UL HPC School 2017
PS5: Advanced Scheduling with SLURM and OAR on UL HPC clusters

UL High Performance Computing (HPC) Team
V. Plugaru

University of Luxembourg (UL), Luxembourg
http://hpc.uni.lu
Latest versions available on Github:

UL HPC tutorials:  
https://github.com/ULHPC/tutorials

UL HPC School:  
http://hpc.uni.lu/hpc-school/

PS5 tutorial sources:  
https://github.com/ULHPC/tutorials/tree/devel/advanced/advanced_scheduling
1 Introduction

2 SLURM workload manager
   SLURM concepts and design for iris
   Running jobs with SLURM

3 OAR and SLURM

4 Conclusion
Introduction

Main Objectives of this Session

- Design and usage of **SLURM**
  - cluster workload manager of the UL HPC **iris** cluster
  - ... and future HPC systems

The tutorial will show you:

- the way SLURM was **configured, accounting** and **permissions**
- **common** and **advanced** SLURM tools and commands
  - `srun`, `sbatch`, `squeue` etc.
  - job specification
  - SLURM job types
  - comparison of SLURM (**iris**) and OAR (**gaia & chaos**) 
- **SLURM** generic **launchers** you can use for your own jobs

Documentation & comparison to OAR

[https://hpc.uni.lu/users/docs/scheduler.html](https://hpc.uni.lu/users/docs/scheduler.html)
Summary

1 Introduction

2 SLURM workload manager
   SLURM concepts and design for iris
   Running jobs with SLURM

3 OAR and SLURM

4 Conclusion
SLURM workload manager

SLURM - core concepts

- SLURM manages user jobs with the following key characteristics:
  - set of requested resources:
    - ✓ number of computing resources: nodes (including all their CPUs and cores) or CPUs (including all their cores) or cores
    - ✓ amount of memory: either per node or per (logical) CPU
    - ✓ (wall)time needed for the user’s tasks to complete their work
  - a requested node partition (job queue)
  - a requested quality of service (QoS) level which grants users specific accesses
  - a requested account for accounting purposes

- **Example**: run an interactive job

  **Alias**: `si [...]

  ```bash
  (access)$ srung -p interactive --qos qos --interactive --pty bash
  (node)$ echo $SLURM_JOBID
  2058
  ```

  Simple interactive job running under SLURM
**Simple interactive job running under SLURM**
Many metrics available during and after job execution

- including energy (J) – but with caveats
- job steps counted individually
- enabling advanced application debugging and optimization

Job information available in easily parseable format (add -p/-P)

```
$ sacct -j 2058 --format=account,user,jobid,jobname,partition,state
  Account  User  JobID  JobName  Partition  State
  ulhpc    vplugaru 2058  bash interacti +  COMPLETED

5 $ sacct -j 2058 --format=elapsed,elapsedraw,start,end
  Elapsed  ElapsedRaw  Start  End
  00:02:56  176 2017-06-09T16:49:42 2017-06-09T16:52:38

$ sacct -j 2058 --format=maxrss,maxvmsize,consumedenergy,consumedenergyraw,nnodes,ncpus,nodelist
  MaxRSS  MaxVMSIZE ConsumedEnergy ConsumedEnergyRaw NNodes  NCPUS  NodeList
  0 299660K  17.89K  17885.000000 1 1  iris -081
```

Job metrics after execution ended
### SLURM - design for iris (I)

<table>
<thead>
<tr>
<th>Partition</th>
<th># Nodes</th>
<th>Default time</th>
<th>Max time</th>
<th>Max nodes/user</th>
</tr>
</thead>
<tbody>
<tr>
<td>batch*</td>
<td>88 (82%)</td>
<td>0-2:0:0</td>
<td>5-0:0:0</td>
<td>unlimited</td>
</tr>
<tr>
<td>interactive</td>
<td>10 (9%)</td>
<td>0-1:0:0</td>
<td>0-4:0:0</td>
<td>2</td>
</tr>
<tr>
<td>long</td>
<td>10 (9%)</td>
<td>0-2:0:0</td>
<td>30-0:0:0</td>
<td>2</td>
</tr>
</tbody>
</table>
### SLURM - design for iris (I)

<table>
<thead>
<tr>
<th>Partition</th>
<th># Nodes</th>
<th>Default time</th>
<th>Max time</th>
<th>Max nodes/user</th>
</tr>
</thead>
<tbody>
<tr>
<td>batch*</td>
<td>88 (82%)</td>
<td>0-2:0:0</td>
<td>5-0:0:0</td>
<td>unlimited</td>
</tr>
<tr>
<td>interactive</td>
<td>10 (9%)</td>
<td>0-1:0:0</td>
<td>0-4:0:0</td>
<td>2</td>
</tr>
<tr>
<td>long</td>
<td>10 (9%)</td>
<td>0-2:0:0</td>
<td>30-0:0:0</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>QoS</th>
<th>User group</th>
<th>Max cores</th>
<th>Max jobs/user</th>
</tr>
</thead>
<tbody>
<tr>
<td>qos-besteffort</td>
<td>ALL</td>
<td>no limit</td>
<td></td>
</tr>
<tr>
<td>qos-batch</td>
<td>ALL</td>
<td>1064</td>
<td>100</td>
</tr>
<tr>
<td>qos-interactive</td>
<td>ALL</td>
<td>224</td>
<td>10</td>
</tr>
<tr>
<td>qos-long</td>
<td>ALL</td>
<td>224</td>
<td>10</td>
</tr>
<tr>
<td>qos-batch-001</td>
<td>private</td>
<td>1400</td>
<td>100</td>
</tr>
<tr>
<td>qos-interactive-001</td>
<td>private</td>
<td>56</td>
<td>10</td>
</tr>
<tr>
<td>qos-long-001</td>
<td>private</td>
<td>56</td>
<td>10</td>
</tr>
</tbody>
</table>
Default partition: **batch**, meant to receive most user jobs

