CNAG Advanced User Training

Aníbal Moreno, CNAG System Administrator
Pablo Ródenas, BSC HPC Support
Rubén Ramos Horta, CNAG HPC Support

Barcelona, May the 5th
Aim

Understand CNAG’s cluster design
- HPC Cluster: Big and complex machine unlike your desktop

Understand Slurm Scheduling
- Knowledge of Slurm’s internals
- Slurm tips and tweaks

Understand Lustre internals
- Complex network underneath

Who to contact?
- www.bsc.es/user-support/cnag.php
- cnag_support@bsc.es
Components

The cluster consists of:
- 2 login nodes
- 108 GenC nodes (24 cores, 256 GB RAM)
- 12 GPU nodes (16 cores, 128 GB RAM)
- 20 GenB nodes (16 cores, 128 GB RAM)
- 94 GenA nodes (8 cores, 48 GB RAM)
- 1 SMP node (48 cores, 1TB RAM)

Data is stored in:
- 1 NFS exported /home on all nodes
- 1 NFS exported /apps on all nodes
- 2 High Performance Distributed Filesystems (Lustre) (/project /scratch)

Jobs are executed with:
- Slurm Resource Manager (Batch system)
Batch System (Slurm)
Slurm concepts

Scheduling

- FIFO with priorities
- Backfilling
  - Schedules less priority jobs but only those which meet free resources conditions.
  - Increases utilization of the cluster
  - Requires declaration of max execution time of jobs
    - \( \text{--time} \) on \text{srun}
    - \text{DefaultTime} or \text{MaxTime} on Partition
Backfill

FIFO Scheduler

Backfill Scheduler
Backfilling in the cluster

$ sdiag
[..-]

Main schedule statistics (microseconds):
  Last cycle: 9773
  Max cycle: 89151
  Total cycles: 1081
  Mean cycle: 6197
  Mean depth cycle: 366
  Cycles per minute: 1
  Last queue length: 423

Backfilling stats
  Total backfilled jobs (since last slurm start): 1388236
  Total backfilled jobs (since last stats cycle start): 1350
  Total cycles: 184
  Last cycle when: Thu Apr 27 14:19:05 2017
  Last cycle: 14770246
  Max cycle: 24826346
  Mean cycle: 10639957
  Last depth cycle: 416
  Last depth cycle (try sched): 416
  Depth Mean: 1014
  Depth Mean (try depth): 584
  Last queue length: 416
  Queue length mean: 1014
Priorities

Determining factors

- **Age** → the length of time a job has been waiting in the queue, eligible to be scheduled. This factor will reach its highest value after 7 days

- **QoS** → a factor specified to a specific QoS

- **Fair-share** → the difference between the portion of the computing resource that has been promised and the amount of resources that has been consumed. This factor will consider only the last 14 days

- **Job size** → the number of nodes or CPUs a job is allocated

- **Partition** → a factor associated with each node partition

\[
\text{Job\_priority} = \\
\quad (\text{PriorityWeightAge}) \times (\text{age\_factor}) + \\
\quad (\text{PriorityWeightFairshare}) \times (\text{fair\_share\_factor}) + \\
\quad (\text{PriorityWeightJobSize}) \times (\text{job\_size\_factor}) + \\
\quad (\text{PriorityWeightPartition}) \times (\text{partition\_factor}) + \\
\quad (\text{PriorityWeightQOS}) \times (\text{QOS\_factor}) + \\
\quad \text{SUM}(\text{TRES\_weight\_cpu} \times \text{TRES\_factor\_cpu}, \\
\quad \text{TRES\_weight\_<type>} \times \text{TRES\_factor\_<type>}, \\
\quad \ldots)
\]
Priorities

Check the cluster weights for each factor
$ sprio --weights

```
JOBID   PRIORITY        AGE  FAIRSHARE    JOBSIZE  PARTITION        QOS
Weights
   5000000   10000000   10000   20000000   100000000
```

Check your job's priority that is still in the queue waiting to be scheduled

$ sprio -j 6486323

```
JOBID   PRIORITY        AGE  FAIRSHARE    JOBSIZE  PARTITION        QOS
86323   20538776       2290     436465         22   20000000     100000
```

