Batch System/Job Submission

Dr. Dimitris Dellis

GRNET

Athens, 11 Dec. 2018

Resources Manager - Batch System

- Typical Situation in a small research group :
  - Weekend is coming, one job is running in whole cluster. It is expected to finish sometime at Sun. morning.
  - Let's start another run for weekend. It will run for $\sim$ 24 hours sharing the cpu-memory-I/O of nodes, with all consequences, like swap memory etc. In general reduced performance for both jobs.
  - Afrer sometime at Sun, system will run nothing.
  - User X runs many jobs, let's start few more to get part of system power at the same time, whatever it means for performance.
  - Vacation period for 15 days, let's start all the jobs that I expect that will finish in 15 days.
  - (Probably, other users thought the same)
  - How many of you (who do not use batch system/resource manager), are familiar with such situations ?

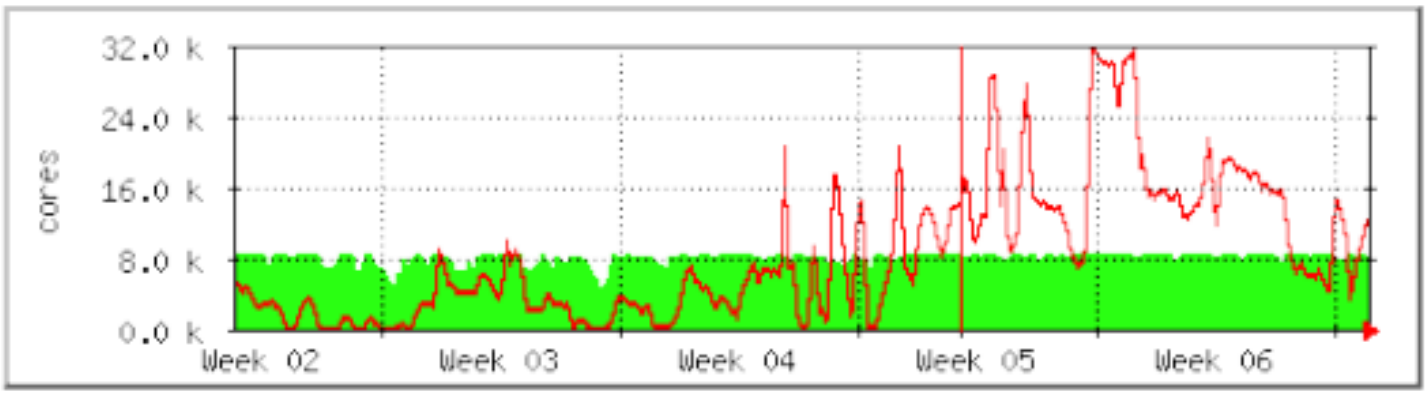
Resources Manager - Batch System

- What is Batch System/Resources Manager
  - A Batch system/Resource Manager controls the full access to the available resources in order all users to have access to resources.
  - Typically, the need for resources is higher than available resources, at least for large periods.
  - Resource Managers allow users to specify the resources for a job, allow them to submit the job, allows user to disconnect from system, and when the requested resources, number of cores, memory etc. are available, it runs the job.
- ARIS Batch System : SLURM, with accounting, fairshare, budgeting etc. features, with PBS emulation

Resources Manager - Batch System

- How I am affected ?

`Monthly' Graph (2 Hour Average)



	Max	Average	Current
Cores used by RUNNING jobs	8520 Cores	7968 Cores	8236 Cores
Cores requested by QUEUED jobs	31 kCores	8146 Cores	12 kCores

When a job is submitted to a Batch system : I

- It describes the requested resoures, cores, nodes, memory, execution time etc.
- System collects the requirements, and put the job in a Queue
- When the requested resources are available, it starts the execution
- Guarantee that each job will have full access to the requested resources. i.e. cores, memory, accelerators etc.
- Guarantee that if I submit say 1000 jobs, each of them will have full access to the requested resources.

When a job is submitted to a Batch system : II

- If another user submit jobs, they will be executed without concurrent use of resources, taking into account fair sharing.
- Allocated resources can be used in many ways, it is described in SLURM script.
  - One MPI job (that is the main/suggested use)
  - Many concurrent serial jobs. Although this is possible, it is not the suggested usage. Typically, threre is no gain from special hardware available on HPC systems, like Infiniband.
  - Probably a Grid/Cloud infrastructure fits better to this type of workflow.