- we hope to see majority of user jobs being able to scale

All partitions have a correspondingly named **QOS**

- granting resource access (**long** – **qos-long**)
- any job is tied to one QOS (user specified or inferred)
- automation in place to select QOS based on partition
SLURM workload manager

SLURM - design for iris (II)

- **Default partition**: **batch**, meant to receive most user jobs
  - we hope to see majority of user jobs being able to scale
- All partitions have a correspondingly named **QOS**
  - granting resource access (**long** – **qos-long**)
  - any job is tied to one QOS (user specified or inferred)
  - automation in place to select QOS based on partition

- **Preemptible besteffort** QOS available for **batch** and **interactive** partitions (but not for **long**)
  - meant to ensure maximum resource utilization
  - should be used together with checkpointable software
- **QOSs** specific to particular group accounts exist (discussed later)
  - granting additional accesses to platform contributiors
**Backfill** scheduling for efficiency

- **multifactor job priority** (size, age, fairshare, QOS, ...)  
- currently weights set for: job age, partition and fair-share  
- other factors/decay to be tuned **after observation** period  
  ✓ with more user jobs in the queues

**Resource selection:** **consumable resources**

- **cores and memory** as consumable (per-core scheduling)  
- block distribution for cores (best-fit algorithm)  
- default memory/core: 4GB (4.1GB maximum, rest is for OS)
SLURM workload manager

SLURM - design for iris (III)

- **Backfill** scheduling for efficiency
  - *multifactor job priority* (size, age, fairshare, QOS, ...)
  - currently weights set for: job age, partition and fair-share
  - other factors/decay to be tuned *after observation* period
    - ✓ with more user jobs in the queues

- **Resource selection**: *consumable resources*
  - *cores and memory* as consumable (per-core scheduling)
  - block distribution for cores (best-fit algorithm)
  - default memory/core: 4GB (4.1GB maximum, rest is for OS)

- **Reliable user process tracking with cgroups**
  - cpusets used to constrain cores and RAM (no swap allowed)
  - task affinity used to bind tasks to cores (hwloc based)

- **Hierarchical tree topology defined** (for the network)
  - for optimized job resource allocation
SLURM workloads manager

SLURM - design for iris (III)

- **Backfill** scheduling for efficiency
  - multifactor job priority (size, age, fairshare, QOS, etc.)
  - currently weights set for: job age, partition, and fairshare
  - other factors/decay to be tuned after observation period
  - ✓ with more user jobs in the queues

- Resource selection: **consumable resources**
  - cores and memory as consumable (per-core scheduling)
  - block distribution for cores (best-fit algorithm)
  - default memory/core: 4GB (4.1GB maximum, rest is for OS)

- Reliable user process tracking with **cgroups**
  - cpusets used to constrain cores and RAM (no swap allowed)
  - task affinity used to bind tasks to cores (hwloc based)

- Hierarchical tree topology defined (for the network)
  - for optimized job resource allocation

Help will be needed on your part to optimize your job parameters!
A note on job priority

Job_priority =

(\text{PriorityWeightAge}) \times (\text{age_factor}) +
(\text{PriorityWeightFairshare}) \times (\text{fair-share_factor}) +
(\text{PriorityWeightJobSize}) \times (\text{job_size_factor}) +
(\text{PriorityWeightPartition}) \times (\text{partition_factor}) +
(\text{PriorityWeightQOS}) \times (\text{QOS_factor}) +
\text{SUM}(\text{TRES_weight_cpu} \times \text{TRES_factor_cpu},
\text{TRES_weight_<type>} \times \text{TRES_factor_<type>},
...)

- TRES - Trackable RESources
  - CPU, Energy, Memory and Node tracked by default. All details at slurm.schedmd.com/priority_multifactor.html

- The corresponding weights and reset periods we need to tune
  - We require (your!) real application usage to optimize them
Some details on job permissions...

- Partition limits + association-based rule enforcement
  - association settings in SLURM’s accounting database
- QOS limits imposed, e.g. you will see (QOSGrpCpuLimit)
- Only users with existing associations able to run jobs
Some details on job permissions...

- Partition limits + association-based rule enforcement
  - association settings in SLURM’s accounting database
- QOS limits imposed, e.g. you will see (QOSGrpCpuLimit)
- Only users with existing associations able to run jobs

- Best-effort jobs possible through preemptible QOS: qos-besteffort
  - of lower priority and preemptible by all other QOS
  - preemption mode is requeue, requeueing enabled by default
Some details on job permissions...

- Partition limits + association-based rule enforcement
  - association settings in SLURM’s accounting database
- QOS limits imposed, e.g. you will see (QOSGrpCpuLimit)
- Only users with existing associations able to run jobs

Best-effort jobs possible through preemptible QOS: qos-besteffort
  - of lower priority and preemptible by all other QOS
  - preemption mode is requeue, requeueing enabled by default

On metrics: Accounting & profiling data for jobs sampled every 30s
  - tracked: cpu, mem, energy
  - energy data retrieved through the RAPL mechanism
  - caveat: for energy not all hw. that may consume power is monitored with RAPL (CPUs, GPUs and DRAM are included)
On tightly coupled parallel jobs (MPI)

- Process Management Interface (PMI 2) highly recommended
- PMI2 used for better scalability and performance
  - faster application launches
  - tight integration w. SLURM’s job steps mechanism (& metrics)
  - we are also testing PMIx (PMI Exascale) support
On tightly coupled parallel jobs (MPI)

- Process Management Interface (PMI 2) highly recommended
- PMI2 used for better scalability and performance
  - faster application launches
  - tight integration w. SLURM’s job steps mechanism (& metrics)
  - we are also testing PMIx (PMI Exascale) support
- PMI2 enabled in default software set for IntelMPI and OpenMPI
  - requires minimal adaptation in your workflows
  - replace mpirun with SLURM’s srun (at minimum)
  - if you compile/install your own MPI you’ll need to configure it

Example: https://hpc.uni.lu/users/docs/slurm_launchers.html
On tightly coupled parallel jobs (MPI)

- Process Management Interface (PMI 2) highly recommended
- PMI2 used for better scalability and performance
  - faster application launches
  - tight integration w. SLURM’s job steps mechanism (& metrics)
  - we are also testing PMIx (PMI Exascale) support
- PMI2 enabled in default software set for IntelMPI and OpenMPI
  - requires minimal adaptation in your workflows
  - replace mpirun with SLURM’s srun (at minimum)
  - if you compile/install your own MPI you’ll need to configure it

Example: https://hpc.uni.lu/users/docs/slurm_launchers.html

SSH-based connections between computing nodes still possible

- other MPI implementations can still use ssh as launcher
  - but really shouldn’t need to, PMI2 support is everywhere
- user jobs are tracked, no job == no access to node
SLURM workload manager

SLURM - design for iris (VI)

ULHPC customizations through plugins

- Job submission rule / filter
  - for now: QOS initialization (if needed)
  - more rules to come (group credits, node checks, etc.)

