands to be run once the resources are available.
- You issue the command, and the job is registered in the background and you can continue work.
- Your job will be placed in the global queue in the background, allowing you to log off or run asynchronous jobs

Both commands use common arguments keeping the configurations similar

Considerations for writing sbatch jobs

- Slurm itself isn't a complex of a system to use, but writing batch jobs that maximize its abilities requires automation to be implemented.
- Your job is triggered as a shell script (bash), and ends when the script either finishes or encounters an error; it may not be immediately obvious if things worked or not.
- Your job does need to run without any type of user input or feedback (headless)
    - This is where most of the work will be.
    - Tools like lmod, papermill and apptainer/singularity become important.

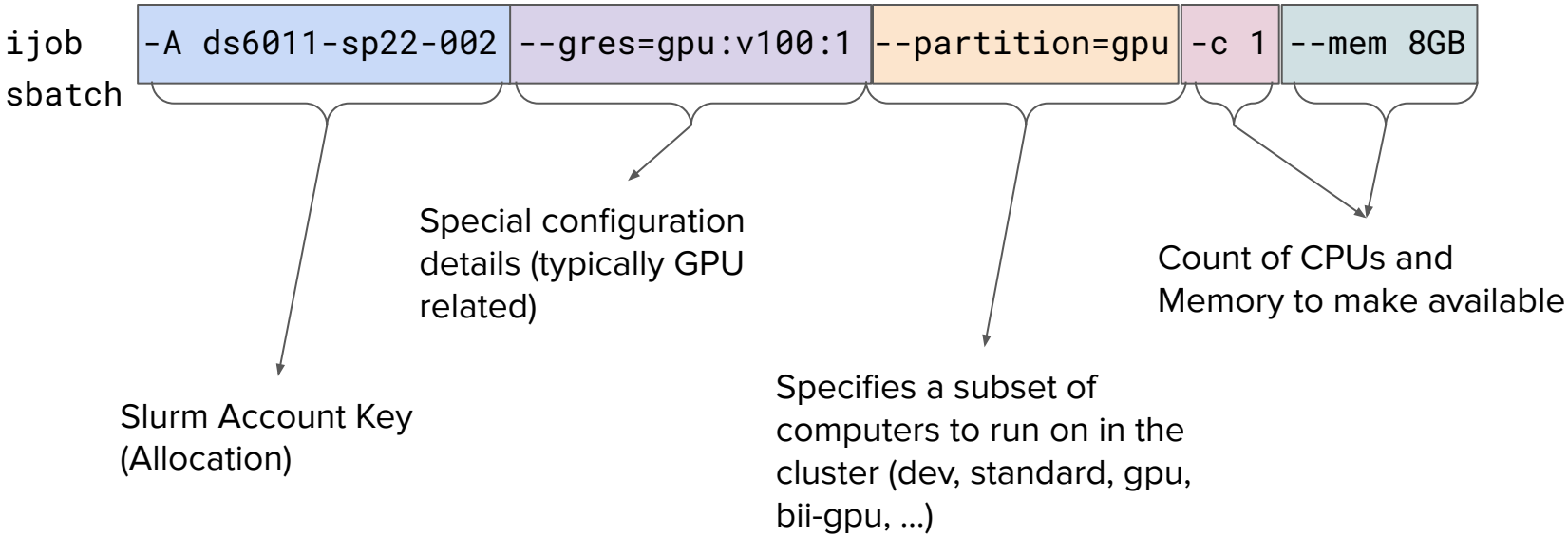
- To convert a shell script to an slurm-aware script, you simply add the CLI arguments with the prefix of `#SBATCH <argument-here>` in the file.

Reminder: you can supply any of the sbatch CLI arguments in these files.
See https://slurm.schedmd.com/sbatch.html for all possible arguments

How to spawn a job

1. Login to Rivanna either via SSH, or by using the cluster access at https://rivanna-portal.hpc.virginia.edu/
2. Position your desired files on rivanna (either check out your code using git, or upload it to your home directory)
3. Request a reservation using **ijob** / **sbatch**

Parts of a Slurm Reservation Request



This is just a subset of all the options.
See https://slurm.schedmd.com/sbatch.html for all possible arguments

Expanding on Slurm Arguments

Partition	Max time / job	Max nodes / job	Max cores / job	Max cores / node	Max memory / core	Max memory / node / job	SU Charge Rate
standard	7 days	1	40	40	9GB	375GB	1.00
parallel	3 days	25	1000	40	9GB	375GB	1.00
largemem	4 days	1	16	16	64GB	975GB	1.00
gpu	3 days	4	10	10	32GB	375GB	3.00 *
knl	3 days	8	512 cores / 2048 threads	512	3GB (per physical core)	192GB	1.00
dev	1 hour	2	8	4	6GB	36GB	0.00

* GPU charge rate = number of cores + 2 * number of GPU devices.