Final priority calculated with normalized jobs priorities

$ sprio -j 6486327 -n

```
JOBID   PRIORITY        AGE  FAIRSHARE    JOBSIZE  PARTITION        QOS
6486327 0.00478336 0.0014501  0.0437138  0.0022012  1.0000000  0.0010000
```

\[\text{PRIORITY} = 0.0014501 \times \text{AGE\_weight} + 0.0437138 \times \text{FAIRSHARE\_weight} + 0.0022012 \times \text{JOBSIZE\_weight} + 1.0000000 \times \text{PARTITION\_weight} + 0.0010000 \times \text{QOS\_weight} \]

\[= 20538776\]
### Partitions

*alloc → node allocated to one or more jobs*
*mix → node allocated but still with free resources*
*idle → node completely free*
*drain → node not available*
*resv → special reservations*

```
$ sinfo
PARTITION   AVAIL  TIMELIMIT  NODES  STATE NODELIST
main*        up     infinite  18    mix     cnc[2,14,36-37,46-59]
main*        up     infinite  13    alloc   cnc[1,3-4,6,38-45,68]
main*        up     infinite  77    idle    cnc[5,15-35,60-67,69-108]
gpu          up     infinite  11    resv    cga[2-12]
genA         up     infinite  17    drain*  cna[21,27-28...67-68]
genA         up     infinite  2     mix     cna[24,49]
genA         up     infinite  6     alloc   cna[26,39,43,50-51,53]
genA         up     infinite  51    idle    cna[11-20,22-23,...,44-46,88]
genB         up     infinite  4     mix     cnb[8,10,12-13]
genB         up     infinite  16    idle    cnb[1-7,9,11,14-20]
interactive  up     infinite  1     resv    cga1
interactive  up     infinite  2     mix     cna[1,8]
interactive  up     infinite  3     alloc   cna[5-7]
interactive  up     infinite  5     idle    cna[2-4,9-10]
smp          up     infinite  1     alloc   smp1
```
Accounting

Limit Enforcement → Accounting of resources usage by users for controlling access to resources

- Fair share
- Maximum limits
  - MaxCPUsPerJob, MaxJobs, MaxNodesPerJob, MaxSubmitJobs, MaxWallDurationPerJob
- Quality of Service (QoS) ($ sacctmgr show qos)
$ ssh -X login1
$ sview
Job arrays (I)

- Useful when a set of jobs share logic but differ in some parameter or input
- Modifying slightly your jobs, instead of launching N jobs, just need to launch a single one
- Job script directive: `#@ array=1-N`
Environment variables for handling your job arrays

- \texttt{SLURM\_JOB\_ID}
- \texttt{SLURM\_ARRAY\_JOB\_ID}
- \texttt{SLURM\_ARRAY\_TASK\_ID}
- \texttt{SLURM\_ARRAY\_TASK\_COUNT}
- \texttt{SLURM\_ARRAY\_TASK\_MAX}
- \texttt{SLURM\_ARRAY\_TASK\_MIN}

```bash
#!/bin/bash
#SBATCH --job-name=arrayJob
#SBATCH --output=arrayJob_%A_%a.out
#SBATCH --error=arrayJob_%A_%a.err
#SBATCH --array=1-1024
#SBATCH --time=01:00:00
#SBATCH --partition=main
#SBATCH --ntasks=1
#SBATCH --mem-per-cpu=4000

# Print this sub-job's task ID
echo "My ARRAY\_TASK\_ID: " $SLURM\_ARRAY\_TASK\_ID

# Do some work based on the SLURM\_ARRAY\_TASK\_ID
./my_program $SLURM\_ARRAY\_TASK\_ID
```
Job arrays (II)

Job1

SLURM_JOB_ID=12780
SLURM_ARRAY_JOB_ID=6570001
SLURM_ARRAY_TASK_ID=1
SLURM_ARRAY_TASK_COUNT=1024
SLURM_ARRAY_TASK_MAX=1024
SLURM_ARRAY_TASK_MIN=1

