University of Manitoba

Introduction to HPC: SLURM, best practices for Grex

UofM-Spring-Workshop 2021

April 21st-22nd, 2021

Grigory Shamov

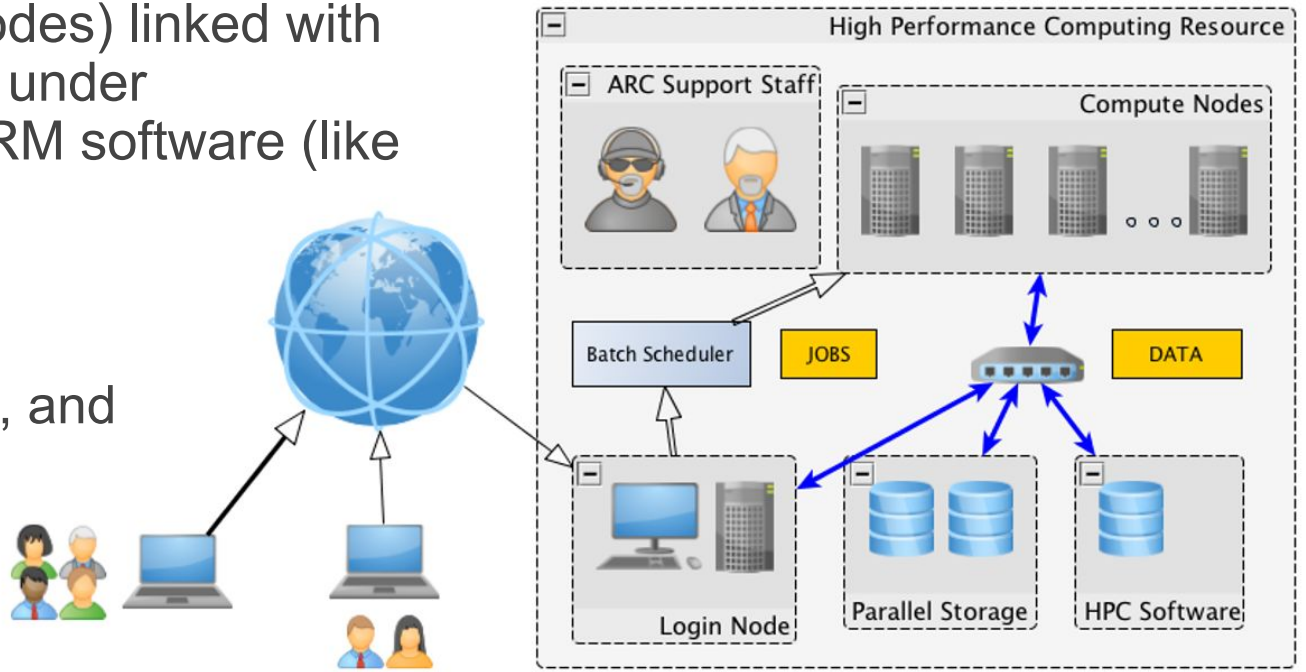
University of Manitoba

University of Manitoba

HPC

An HPC cluster is a collection of compute servers (nodes) linked with an Interconnect and under management of an RM software (like SLURM)