- Per-job temporary directories creation & cleanup
  - better security and privacy, using kernel namespaces and binding
  - /tmp & /var/tmp are /tmp/$jobid.$rscnt/[tmp, var_tmp]
  - transparent for apps. ran through srun
  - apps. ran with ssh cannot be attached, will see base /tmp!

- X11 forwarding (GUI applications)
  - enabled with --x11 parameter to srun/salloc
  - currently being rewritten to play nice with per-job tmpdir
    - workaround: create job and ssh -X to head node (need to propagate job environment)
Software licenses in SLURM

- Allinea Forge and Performance Reports for now
  - static allocation in SLURM configuration
  - dynamic checks for FlexNet / RLM based apps. coming later
- Number and utilization state can be checked with:
  - `scontrol show licenses`
- Use not enforced, **honor system** applied
  - `srun [...] -L $licname:$licnumber`

```bash
$> srun -N 1 -n 28 -p interactive -L forge:28 --pty bash -i
```
Hierarchical bank (group) accounts

UL as root account, then underneath accounts for the 3 Faculties and 3 ICs

All Prof., Group leaders and above have bank accounts, linked to a Faculty or IC

→ with their own name: Name.Surname

All user accounts linked to a bank account

→ including Profs.’s own user

Iris accounting DB contains over

→ 75 group accounts from all Faculties/ICs

→ comprising 477 users

Allows better usage tracking and reporting than was possible before.
SLURM workload manager

SLURM - brief commands overview

- **squeue**: view queued jobs
- **sinfo**: view partition and node info.
- **sbatch**: submit job for batch (scripted) execution
- **srun**: submit interactive job, run (parallel) job step
- **scancel**: cancel queued jobs
SLURM - brief commands overview

- **squeue**: view queued jobs
- **sinfo**: view partition and node info.
- **sbatch**: submit job for batch (scripted) execution
- **srun**: submit interactive job, run (parallel) job step
- **scancel**: cancel queued jobs

- **scontrol**: detailed control and info. on jobs, queues, partitions
- **ssstat**: view system-level utilization (memory, I/O, energy)
  - for running jobs / job steps
- **sacct**: view system-level utilization
  - for completed jobs / job steps (accounting DB)
- **sacctmgr**: view and manage SLURM accounting data
squeue: view queued jobs
sinfo: view partition and node info.
sbatch: submit job for batch (scripted) execution
srun: submit interactive job, run (parallel) job step
scancel: cancel queued jobs

scontrol: detailed control and info. on jobs, queues, partitions
sstat: view system-level utilization (memory, I/O, energy)
   for running jobs / job steps
sacct: view system-level utilization
   for completed jobs / job steps (accounting DB)
sacctmgr: view and manage SLURM accounting data

sprio: view job priority factors
sshare: view accounting share info. (usage, fair-share, etc.)
### SLURM - basic commands

<table>
<thead>
<tr>
<th>Action</th>
<th>SLURM command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Submit passive/batch job</td>
<td>sbatch $script</td>
</tr>
<tr>
<td>Start interactive job</td>
<td>srun --pty bash -i</td>
</tr>
<tr>
<td>Queue status</td>
<td>squeue</td>
</tr>
<tr>
<td>User job status</td>
<td>squeue -u $user</td>
</tr>
<tr>
<td>Specific job status (detailed)</td>
<td>scontrol show job $jobid</td>
</tr>
<tr>
<td>Job metrics (detailed)</td>
<td>sstat --job $jobid -l</td>
</tr>
<tr>
<td>Job accounting status (detailed)</td>
<td>sacct --job $jobid -l</td>
</tr>
<tr>
<td>Delete (running/waiting) job</td>
<td>scancel $jobid</td>
</tr>
<tr>
<td>Hold job</td>
<td>scontrol hold $jobid</td>
</tr>
<tr>
<td>Resume held job</td>
<td>scontrol release $jobid</td>
</tr>
<tr>
<td>Node list and their properties</td>
<td>scontrol show nodes</td>
</tr>
<tr>
<td>Partition list, status and limits</td>
<td>sinfo</td>
</tr>
</tbody>
</table>

**QoS deduced if not specified, partition needs to be set if not "batch"**
### SLURM - basic options for sbatch/srun

<table>
<thead>
<tr>
<th>Action</th>
<th>sbatch/srun option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request (n) distributed nodes</td>
<td>(-N) (n)</td>
</tr>
<tr>
<td>Request (m) memory per node</td>
<td>(--mem=) (m)GB</td>
</tr>
<tr>
<td>Request (mc) memory per core (logical cpu)</td>
<td>(--mem-per-cpu=) (mc)GB</td>
</tr>
<tr>
<td>Request job walltime</td>
<td>(--time=d-hh:mm:ss)</td>
</tr>
<tr>
<td>Request (nt) tasks per node</td>
<td>(--ntasks-per-node=) (nt)</td>
</tr>
<tr>
<td>Request (ct) cores per task (multithreading)</td>
<td>(-c) (ct)</td>
</tr>
<tr>
<td>Request (nt) total # of tasks</td>
<td>(-n) (nt)</td>
</tr>
<tr>
<td>Request to start job at specific (time)</td>
<td>(--begin) (time)</td>
</tr>
<tr>
<td>Specify job name as (name)</td>
<td>(-J) (name)</td>
</tr>
<tr>
<td>Specify job partition</td>
<td>(-p) (partition)</td>
</tr>
<tr>
<td>Specify QOS</td>
<td>(--qos) (qos)</td>
</tr>
<tr>
<td>Specify account</td>
<td>(-A) (account)</td>
</tr>
<tr>
<td>Specify email address</td>
<td>(--mail-user=) (email)</td>
</tr>
<tr>
<td>Request email on event</td>
<td>(--mail-type=all[,begin,end,fail])</td>
</tr>
<tr>
<td>Use the above actions in a batch script</td>
<td>#SBATCH \ $option</td>
</tr>
</tbody>
</table>
### SLURM - basic options for sbatch/srun