Running Applications

- Running on login nodes is not allowed, although someone can run a few minutes check with small number of cores.
- In fact the current limits are 8 CPU core Hours, to allow procs like tar/zip, compilation, filetransfers etc. It will be lower in near future.
- You should use Resource Manager/Batch system to submit a job to compute nodes.
- Batch system on ARIS is SLURM.
- How to use it ? See http://doc.aris.grnet.gr/
- There is a script generator validator that is a good starting point to create a SLURM script.
- What is the content of this script ?
- You define the resources you need for your job and how to run.

Workload manager/Batch system : SLURM

```
#!/bin/bash
#SBATCH --job-name="testSlurm" # JobName
#SBATCH --error=job.err.%j     # Filename : stderr
#SBATCH --output=job.out.%j    # Filename : stdout
                               # %j value of JobID
#SBATCH --nodes=2              # Number of nodes
#SBATCH --ntasks=4             # Number of (usually MPI) Tasks
#SBATCH --ntasks-per-node=2    # Number of Tasks / node
#SBATCH --cpus-per-task=10     # Number of Threads / MPI Task
#SBATCH --mem=56G              # Memory per node  #  One of these 2 specs
#SBATCH --mem-per-cpu=2800M    # Memory per core  #
#SBATCH -A ptc                 # Accounting tag # ptc for training
#SBATCH -t 1-01:00:00          # Requsted  DD-HH:MM:SS
#SBATCH -p compute             # partition, one of compute, gpu, phi, fat, taskp, short
#SBATCH --gres=gpu:2           # Accelerated partitions. gres=gpu or mic
module purge
module load gnu/4.9.2
module load intel/15.0.3
module load intelmpi/5.0.3
if [ x$SLURM_CPUS_PER_TASK == x ]; then            #
  export OMP_NUM_THREADS=1                         #
else                                               # Never delete these lines
  export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK      # unless you exactly know what you do
fi                                                 #
srun EXECUTABLE ARGUMENTS   # Executable and possible arguments
```

SLURM Scripts I

- Previous slide SLURM Script is a complete job description.
- A script may contain less directives :
  - Specifying only −−nodes without −−ntasks system can calculate how many tasks to use
  - Specifying only −−ntasks system can calculate how many nodes are required/
  - Mandatory specification is one of −−ntasks, −−nodes.
  - Ommiting −−job-name, sets output to slurm-JOBID.out/.err
  - Use of −−account (or -A), is mandatory.

SLURM Scripts II

- Setting all variables, you have fine control on how to handle the allocated resources. Sometimes, in routine jobs ommiting to change one variable while changing others, may have undesirable results.
- It’s up to your experience if ommiting variables may help you.

SLURM Scripts I

- See at System Documentation for more details and script generator/validator.
- http://doc.aris.grnet.gr/scripttemplate/

Use of **srun** to run jobs I

- MPI flavors have mpirun/mpiexec etc.
- It is strongly suggested to use srun instead of mpirun/mpiexec.
- Some reasons
  - srun spawns the processes to all nodes, has full control and status over them
  - In cases of application problems, for example exit of one of the tasks, srun will be informed about exit, while mpirun/mpiexec may not, depending on MPI calls of each process. This may lead to think that the application runs while it is really crashed.

Use of **srun** to run jobs II

- srun keeps account of many run parameters, like power consumption, memory, network and disk usage per process and on average, etc. Usefull to determine problems after job crash.
- It is the common way to start parallel processes for all MPI flavors in system.
- srun propagates all the environment variables accross processes, while mpirun/mpiexec may not, depending on the names of variables.