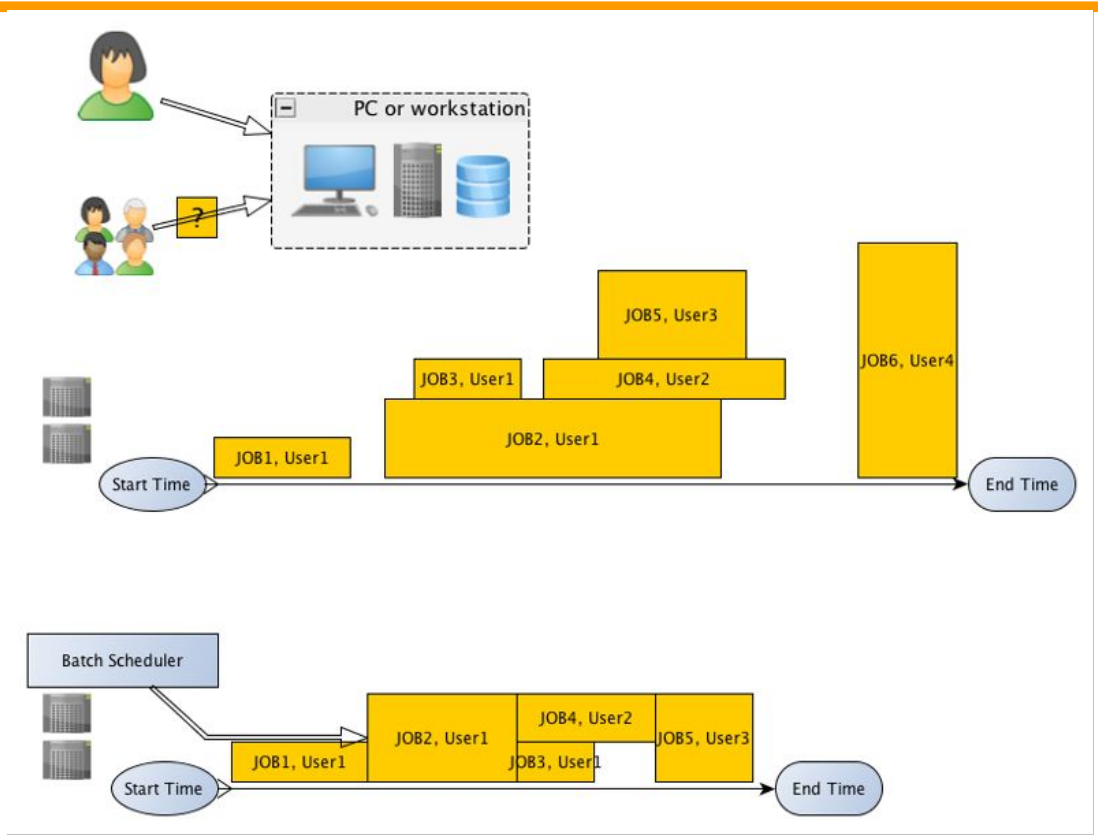
Users connect to it through a login node, and submit batch jobs



University of Manitoba

Why batch mode

- Free for all server
  - Immediate access
  - Resource Congestion
  - Low utilization
- Batch mode (HPC)
  - Queuing involved!
  - “Traffic control”
  - High utilization and efficiency
  - Paralel jobs, large resources



University of Manitoba

What is SLURM

- **S**imple **L**inux **U**tility for **R**esource **M**anagement
  - Probably one of the most complex ones on the market
  - Modular structure, constantly under development
  - Open Source

SLURM is a software for managing the batch mode.

- Prevents resource congestion by users
- Ensures “fairness” of the resource allocation to the users
- Maximizes utilization of the resources by the users

University of Manitoba

Partitions in SLURM

- In practice, the compute nodes are often grouped into “partitions”
- Nodes of different kinds of hardware (old, new, GPUs, memory, ..)
- Same nodes to prioritize different workloads (long vs short jobs, interactive jobs)
- Same or different nodes, to regulate or prioritize access (in case of user-contributed systems, for example)

University of Manitoba

What are the HPC resources?

- CPU cores
- Compute Nodes -- not exactly the same as CPUs!
- Memory (RAM), per node or per CPU core
- GPU devices
- Other resources: disk space, burst buffer space, software licenses etc.

In the batch mode, the above resources are given (allocated) for a “Job” for a time. SLURM prevents other jobs from using these resources during that time.

- Time is the resource too.

University of Manitoba

Policies and Limits

There is a policing engine, the Scheduler, in SLURM that will enforce priorities, allocations, and limits.

- Limits can be per User or Group, in the form of Maximum Walltime, Max number of jobs per User, etc.
- Limits can be per Partition
- Limits can be per QoS policy (not used on Grex or CC)

Priorities are set based on allocation values and recent usage and waiting time in the queue. Priorities is what determines which job can run first (if not blocked by a limit).

University of Manitoba

Simple ..

The “simple” way probably was to use the **srun** command, to run an executable across the selected resources. **srun** would inherit environment from the shell it is run.

Resources can be specified in the command line (**--time, --ntasks, --cpus-per-task, --ntasks-per-node, --mem-per-cpu, --gpus** etc.)