<table>
<thead>
<tr>
<th>Action</th>
<th>sbatch/srun option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request $n distributed nodes</td>
<td>-N $n</td>
</tr>
<tr>
<td>Request $m memory per node</td>
<td>--mem=$mGB</td>
</tr>
<tr>
<td>Request $mc memory per core (logical cpu)</td>
<td>--mem-per-cpu=$mcGB</td>
</tr>
<tr>
<td>Request job walltime</td>
<td>--time=d-hh:mm:ss</td>
</tr>
<tr>
<td>Request $tn tasks per node</td>
<td>--ntasks-per-node=$tn</td>
</tr>
<tr>
<td>Request $ct cores per task (multithreading)</td>
<td>-c $ct</td>
</tr>
<tr>
<td>Request $nt total # of tasks</td>
<td>-n $nt</td>
</tr>
<tr>
<td>Request to start job at specific $time</td>
<td>--begin $time</td>
</tr>
<tr>
<td>Specify job name as $name</td>
<td>-J $name</td>
</tr>
<tr>
<td>Specify job partition</td>
<td>-p $partition</td>
</tr>
<tr>
<td>Specify QOS</td>
<td>--qos $qos</td>
</tr>
<tr>
<td>Specify account</td>
<td>-A $account</td>
</tr>
<tr>
<td>Specify email address</td>
<td>--mail-user=$email</td>
</tr>
<tr>
<td>Request email on event</td>
<td>--mail-type=all[,begin,end,fail]</td>
</tr>
</tbody>
</table>

Use the above actions in a batch script

#SBATCH $option

- **Diff. between** `-N`, `-c`, `-n`, `--ntasks-per-node`, `--ntasks-per-core`?
- Normally you’d specify `-N` and `--ntasks-per-node`
  - fix the latter to 1 and add `-c` for MPI+OpenMP jobs
- If your application is scalable, just `-n` might be enough
  - iris is homogeneous (for now)
### SLURM workload manager

#### SLURM - more options for sbatch/srun

<table>
<thead>
<tr>
<th>Start job when... (dependencies)</th>
<th>sbatch/srun option</th>
</tr>
</thead>
<tbody>
<tr>
<td>these other jobs have started</td>
<td>-d after:$jobid1:$jobid2</td>
</tr>
<tr>
<td>these other jobs have ended</td>
<td>-d afterany:$jobid1:$jobid2</td>
</tr>
<tr>
<td>these other jobs have ended with no errors</td>
<td>-d afterok:$jobid1:$jobid2</td>
</tr>
<tr>
<td>these other jobs have ended with errors</td>
<td>-d afternok:$jobid1:$jobid2</td>
</tr>
<tr>
<td>all other jobs with the same name have ended</td>
<td>-d singleton</td>
</tr>
</tbody>
</table>

*Job dependencies and especially "singleton" can be very useful!*
### SLURM workload manager

#### SLURM - more options for sbatch/srun

<table>
<thead>
<tr>
<th>Start job when... (dependencies)</th>
<th>sbatch/srun option</th>
</tr>
</thead>
<tbody>
<tr>
<td>these other jobs have started</td>
<td>-d after:$jobid1:$jobid2</td>
</tr>
<tr>
<td>these other jobs have ended</td>
<td>-d afterany:$jobid1:$jobid2</td>
</tr>
<tr>
<td>these other jobs have ended with no errors</td>
<td>-d afterok:$jobid1:$jobid2</td>
</tr>
<tr>
<td>these other jobs have ended with errors</td>
<td>-d afternok:$jobid1:$jobid2</td>
</tr>
<tr>
<td>all other jobs with the same name have ended</td>
<td>-d singleton</td>
</tr>
</tbody>
</table>

**Job dependencies and especially "singleton" can be very useful!**

<table>
<thead>
<tr>
<th>Allocate job at... (specified time)</th>
<th>sbatch/srun option</th>
</tr>
</thead>
<tbody>
<tr>
<td>exact time today</td>
<td>--begin=16:00</td>
</tr>
<tr>
<td>tomorrow</td>
<td>--begin=tomorrow</td>
</tr>
<tr>
<td>specific time relative to now</td>
<td>--begin=now+2hours</td>
</tr>
<tr>
<td>given date and time</td>
<td>--begin=2017-06-23T07:30:00</td>
</tr>
</tbody>
</table>

Jobs run like this will wait as PD – Pending with "(BeginTime)" reason
## SLURM - more options for sbatch/srun

### Start job when... (dependencies)

<table>
<thead>
<tr>
<th>Event Description</th>
<th>sbatch/srun Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>these other jobs have started</td>
<td>-d after:$jobid1:$jobid2</td>
</tr>
<tr>
<td>these other jobs have ended</td>
<td>-d afterany:$jobid1:$jobid2</td>
</tr>
<tr>
<td>these other jobs have ended with no errors</td>
<td>-d afterok:$jobid1:$jobid2</td>
</tr>
<tr>
<td>these other jobs have ended with errors</td>
<td>-d afternok:$jobid1:$jobid2</td>
</tr>
<tr>
<td>all other jobs with the same name have ended</td>
<td>-d singleton</td>
</tr>
</tbody>
</table>