Communication with SLURM I

- Job Submission

```
sbatch SLURM_JobScript.sh
Submitted batch job 123456
```

- Job List

```
squeue
```

- Detailed job List output

```
squeue -o "%.8i %.9P %.10j %.10u %.8T %.5C
%.4D %.6m %.10l %.10M %.10L %.16R"
```

- Job Cancel

Communication with SLURM II

```
scancel JobID
```

- In some cases, executables use signal handlers and do not terminate when they receive TERM/CONT signal. It is necessary to issue KILL signal

```
scancel -s KILL JobID
```

- Estimation of start time of a queued job

```
squeue --start
```

- Information on the system resources status

```
sinfo
```

Communication with SLURM III

- Information on the resources status of a certain part of system (partition)

  `π.χ. sinfo -p gpu`

SLURM jobs dependency I

- When a job should start after the completion (or just start) of another job, you should include among others :

  ```
  #SBATCH --dependency=after:Job_ID
  ```

  or

  ```
  #SBATCH --dependency=afterok:Job_ID
  ```

- If a job should start after completion of another job with the same name, among others :

  ```
  #SBATCH --dependency=singleton
  ```

SLURM jobs dependency II

- If a job should start after a certain time (and at submission time the requested resources could be reserved at this time)
  - Start at next occurence of 16:00

    ```
    #SBATCH --begin=16:00
    ```
  - Start at certain date/time

    ```
    #SBATCH --begin=2016-10-26T14:32:00
    ```

SLURM job Status I

- When a job is in pending state, i.e. not running yet, in squeue you'll see in nodelist/REASON column the reason why it is not started up to now.
  - If REASON is Resources, it means that is the first eligible to run when resources will be available.
  - If REASON is Priority, it means that other jobs have higher priority and should start first.
  - If REASON is something like Assoc#########... it means that you asked for more than allowed resources. Message is self explanatory. If not, check the slurm status messages.

SLURM job Status II

- Priority is dynamic, i.e. it depends on the queued jobs and their priority. It is not to see varying execution start time when new jobs are involved in queued jobs.

- `AssocMaxNodesPerJobLimit`
  We ask for more node than those allowed to our account.
- `AssocMaxWallDur`
  We ask for job Wall Time more than what is allower to our account
- If reason is not self explanatory, check the slurm status messages.

SLURM Environment Variables I

When a SLURM job starts, few environment variables are set. Commonly, thery may be user by executable(s).

```
$SLURM_NNODES                 # Number of nodes availble to Job
$SLURM_NTASKS                 # Number of Tasks
$SLURM_NPROCS                 #  ”  ”    ”
$SLURM_NTASKS_PER_NODE        # Number of Tasks per Node
$SLURM_TASKS_PER_NODE         #  ”   ”    ”    ”
$SLURM_CPUS_PER_TASK          # Number of threads / Task
$SLURM_MEM_PER_NODE           # Memory per node (MB)
```

- May be used as runtime arguments to instruct executable what/how resources to use.

SLURM Environment Variables II

- The parameter used by many applications, that need a specification of how many threads/tasks to use, is `$SLURM_CPUS_PER_TASK`. `OMP_NUM_THREADS` is set to this value (See full SLURM Script)

SLURM User/Group resource limits I

- Each account has certain resource limits.
  - Maximum number of running Jobs, Jobs in queue.
  - Maximum number of concurrently used cores and/or nodes
  - Maximum Wall time duration of a job
  - Maximum consumable Core Hours for project duration (=Budget).

SLURM Scheduler I

- ARIS Jobs Schedulre is : FIFO, with backfill and Fair Sharing. Thins means :
  - The Job that submitted first will be executed first
  - Once Job execution starts, the job will finish at latest after the requested wall time in script (+∼30 sec for gracefull exit).
  - If there are available free resources but not enough to start the first queued job, it will look if another of the queued jobs whose requirements may fit in the available resources.
  - If in queue there is a lower priority job whose wall time requirement falls into the ”dead” time, it will start.
  - This way, the system maximizes its usage.

SLURM Scheduler II

- Fairshare

- Factors that affect the priority in Queue
  - Wait Time : Longer time in queue, gives extra bonus in priority.
  - Job Size (In # nodes) : Higher number of requested nodes result to higher priority. It helps to fill the system capacity,
  - Relative project type usage : i.e. 80% production, 10% preparatory etc.
- What will happen if a user send few 100s of jobs (high percentage of queued jobs) ?

SLURM Scheduler III

- Fairshare cares about this. Priority of jobs from one user or account that had increased usage during last week, get lower priority
- While an account reaches the limit of the allocated budget, its priority decreases.