Resources can be communicated to the executable (usually script) via environment variables lik *$SLURM_CPUS_PER_TASK*.

University of Manitoba

Batch and interactive jobs

salloc (which is a wrapper for **srun bash**) is the command to get interactive bash jobs. Returns the interactive shell.

sbatch submits a job script to be ran as batch job.

- Within the job script there can be multiple job “steps” with **srun** or serial codes.
- Within the script, special **#SBATCH** comments can contain the same resource requests as command line options.

University of Manitoba

Getting most out of SLURM

The key is to know what resources are available on a given HPC machine, and to adjust your requests accordingly.

- It is up to the users to figure it out!
- Know what partitions are there, and what are their limits
- Know what is the hardware (how many CPUs per node, how much memory per CPU available, etc.)
- Know if your code is efficient for a given resource
- Know time limits and estimate runtime of your jobs
  - comes after some trial and error, with experience
- Make sure your app obeys the SLURM resource limits

University of Manitoba

Time

There is, as a rule, maximal walltime, per partition

- To prevent stuck jobs to block resources forever
- Will terminate jobs after the requested time is over
- Should we always ask for the maximal --time=?
- The format is --time=days-hh:mm[:ss] ; --time=0-0:30

SLURM is sort of playing a game of Tetris, in time

- it can Backfill jobs out of their order in the queue, if their time is short enough to fit a gap
- Accurate estimates of walltime give better utilization!
- Jobs on preemptible partitions might be terminated no matter what --time they specify (we'll see later)

University of Manitoba

Memory

- There are two resources, memory per node and memory per cpu core.
  --mem=4000mb (per node, --nodes=1 --ntasks=4 takes 4GB/node)
  --mem-per-cpu=4000mb (per core, --ntasks=4 takes 16GB/job)
- Memory limits are enforced; over-use of memory will terminate the job
- “Process Equivalent” : if a memory request takes more than physical memory per core installed, the job blocks more CPUs than requested. --mem=47gb --ntasks=1 takes a whole 12-core old compute node.
- Asking for too much memory results in longer queuing times and waste of resources
- Larger memory partitions should be used for larger memory jobs (--partition=**bigmem** or --partition=**skylake**)

CPUs, cores and nodes 1

These Resources should match the capabilities of the software.
It matters how nodes, tasks and cpus are selected. SLURM would not make software parallel by some magic!

- Some software is serial, and can use only single core
  - This is the default --ntasks=1 --nodes=1 --cpus-per-task=1
- Some codes are “threaded”, or Symmetric Multiprocessing (SMP) or concurrent, etc.
  - can utilize a single node only.
  - can utilize from One to Max number of physical cores on the node
  - must use one task! SLURM would isolate tasks and SMP threads should be using shared memory space.
  - **--nodes=1 --ntasks=1 --cpus-per-task=12 (right)**
  - **--noded=1 --ntasks=12 (try if same as above?)**
- Serial or SMP codes are simply started in the job script. *./mycode*

University of Manitoba

CPUs, cores and nodes 2

The Resources should match capabilities of the software!
It matters how nodes, tasks and cpus are selected

- Some software are “Massively Parallel”, Distributed memory, etc.. Often, Message Passing Interface layer (MPI) is used by them.
- These can span multiple nodes, running multiple tasks
- Each parallel process needs a task allocated for it
- Depending on the MPI code assumptions, it might matter how the tasks are spread across the nodes
  - **--ntasks=N** (random node/task layout; easiest to schedule)
  - **--nodes=P --ntasks-per-node=Q** (when P*Q = N; force an uniform task distribution)
- Each parallel job needs its tasks to be spawned, on the allocated cores (and nowhere else!). This might depend on MPI
  - best using **srun** in the job scrit (if the MPI allows for it)
  - Next best using **mpiexec** with some SLURM support (mpiexec.hydra --rmk=slurm)
  - Using SSH as a common fallback (worst case, ask for whole nodes to minimize side effects)

University of Manitoba

GPUs

New SLURM code (since 2019) supports GPU detection and the new GPU resources syntax.