*Job dependencies and especially "singleton" can be very useful!*

### Allocate job at... (specified time)

<table>
<thead>
<tr>
<th>Time Specification</th>
<th>sbatch/srun Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>exact time today</td>
<td>--begin=16:00</td>
</tr>
<tr>
<td>tomorrow</td>
<td>--begin=tomorrow</td>
</tr>
<tr>
<td>specific time relative to now</td>
<td>--begin=now+2hours</td>
</tr>
<tr>
<td>given date and time</td>
<td>--begin=2017-06-23T07:30:00</td>
</tr>
</tbody>
</table>

*Jobs run like this will wait as PD – Pending with "(BeginTime)" reason*

### Other scheduling request

<table>
<thead>
<tr>
<th>Request Description</th>
<th>sbatch/srun Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ask for minimum/maximum # of nodes</td>
<td>-N minnodes-maxnodes</td>
</tr>
<tr>
<td>Ask for minimum run time (start job faster)</td>
<td>--time-min=d-hh:mm:ss</td>
</tr>
<tr>
<td>Ask to remove job if deadline can’t be met</td>
<td>--deadline=YYYY-MM-DD[THH:MM[:SS]]</td>
</tr>
<tr>
<td>Run job within pre-created (admin) reservation</td>
<td>--reservation=$reservationname</td>
</tr>
<tr>
<td>Allocate resources as specified job</td>
<td>--jobid=$jobid</td>
</tr>
</tbody>
</table>

*Can use --jobid to connect to running job (different than sattach!)*
SLURM workload manager

**SLURM - environment variables**

- 53 input env. vars. can be used to define job parameters
  - almost all have a command line equivalent
- up to 59 output env. vars. available within job environment
  - some common ones:

<table>
<thead>
<tr>
<th>Description</th>
<th>Environment variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job ID</td>
<td>$SLURM_JOBID</td>
</tr>
<tr>
<td>Job name</td>
<td>$SLURM_JOB_NAME</td>
</tr>
<tr>
<td>Name of account under which job runs</td>
<td>$SLURM_JOB_ACCOUNT</td>
</tr>
<tr>
<td>Name of partition job is running in</td>
<td>$SLURM_JOB_PARTITION</td>
</tr>
<tr>
<td>Name of QOS the job is running with</td>
<td>$SLURM_JOB_QOS</td>
</tr>
<tr>
<td>Name of job’s advance reservation</td>
<td>$SLURM_JOB_RESERVATION</td>
</tr>
<tr>
<td>Job submission directory</td>
<td>$SLURM_SUBMIT_DIR</td>
</tr>
<tr>
<td>Number of nodes assigned to the job</td>
<td>$SLURM_NNODES</td>
</tr>
<tr>
<td>Name of nodes assigned to the job</td>
<td>$SLURM_JOB_NODELIST</td>
</tr>
<tr>
<td>Number of tasks for the job</td>
<td>$SLURM_NTASKS or $SLURM_NPROCS</td>
</tr>
<tr>
<td>Number of cores for the job on current node</td>
<td>$SLURM_JOB_CPUS_PER_NODE</td>
</tr>
<tr>
<td>Memory allocated to the job per node</td>
<td>$SLURM_MEM_PER_NODE</td>
</tr>
<tr>
<td>Memory allocated per core</td>
<td>$SLURM_MEM_PER_CPU</td>
</tr>
<tr>
<td>Task count within a job array</td>
<td>$SLURM_ARRAY_TASK_COUNT</td>
</tr>
<tr>
<td>Task ID assigned within a job array</td>
<td>$SLURM_ARRAY_TASK_ID</td>
</tr>
</tbody>
</table>

Outputting these variables to the job log is essential for bookkeeping!
Usage examples (I)

> Interactive jobs

```
srun -p interactive --qos qos-interactive --time=0:30 -N2 --ntasks-per-node=4 --pty bash -i
srun -p interactive --qos qos-interactive --pty --x11 bash -i
srun -p interactive --qos qos-besteffort --pty bash -i
```
SLURM workload manager

Usage examples (I)

> Interactive jobs

sr

2

Interactive jobs

> Batch jobs

sbatch job.sh
sbatch -N 2 job.sh
sbatch -p batch --qos qos-batch job.sh
sbatch -p long --qos qos-long job.sh
sbatch --begin=2017-06-23T07:30:00 job.sh
sbatch -p batch --qos qos-besteffort job.sh
SLURM workload manager

Usage examples (I)

> Interactive jobs

- `srun -p interactive --qos qos-interactive --time=0:30 -N2 --ntasks-per-node=4 --pty bash -i`
- `srun -p interactive --qos qos-interactive --pty --x11 bash -i`
- `srun -p interactive --qos qos-besteffort --pty bash -i`

> Batch jobs

- `sbatch job.sh`
- `sbatch -N 2 job.sh`
- `sbatch -p batch --qos qos-batch job.sh`
- `sbatch -p long --qos qos-long job.sh`
- `sbatch --begin=2017-06-23T07:30:00 job.sh`
- `sbatch -p batch --qos qos-besteffort job.sh`

Status and details for partitions, nodes, reservations

- `squeue / squeue -l / squeue -la / squeue -l -p batch / squeue -t PD`
- `scontrol show nodes / scontrol show nodes $nodename`
- `sinfo / sinfo -s / sinfo -N`
- `sinfo -T`
Collecting job information, priority, expected start time

scontrol show job $jobid # this is only available while job is in the queue + 5 minutes
sprio -l
squeue --start -u $USER
Collecting job information, priority, expected start time

- `scontrol show job $jobid` # this is only available while job is in the queue + 5 minutes
- `sprio -l`
- `squeue --start -u $USER`

Running job metrics – sstat tool

- `sstat -j $jobid / sstat -j $jobid -l`
- `sstat -j $jobid1 --format=AveCPU,AveRSS,AveVMSize,MaxRSS,MaxVMSize`
- `sstat -p -j $jobid1,$jobid2 --format=AveCPU,AveRSS,AveVMSize,MaxRSS,MaxVMSize`
Usage examples (II)