...
Job arrays (III)

```
#SBATCH -array=0-1024
  • Create a job array

#SBATCH -array=0-1024%10
  • Only execute 10 jobs simultaneously from this job array

$ squeue

<table>
<thead>
<tr>
<th>JOBID</th>
<th>PARTITION</th>
<th>NAME</th>
<th>USER</th>
<th>ST</th>
<th>TIME</th>
<th>NODES</th>
<th>NODELIST(REASON)</th>
</tr>
</thead>
<tbody>
<tr>
<td>12780_[5-1024]</td>
<td>debug</td>
<td>tmp</td>
<td>mac</td>
<td>PD</td>
<td>0:00</td>
<td>1</td>
<td>(Resources)</td>
</tr>
<tr>
<td>12780_1</td>
<td>debug</td>
<td>tmp</td>
<td>mac</td>
<td>R</td>
<td>0:17</td>
<td>1</td>
<td>tux0</td>
</tr>
<tr>
<td>12780_2</td>
<td>debug</td>
<td>tmp</td>
<td>mac</td>
<td>R</td>
<td>0:16</td>
<td>1</td>
<td>tux1</td>
</tr>
<tr>
<td>12780_3</td>
<td>debug</td>
<td>tmp</td>
<td>mac</td>
<td>R</td>
<td>0:03</td>
<td>1</td>
<td>tux2</td>
</tr>
<tr>
<td>12780_4</td>
<td>debug</td>
<td>tmp</td>
<td>mac</td>
<td>R</td>
<td>0:03</td>
<td>1</td>
<td>tux3</td>
</tr>
</tbody>
</table>

scancel 128080_[1-3]
  • Cancel a range of tasks from a job array
Modify your jobscript once launched and before running

- `scontrol update jobid=<jobid> <parameter>`
  - **Timelimit**
    - `scontrol update jobid=9450064 timelimit=9-00:00:00`
  - **Qos**
    - `scontrol update jobid=9450064 qos=highprio`
  - **Partition**
    - `scontrol update jobid=9450064 partition=main`

- Prevent a job from being started
  - `scontrol hold <jobid>`
  - `Scontrol uhold <jobid>`
What is Lustre

A High Performance Parallel Distributed Filesystem

- High Performance → Fast access
- Parallel → Many different processes in different computers can access the same file
- Distributed → Many different servers holding different parts of the same file, working together

Meaning: Formula 1, not a 4WD
Why Lustre?

- NFS (Network FileSystem): /home and /apps
  - Slow access
  - Not Parallel
  - Not distributed

- All users trying to access to resources at the same time → Bottleneck
Lustre components

 Metadata Servers (MDS): I/O service and network request for one or more local MDT
 Metadata Targets (MDT): where metadata is actually stored (RAID 1 of SSD disks)
 Object Storage Servers (OSS): I/O service and network request for one or more local OST
 Object Storage Targets (OST): where data is actually stored (RAID 5+1 SATA2 disk)
Lustre’s stripes

Diagram showing the layout of FID, Layout EA, and Data Stored on OSTs, illustrating how objects are stored and striped across different OSTs.
Lustre’s stripes
Lustre’s stripes

Get stripes from a specific folder

```
$ lfs getstripe pruebas
pruebas/
  stripe_count:  4 stripe_size:  1048576
  stripe_offset: -1
pruebas/2
  stripe_count:  4 stripe_size:  1048576
  stripe_offset: -1
pruebas/1
  stripe_count:  4 stripe_size:  1048576
  stripe_offset: -1
```

Set number of stripes for a specific folder

- A lot of small files → faster access with just one stripe
- Big files → faster and distributed with several stripes

```
$ mkdir pruebas/3
$ lfs setstripe -c 1 pruebas/3
$ lfs getstripe pruebas/3
pruebas/3
  stripe_count:  1 stripe_size:  1048576 stripe_offset: -1
```

Use carefully! Contact us in case of doubt
An easy way of creating dependencies between processes

- Each line is a task
- Lines starting with # are comments
- Line number = task id
- Each line [# optional list of dependencies #] <new_task>
- Only backwards dependencies are allowed

More info:

GREASY: Example

# this line is a comment
/bin/sleep 2

# following task id is 4. Tasks ids 1 and 3 don’t exist
/bin/sleep 4

# following task will run after completion of the “sleep 2”
[## 2 ##] /bin/sleep 6

# following task will be run after completion of the "sleep 6"
[## -2 ##] /bin/sleep 8

# following task is invalid because tasks 1 and 3 do not exist
[#1-3#] /bin/sleep 10

# following task will be run after completion of tasks 2, 4, 8
[##2, 4, -4 ##] /bin/sleep 12
Data Management
Data management tips

Avoid intermediate steps
  • Use bash pipelines
    – ./do_stuff | ./manage_stuff_1 | ... | ./manage_stuff_N > out

Avoid replicating data

Compress as much data as possible
  • Standard compression
    – tar -cvf name.tar.gz <path_to_file>
  • Parallel compression → ++faster (Not in login nodes!)
    – tar -cvf <path_to_file> | pigz -p N_PROC > name.tar.gz

No need to untar every time
  • zcat, zless, zgrep, zdiff
Improve performance with GPU
GPU nodes: Nvidia Tesla K80

GenC processors
- 1.92TFlops
- 68 GB/s of bandwidth

Improve your program performance with CUDA
- 2.91TFlops double precision performance
- 8.73TFlops single precision performance
- 480 GB/sec of bandwidth

Some developers have a version ready to run on CUDA. Ask us to install it

Ask for help if you have an idea for parallelizing your program
Major incidences
Running out of inodes

One user created 80 millions of files in scratch

- Non expected use of the cluster
- Hard to monitor everything
**Major incidences**

**Sending too many jobs → Use job arrays instead**
- Affects Slurm’s scheduling
- Cluster runs out of resources
  - Max 100000 jobs to be scheduled at the same time
  - Max 3000 max jobs to be backfilled at the same time
  - If limit is exceed, no other jobs will be scheduled until queue is emptied

**WallClock accuracy**
- Smarter use of resources
- Allows more jobs to be scheduled
Thank you!

For further information please contact cnag_support@bsc.es