- **--gpus=N** (N per job)
- **--gpus-per-node=N --nodes=2** (N per node)
- **--cpus-per-gpu=6**
- Not all options' combinations make sense. --ntasks-per-node and --cpus-per-gpu wont coexist
- On Grex, a partition with GPU hardware must also be selected for --gpus options (--partition=gpus)
- On Grex, selecting a partition with GPUs requires one of --gpu options set. Only whole GPUs can be requested (no MPS, no virtual GPUs yet).

University of Manitoba king software to heed SLURM Limits

Memory, number of CPU cores, threads often set in Input. It should match the resource request from SLURM!

- **srun** sets correct cpu cores and nodeilists automatically for MPI jobs. It also takes care of the proper process placement.
- for SMP jobs, use $SLURM_CPUS_PER_TASK variable to set number of threads.
  - export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
- for jobs using MKL or other threaded BLAS/LAPACK, set MKL_NUM_THREADS either to 1 or to $SLURM_CPUS_PER_TASK, depending on what you are doing.
- It is a good practice to set memory request 10-15% smaller in the input than in the SLURM request, to allow for file buffers, etc.

University of Manitoba

Partitions

Grex is a very heterogeneous system now!

- Old compute nodes (12 cores, 48 GB RAM) **--partition=compute**
- Old compute with more memory (94GB RAM) **--partition=bigmem**
- New compute with more memory (40 cores, 384 GB RAM) **--partition=skylake**
- Newest compute with less memory (52 cores, 96GB RAM) --partition= TBD
- New GPU partition (32 cores, 192GB RAM, 4x V100 GPUs) **--partition=gpu**

Unlike ComputeCanada, partition selection is manual, except that by default, short jobs would go to **compute** and longer to **skylake**.

University of Manitoba

More Partitions

Grex is a home for several contributed systems.

- Newest contributed GPU server (48 cores, 1.5TB RAM, 16x V100 GPUs) **--partition=livi-b**
- Contributed GPU partition, three nodes (32 cores, 192GB RAM, 4x V100 GPUs) **--partition=stamps-b**
- The contributors have preemptor partitions (livi, stamps); jobs in **-b** partitions can be preempted after a grace period (1h now).
- The purpose of preemptable partitions is to let others utilize the contributed resources when these are not used by the owners

Unlike ComputeCanada, there is no partitions related to walltime or size of the jobs on Grex.

How to find out about resources?

- By reading documentation, first. https://monitor.hpc.umanitoba.ca/doc/
- Short of exploring the Linux commands in an interactive job, sinfo is the command to list partition time limits
  sinfo ; sinfo -p skylake
- The more general command scontrol will show parts of SLURM configuration about nodes and partitions
  scontrol show partition=gpu
  scontrol show node=g324
- The seff command shows efficiency of finished jobs; sacct shows job records
  sacct -u $USER --start 0104 --end now --allocations
  seff 4379743 (replace with a valid JobID)
- manual monitoring; **squeue -u $USER** ; then SSH to a node and run top or something (only possible for the regular partitions, where jobs are running on the particular node).

University of Manitoba

Array jobs

SLURM has support for automatically running a job script on multiple data sets.

- You have regularly named, independent datasets (test1, test2, test3, …, test999) to process with a single software code
- Instead of making and submitting 999 job scripts, a single script can be used with the **--array=1-999** option to **sbatch**
- Within the job script, $SLURM_ARRAY_TASK_ID can be used to pick an array element to process
  - ./my_code test${SLURM_ARRAY_TASK_ID}
- When submitted, once, the script will create 999 jobs with the index added to JobID (12345_1, 12345_2, … , 12345_999)
- You can use usual SLURM commands (scancel, scontrol, squeue) on either entire array or on its individual elements

University of Manitoba

Job dependencies

SLURM provides job dependencies to build pipelines when subsequent jobs run depending on the result of the previous jobs

- A job or jobs is submitted first, their JobIDs are known
- Then dependent jobs can be submitted after as follows:

--dependency=afterok:jobid1
--dependency=afterallok:jobid1:jobid2:jobid3:jobid4

University of Manitoba

Further reading

SLURM website:
https://slurm.schedmd.com/documentation.html

ComputeCanada documentation on running jobs:
https://docs.computecanada.ca/wiki/Running_jobs

Grex documentation on running jobs:
https://monitor.hpc.umanitoba.ca/doc/docs/grex/running/

Westgrid training materials:
https://westgrid.github.io/trainingMaterials/