Collecting job information, priority, expected start time

```
scontrol show job $jobid # this is only available while job is in the queue + 5 minutes
sprio -l
squeue --start -u $USER
```

Running job metrics – sstat tool

```
sstat -j $jobid / sstat -j $jobid -l
sstat -j $jobid1 --format=AveCPU,AveRSS,AveVMSize,MaxRSS,MaxVMSize
sstat -p -j $jobid1,$jobid2 --format=AveCPU,AveRSS,AveVMSize,MaxRSS,MaxVMSize
```

Completed job metrics – sacct tool

```
sacct -j $jobid / sacct -j $jobid -l
sacct -p -j $jobid --format=account, user, jobid, jobname, partition, state, elapsed, elapsedraw,\ start, end, maxrss, maxvmsize, consumedenergy, consumedenergysraw, nnodes, ncpus, nodelist
sacct --starttime 2017-06-12 -u $USER
```
Usage examples (III)

**Controlling queued and running jobs**

scontrol hold $jobid
scontrol release $jobid
scontrol suspend $jobid
scontrol resume $jobid
scancel $jobid
scancel -n $jobname
scancel -u $USER
scancel -u $USER -p batch
scontrol requeue $jobid
Controlling queued and running jobs

- `scontrol hold $jobid`
- `scontrol release $jobid`
- `scontrol suspend $jobid`
- `scontrol resume $jobid`
- `scancel $jobid`
- `scancel -n $jobname`
- `scancel -u $USER`
- `scancel -u $USER -p batch`
- `scontrol requeue $jobid`

Checking accounting links and QOS available for you

- `sacctmgr show user $USER format=user%20s,defaultaccount%30s`
- `sacctmgr list association where users=$USER format=account%30s,user%20s,qos%120s`
Controlling queued and running jobs

- `scontrol hold $jobid`
- `scontrol release $jobid`
- `scontrol suspend $jobid`
- `scontrol resume $jobid`
- `scancel $jobid`
- `scancel -n $jobname`
- `scancel -u $USER`
- `scancel -u $USER -p batch`
- `scontrol requeue $jobid`

Checking accounting links and QOS available for you

- `sacctmgr show user $USER format=user%20s,defaultaccount%30s`
- `sacctmgr list association where users=$USER format=account%30s,user%20s,qos%120s`

Checking accounting share info - usage, fair-share, etc.

- `sshare -U`
- `sshare -A $accountname`
- `sshare -A $(sacctmgr -n show user $USER format=defaultaccount%30s)`
- `sshare -a`
#!/bin/bash -l
#SBATCH --N 1
#SBATCH --ntasks-per-node=1
#SBATCH --time=0-00:05:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

echo "Hello from the batch queue on node ${SLURM_NODELIST}"
# Your more useful application can be started below!

Submit it with: sbatch launcher.sh
#!/bin/bash -l
#SBATCH -N 2
#SBATCH --ntasks-per-node=2
#SBATCH --time=0-03:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

echo "== Starting run at $(date)"
echo "== Job ID: ${SLURM_JOBID}"     
echo "== Node list: ${SLURM_NODELIST}"
echo "== Submit dir. : ${SLURM_SUBMIT_DIR}" 
# Your more useful application can be started below!
#!/bin/bash
#SBATCH --mail-type=end,fail
#SBATCH --mail-user=Your.Email@Address.lu
#SBATCH -N 2
#SBATCH --ntasks-per-node=2
#SBATCH --time=0-03:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

echo "== Starting run at $(date)"
echo "== Job ID: ${SLURM_JOBID}" 
echo "== Node list: ${SLURM_NODELIST}"
echo "== Submit dir.: ${SLURM_SUBMIT_DIR}"
# Your more useful application can be started below!
#!/bin/bash -l
#SBATCH -J MyLargeMemorySequentialJob
#SBATCH --mail-type=end,fail
#SBATCH --mail-user=Your.Email@Address.lu
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --mem=64GB
#SBATCH --time=1-00:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

echo "== Starting run at $(date)"

Use "mem" to request (more) memory per node for low #core jobs
#!/bin/bash -l
#SBATCH -J MyLongJob
#SBATCH --mail-type=all
#SBATCH --mail-user=Your.Email@Address.lu
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --time=3-00:00:00
#SBATCH -p long
#SBATCH --qos=qos-long

echo "== Starting run at $(date)"
echo "== Job ID: ${SLURM_JOBID}"
echo "== Node list: ${SLURM_NODELIST}"
echo "== Submit dir. : ${SLURM_SUBMIT_DIR}"
# Your more useful application can be started below!

Longer walltime now possible but you should not (!) rely on it.
Always prefer batch and requeue-able jobs.
SLURM workload manager

Job launchers - besteffort

```bash
#!/bin/bash -l
#SBATCH -J MyRerunnableJob
#SBATCH --mail-type=end,fail
#SBATCH --mail-user=Your.Email@Address.lu
#SBATCH -N 1
#SBATCH --ntasks-per-node=28
#SBATCH --time=0-12:00:00
#SBATCH -p batch
#SBATCH --qos=qos-besteffort
#SBATCH --requeue

echo "== Starting run at $(date)"
echo "== Job ID: ${SLURM_JOBID}"
echo "== Node list: ${SLURM_NODELIST}"
echo "== Submit dir. : ${SLURM_SUBMIT_DIR}"
# Your more useful application can be started below!
```

Many scientific applications support internal state saving and restart!
We will also discuss system-level checkpoint-restart with DMTCP.
SLURM workload manager

Job launchers - threaded parallel

```
#!/bin/bash -l
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH -c 28
#SBATCH --time=0-01:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}
/path/to/your/threaded.app
```

By threaded we mean pthreads/OpenMP shared-memory applications.
SLURM workload manager

Job launchers - MATLAB

```
#!/bin/bash -l
#SBATCH -N 1
#SBATCH --ntasks-per-node=28
#SBATCH -c 1
#SBATCH --time=0-01:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

gl module load base/MATLAB
matlab -nodisplay -nosplash < /path/to/infile > /path/to/outfile
```

MATLAB spawns processes, limited for now to single node execution. We are still waiting for Distributed Computing Server availability.
SLURM workload manager

Job launchers - MATLAB

```bash
#!/bin/bash -l
#SBATCH -N 1
#SBATCH --ntasks-per-node=28
#SBATCH -c 1
#SBATCH --time=0-01:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

module load base/MATLAB
matlab -nodisplay -nosplash < /path/to/infile > /path/to/outfile
```

MATLAB spawns processes, limited for now to single node execution. We are still waiting for Distributed Computing Server availability.
Currently the iris cluster is homogeneous. Its core networking is a non-blocking fat-tree.