Account → Your UVA ID must be in the allocation listed to be used

You can check your allocations by running the **`allocations`** command on rivanna

Gres → On rivanna follows the pattern `**`gpu:<card_type>:<number_of_gpus>`**`.

So for a job needing 2 v100 GPUs, you'd use the string `**`--gres=gpu:v100:2`**`

Time → You need to specify the total amount of time upfront for your allocation.

Rivanna doesn't allow jobs greater than 3 days in length

There are rules on the maximum concurrently, and your job may go to a queue.

Example sbatch script

```
#!/usr/bin/env bash
#SBATCH --job-name=mydemojob
#SBATCH --output=%u-%j.out
#SBATCH --error=%u-%j.err
#SBATCH --partition=standard
#SBATCH --cpus-per-task=1
#SBATCH --mem=4GB
#SBATCH --time=3:00
#SBATCH --account=ds6011-sp22-002

module load anaconda # we do have custom versions of python in /project/ds6011-sp22-002 for native python.
conda create -y -n demo python=3.9
conda activate demo
# your automation code here...


(The same configuration using ijob)
ijob --job-name=mydemojob --partition=standard --cpus-per-task=1 --mem=4GB --time=3:00 --account=ds6011-sp22-002
# Or in abbreviated values…
ijob -J mydemojob -p standard -c 1 -t 3:00 -A ds6011-sp22-002 --mem=4GB
```

Example sbatch script with GPU

```
#!/usr/bin/env bash
#SBATCH --job-name=mydemojob
#SBATCH --output=%u-%j.out
#SBATCH --error=%u-%j.err
#SBATCH --partition=gpu
#SBATCH -c 1
#SBATCH --gres="gpu:v100:1"
#SBATCH --mem=4GB
#SBATCH --time=3:00
#SBATCH --account=ds6011-sp22-002

module load cuda cudnn
module load anaconda
conda create -y -n demo python=3.9
conda activate demo
# your automation code here...
```

This line is critical for CUDA and Deep Learning frameworks on rivanna.

If you do not load these modules, it's very likely you'll be using a CPU workloads unless you install your own version of GPU libraries.

Try it out!

- Go to - https://rivanna-portal.hpc.virginia.edu/pun/sys/dashboard
- Click on “Clusters” then “Rivanna Shell Access”
- Try to run the below

```
nvidia-smi # this command should fail
# you may need to conda init bash if you haven’t run anaconda on rivanna before
# Request a very small resource allocation (optionally you can use k80, p100, or a100)
ijob -A ds6011-sp22-002 --gres=gpu:v100:1 --partition=gpu -c 1 --mem 8GB --time=10:00
# Wait for allocation; you will see several salloc messages
nvidia-smi # this command should work
module load cuda cudnn anaconda
conda create -y -n rivanna-demo python=3.9 tensorflow-gpu
conda activate rivanna-demo
python
>>> import tensorflow as tf
>>> tf.config.list_physical_devices('GPU')
# you should see a lot of output and an array of the Physical Devices namedtuple
>>> exit()
conda deactivate && conda env remove -y -n rivanna-demo
exit # this command will end your allocation request (automatically happens at --time)
```

Additional Thoughts

- Establish an experiment management plan (complex parameter grids aren't out-of-the-box)
- Build asynchronous feedback loops (create verbose logs)
- Expect lead-time for testing (consider creating dummy jobs to get more rapid feedback)
- Don't overallocate your reservation (this will make you wait in the queue longer)
- Not a silver bullet for performance (other system limitations may still have impacts)