PBS emulation

- On ARIS, there is available the PBS/Torque emulation : One can use PBS/Torque commands/sytax to submit jobs, in order to enable fast transition to users who are familiar with PBS/Toorque.
- .. For short term.
- PBS emulation can handle alsmost, but not, all capabilities of SLURM.
- Recently CINECA moved from PBS to SLURM....

PBS/SLURM Equivalence

SLURM

sbatch
squeue
scancel

```
#!/bin/sh
#SBATCH --mem-per-cpu=2G
#SBATCH -t 1:00:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=20
#SBATCH -A ptc
#SBATCH -p compute
....
```

PBS

qsub
qstat
qdel

```
#!/bin/sh
#PBS -l pvmem=2G
#PBS -l walltime=1:00:00
#PBS -l nodes=1:ppn=20

#PBS -A ptc
#PBS -q compute
....
```

Accounting I

- **sacct**
  See your jobs details during current day
- See your jobs details since specified date
  sacct -S 2018-01-01
- See your budget consumption per allocation
  myreport
- See usage report (allocated/used
  mybudget

Efficient use with Slurm I

- If your workflow requires some serial parts, typically :

```
./run_serial1
./run_serial2
srun parallel1
srun parallel2
```

- do not use srun (without arguments) with all steps, i.e

```
srun run_serial1
srun run_serial2
srun parallel1
srun parallel2
```

Efficient use with Slurm II

- The behavior of the last is unpredictabpe, it depends on software.
  - You may run multiple times the same execurable. It depends on application.
  - All serial runs write the same file(s) that may contain ”trashes” after execution, or
  - Serial runs care about exising files, and you’ll find many versions of the same output file.
  - Use instead

Efficient use with Slurm III

```
srun -n 1 -N 1 run_serial1
srun -n 1 -N 1 run_serial2
srun parallel1
srun parallel2
```

Efficient use

- ARIS compute nodes have 20 or 40 cores. Use if possible full nodes, i.e. 20/40 cores/node.
- If it is not the case, limit the required nodes. Examples :

cores	Nodes	tasks/node	Unused cores
64	4	20	16 on 1 node
128	7	20	12 on 1 node
256	13	20	4 on 1 node
512	26	20	8 on 1 node

- Common mistake

cores	Nodes	tasks/node	Unused cores
64	8	8	12 cores/node on 8 nodes=96
64	4	16	4 cores/node on 4 nodes = 16
90	6	15	5 cores/node on 6 nodes = 30
128	8	16	4 cores/node on 8 nodes = 32
480	40	12	8 cores/node on 40 nodes = 320
512	32	16	4 cores/node on 32 nodes = 128

Efficient use

- Do not use mpirun/mpiexec nor typical desktop arguments like -np. It happens to forget to change the really needed resources, for example :

```
#SBATCH --nodes=10
#SBATCH --ntasks=200
mpirun -np 8
or
srun -n 8
```

  You allocate (and charged for) 200 cores while you use only 8.
- Try to use the correct combination of tasks and threads with Hybrid applications. Check that the `OMP_NUM_THREADS` is set. In SLURM script template there is code that checks for this.
- Surprisingly, this piece of code is the most frequently removed.

Efficient use I

- If you can use save/restart and need very long time, use it. Instead of a job of 10 days, use 10 jobs of 1 day (propability of a HW failure in 10 days much higher - especially with multinode runs).
- Request from the Resource Manager wall time slightly higher than the expected. NOT the typical 2 days.

Efficient use II

- Example : Submit 100 jobs requesting 2 days each. Scheduler will arrange to run them in $\sim$ 1 week, if there are available resources. If each run takes 5 minutes, requesting 6 minutes, all runs will finish in $\sim$ 1 hour instead of $\sim$ 1 week.
- Even better, submit few jobs with multiple srun, for example 10 jobs with 10 srun.
- Stats : Sept. 2017
  65% of Completed jobs took up to 5% of requested time
  9% between 5 and 10%.
  11% more than 50%

Efficient use III

- Stats : Jan. 2018
  49% of Completed jobs took up to 5% of requested time
  9% between 5 and 10%.
  20.5% more than 50% (Improvement after trainings)

Questions ?