- For now simply requesting a number of tasks (with 1 core/task) should be performant

- Different MPI implementations will however behave differently
  - very recent/latest versions available on iris for IntelMPI, OpenMPI, MVAPICH2
  - we ask that you let us know any perceived benefit for your applications when using one or the other

- We can make available optimized MPI-layer parameters obtained during our tuning runs
  - and hope they will improve even more your time to solution
#!/bin/bash -l
#SBATCH -n 128
#SBATCH -c 1
#SBATCH --time=00:01:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

module load toolchain/intel
srun -n $SLURM_NTASKS /path/to/your/intel-toolchain-compiled-app

IntelMPI is configured to use PMI2 for process management (optimal).
Bare mpirun will not work for now.
SLURM workload manager

Job launchers - OpenMPI

```bash
#!/bin/bash -l
#SBATCH -n 128
#SBATCH -c 1
#SBATCH --time=0-01:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

module load toolchain/foss
srun -n $SLURM_NTASKS /path/to/your/foss-toolchain-compiled-app
```

OpenMPI also uses PMI2 (again, optimal).
Bare mpirun does work but is not recommended.

You can easily generate a hostfile from within a SLURM job with:
```
srun hostname | sort -n > hostfile
```
SLURM workload manager

Job launchers - MPI+OpenMP

```bash
#!/bin/bash -l
#SBATCH -N 10
#SBATCH --ntasks-per-node=1
#SBATCH -c 28
#SBATCH --time=0-01:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch
module load toolchain/intel
export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}
srun -n $SLURM_NTASKS /path/to/your/parallel-hybrid-app
```

Compile and use your applications in hybrid MPI+OpenMP mode when you can for better (best?) possible performance.
Summary

1 Introduction

2 SLURM workload manager
   SLURM concepts and design for iris
   Running jobs with SLURM

3 OAR and SLURM

4 Conclusion
OAR and SLURM

Notes on OAR

- OAR will remain the workload manager of Gaia and Chaos
  - celebrating 4250995 jobs on Gaia! (2017-11-07)
  - celebrating 1615659 jobs on Chaos! (2017-11-07)

- Many of its features are common to other workload managers, incl. SLURM
  - some things are exactly the same
  - but some things work in a different way
  - ... and some have no equivalent or are widely different

- An adjustment period for you is needed if you’ve only used OAR
  - next slides show a brief transition guide
## OAR and SLURM

### OAR/SLURM - commands guide

<table>
<thead>
<tr>
<th>Command</th>
<th>OAR (gaia/chaos)</th>
<th>SLURM (iris)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Submit passive/batch job</td>
<td>oarsub -S $script</td>
<td>sbatch $script</td>
</tr>
<tr>
<td>Start interactive job</td>
<td>oarsub -I</td>
<td>srun -p interactive --qos qos-interactive --pty bash -i</td>
</tr>
<tr>
<td>Queue status</td>
<td>oarstat</td>
<td>squeue</td>
</tr>
<tr>
<td>User job status</td>
<td>oarstat -u $user</td>
<td>squeue -u $user</td>
</tr>
<tr>
<td>Specific job status (detailed)</td>
<td>oarstat -f -j $jobid</td>
<td>scontrol show job $jobid</td>
</tr>
<tr>
<td>Delete (running/waiting) job</td>
<td>oardel $jobid</td>
<td>scancel $jobid</td>
</tr>
<tr>
<td>Hold job</td>
<td>oarhold $jobid</td>
<td>scontrol hold $jobid</td>
</tr>
<tr>
<td>Resume held job</td>
<td>oarresume $jobid</td>
<td>scontrol release $jobid</td>
</tr>
<tr>
<td>Node list and properties</td>
<td>oarnodes</td>
<td>scontrol show nodes</td>
</tr>
</tbody>
</table>

**Similar yet different?**

Many specifics will actually come from the way Iris is set up.
### OAR/SLURM - job specifications

<table>
<thead>
<tr>
<th>Specification</th>
<th>OAR</th>
<th>SLURM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Script directive</td>
<td>#OAR</td>
<td>#SBATCH</td>
</tr>
<tr>
<td>Nodes request</td>
<td>-l nodes=$count</td>
<td>-N $min-$max</td>
</tr>
<tr>
<td>Cores request</td>
<td>-l core=$count</td>
<td>-n $count</td>
</tr>
<tr>
<td>Cores-per-node request</td>
<td>-l nodes=$ncount/core=$ccount</td>
<td>--ntasks-per-node=$ccount</td>
</tr>
<tr>
<td>Walltime request</td>
<td>-l [...],walltime=hh:mm:ss</td>
<td>-t $min OR -t $days-hh:mm:ss</td>
</tr>
<tr>
<td>Job array</td>
<td>--array $count</td>
<td>--array $specification</td>
</tr>
<tr>
<td>Job name</td>
<td>-n $name</td>
<td>-J $name</td>
</tr>
<tr>
<td>Job dependency</td>
<td>-a $jobid</td>
<td>-d $specification</td>
</tr>
<tr>
<td>Property request</td>
<td>-p &quot;$property=$value&quot;</td>
<td>-C $specification</td>
</tr>
</tbody>
</table>

Job specifications will need most adjustment on your side... but thankfully Iris has a homogeneous configuration. Running things in an optimal way will be much easier.
## OAR and SLURM