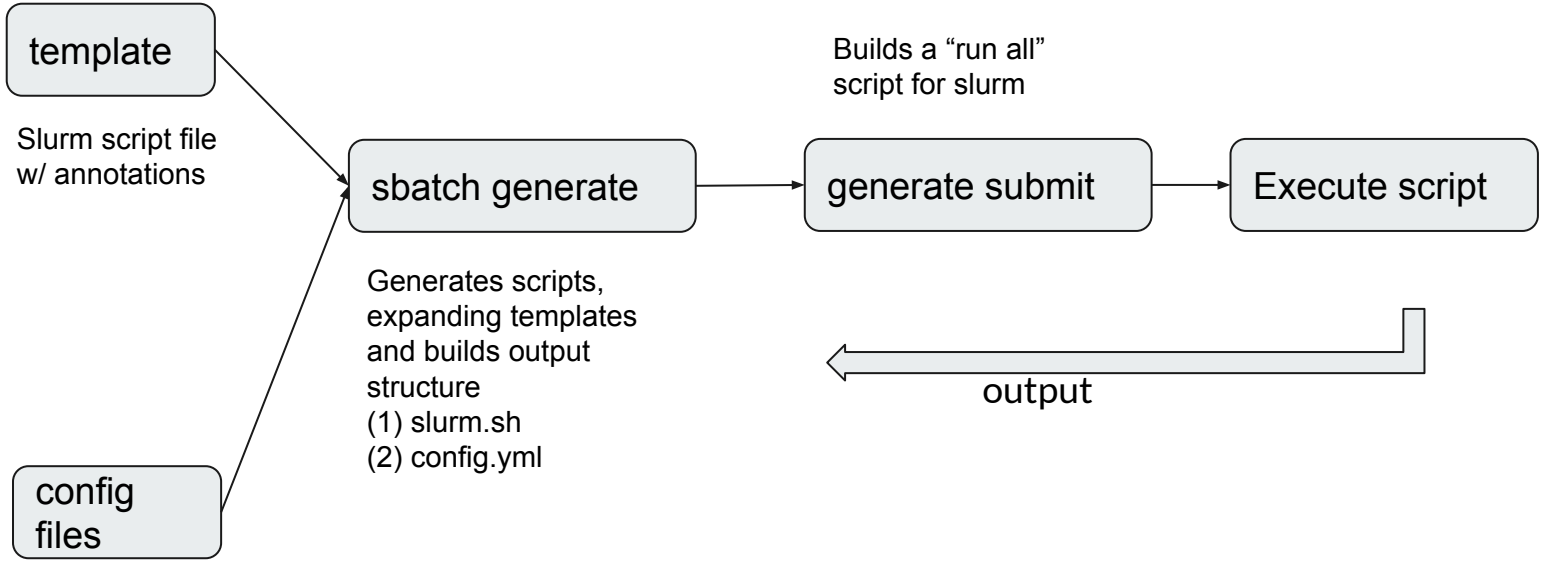
Experiment Management using Cloudmesh Sbatch

- Slurm has a basic configuration system and runs batch jobs well, but does not do parameterization well
    - You can run sbatch and ijob without this tool, but it becomes difficult to scale out experiments consistently.
- Completely lacks a framework for generating multiple permutation based experiments, used when running models as scale

- Cloudmesh Sbatch was built as a templating and parameterization solution to address this gap, so generating 90 different permutations of experiments can be done using a single code baseline.
    - Extremely useful as it automates much of the script generation process

Developed as part of MLCommon-Science efforts (Gregor von Laszewski, Robert Knuuti, Jake Kolessar, Thomas Butler)

How it Works



YAML file including:
(1) attributes (direct subtitutions)
(2) experiments (substitutions that are permutations)
(3) Run configurations

Examples can be found in the sbatch repo:
https://github.com/cloudmesh/cloudmesh-sbatch

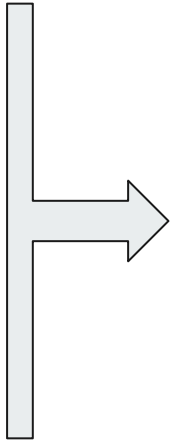
Visual Generation Workflow

Experiment	epoch	x	y	Experiment	epoch	x	y	Experiment	epoch	x	y	Experiment	epoch	x	y
1	1	1	10	**4**	1	4	10	**7**	1	1	11	**10**	1	4	11
2	2	1	10	**5**	2	4	10	**8**	2	1	11	**11**	2	4	11
3	3	1	10	**6**	3	4	10	**9**	3	1	11	**12**	3	4	11

slurm.in.sh

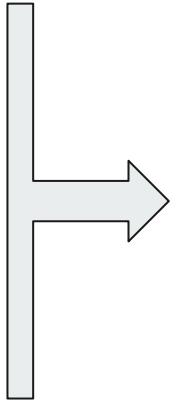
config.yaml

```
name: myexperment
mode:h
dir: example
experiments:
  epoch: “1-3”
  x: “1,4”
  y: “10,11”
attributes:
  cpus: 6
  mem: 32GB
  time: 6:00:00
  rivanna:
    account: ds6011-sp22-002
```



```
#!/usr/bin/env bash
#SBATCH --job-name={id}-{gpu}
#SBATCH --output={id}-{gpu}.out
#SBATCH --error={id}-{gpu}.err
#SBATCH --partition=gpu
#SBATCH --cpus-per-task={cpus}
#SBATCH --mem={mem}
#SBATCH --time={time}
#SBATCH --gres=gpu:{gpu}:1
#SBATCH --account={rivanna.account}

python --epoch={epoch}
```



```
#!/usr/bin/env bash
#SBATCH --job-name=epoch_2-a100
#SBATCH --output=epoch_2-a100.out
#SBATCH --error=epoch_2-a100.err
#SBATCH --partition=gpu
#SBATCH --cpus-per-task=6
#SBATCH --mem=32GB
#SBATCH --time=6:00:00
#SBATCH --gres=gpu:a100:1
#SBATCH --account=ds6011-sp22-002

python --epoch=2
```

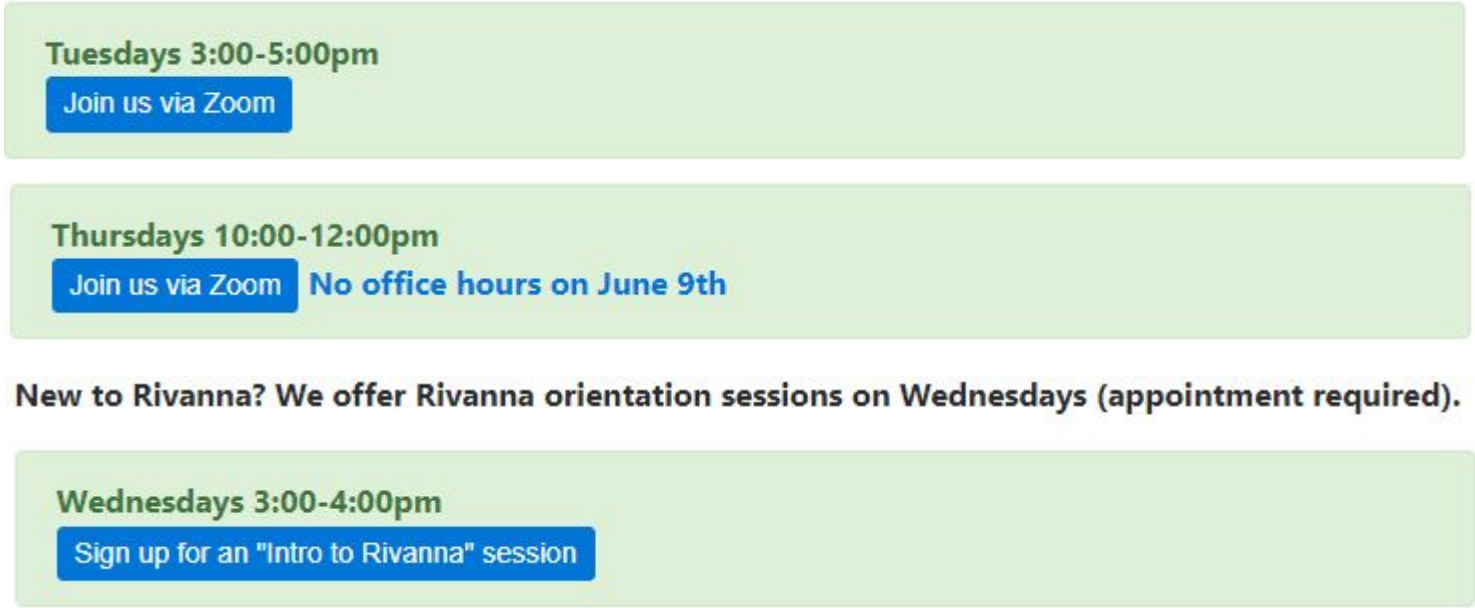

cms sbatch generate slurm.in.sh --setup=config.yaml

Additional Resources

- Rivanna Tensorflow Tutorial
  https://www.rc.virginia.edu/userinfo/rivanna/software/tensorflow/
- Rivanna Slurm Reference / Tutorial
  https://www.rc.virginia.edu/userinfo/rivanna/slurm/
- GPU GRES Line Reference
  https://www.rc.virginia.edu/userinfo/rivanna/slurm/#gpu-computations

Rivanna has regular office hours meetings

See: https://www.rc.virginia.edu/support/



Example Slurm Script

```
#!/bin/bash -l
# --- this job will be run on any available node
#SBATCH --job-name="TF1_2+4_Smoothed_3142"
#SBATCH --partition=bii-gpu
#SBATCH --gres=gpu:v100:1
#SBATCH --time=10:00:00
#SBATCH --mem=120GB
#SBATCH --account=ds6011-sp22-002
#SBATCH --export=NONE

# Load CUDA and CUDNN for training deep learning models
module load cuda cudnn anaconda

# Activate Conda
conda deactivate
conda activate ~/anaconda3/envs/ml

# Automation Code
python tft1_v1_full.py --mode 0 --features 0 4 9 10 12 14 --runname "TF1_2+4_Smoothed_3142" --county_size 3142
```