### OAR/SLURM - env. vars.

<table>
<thead>
<tr>
<th>Environment variable</th>
<th>OAR</th>
<th>SLURM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job ID</td>
<td>$OAR_JOB_ID</td>
<td>$SLURM_JOB_ID</td>
</tr>
<tr>
<td>Resource list</td>
<td>$OAR_NODEFILE</td>
<td>$SLURM_NODELIST #List not file! See below.</td>
</tr>
<tr>
<td>Job name</td>
<td>$OAR_JOB_NAME</td>
<td>$SLURM_JOB_NAME</td>
</tr>
<tr>
<td>Submitting user name</td>
<td>$OAR_USER</td>
<td>$SLURM_JOB_USER</td>
</tr>
<tr>
<td>Task ID within job array</td>
<td>$OAR_ARRAY_INDEX</td>
<td>$SLURM_ARRAY_TASK_ID</td>
</tr>
<tr>
<td>Working directory at submission</td>
<td>$OAR_WORKING_DIRECTORY</td>
<td>$SLURM_SUBMIT_DIR</td>
</tr>
</tbody>
</table>

**Check available variables:** env | egrep "OAR|SLURM"

**Generate hostfile:** srun hostname | sort -n > hostfile
Summary

1. Introduction

2. SLURM workload manager
   SLURM concepts and design for iris
   Running jobs with SLURM

3. OAR and SLURM

4. Conclusion
Conclusion and Practical Session start

We’ve discussed

- The design of SLURM for the **iris** cluster
- The permissions system in use through group accounts and QOS
- Main SLURM tools and how to use them
- Job types possible with SLURM on **iris**
- SLURM job launchers for sequential and parallel applications
- Transitioning from OAR to SLURM

And now..

**Short DEMO time!**
Conclusion

Conclusion and Practical Session start

We’ve discussed

- The design of SLURM for the iris cluster
- The permissions system in use through group accounts and QOS
- Main SLURM tools and how to use them
- Job types possible with SLURM on iris
- SLURM job launchers for sequential and parallel applications
- Transitioning from OAR to SLURM

And now..

Short DEMO time!

Your Turn!
1. Introduction

2. SLURM workload manager
   SLURM concepts and design for iris
   Running jobs with SLURM

3. OAR and SLURM

4. Conclusion
Resource and Job Management Systems

- **Resource and Job Management System (RJMS)**
  - “Glue” for a parallel computer to execute parallel jobs
  - **Goal:** satisfy users’ demands for computation
    - ✓ assign resources to user jobs with an efficient manner
**Resource and Job Management Systems**

- **Resource and Job Management System (RJMS)**
  - "Glue" for a parallel computer to execute parallel jobs
  - **Goal**: satisfy users’ demands for computation
    - assign resources to user jobs with an efficient manner

- **HPC Resources**:
  - Nodes (typically a unique IP address)
    - Sockets / Cores / Hyperthreads
    - Memory
    - Interconnect/switch resources
  - Generic resources (e.g. GPUs)
  - Licenses

- **Strategic Position**
  - Direct/constant knowledge of resources
  - Launch and otherwise manage jobs
Resource Allocation involves three principal abstraction layers:

- **Job Management:**
  - declaration of a job & demand of resources and job characteristics,

- **Scheduling:** matching of the jobs upon the resources,

- **Resource Management:**
  - launching and placement of job instances...
  - ... along with the job’s control of execution

When there is more work than resources

- the job scheduler manages queue(s) of work
  - supports complex scheduling algorithms

- Supports resource limits (by queue, user, group, etc.)
Backup slides

RJMS Detailed Components

- **Resource Management**
  - Resource Treatment (hierarchy, partitions,..)
  - Job Launching, Propagation, Execution control
  - Task Placement (topology, binding,..)
  - **Advanced Features:**
    - High Availability, Energy Efficiency, Topology aware placement
**Resource Management**

- Resource Treatment (hierarchy, partitions,..)
- Job Launching, Propagation, Execution control
- Task Placement (topology, binding,..)
- **Advanced Features:**
  - High Availability, Energy Efficiency, Topology aware placement

**Job Management**

- Job declaration and control (signaling, reprioritizing,..)
- Monitoring (reporting, visualization,..)
- **Advanced Features:**
  - Authentication (limitations, security,..)
  - QOS (checkpoint, suspend, accounting,..)
  - Interfacing (MPI libraries, debuggers, APIs,..)
Backup slides

RJMS Detailed Components

- **Resource Management**
  - Resource Treatment (hierarchy, partitions, ..)
  - Job Launching, Propagation, Execution control
  - Task Placement (topology, binding, ..)
  - **Advanced Features:**
    - High Availability, Energy Efficiency, Topology aware placement

- **Job Management**
  - Job declaration and control (signaling, reprioritizing, ..)
  - Monitoring (reporting, visualization, ..)
  - **Advanced Features:**
    - Authentication (limitations, security, ..)
    - QOS (checkpoint, suspend, accounting, ..)
    - Interfacing (MPI libraries, debuggers, APIs, ..)

- **Scheduling**
  - Queues Management (priorities, multiple, ..)
  - Advanced Reservation
Job Scheduling
Job Scheduling (backfilling)
Job Scheduling (suspension & requeue)

Backfill Scheduler +
Job Preemption (Suspension)

Backfill Scheduler +
Job Preemption (C&R)

Checkpoint ==>
## Main Job Schedulers

<table>
<thead>
<tr>
<th>Name</th>
<th>Company</th>
<th>Version*</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLURM</td>
<td>SchedMD</td>
<td>17.02.8</td>
</tr>
<tr>
<td>LSF</td>
<td>IBM</td>
<td>10.1</td>
</tr>
<tr>
<td>OpenLava</td>
<td>LSF Fork</td>
<td>2.2</td>
</tr>
<tr>
<td>MOAB/Torque</td>
<td>Adaptative Computing</td>
<td>6.1</td>
</tr>
<tr>
<td>PBS</td>
<td>Altair</td>
<td>13.0</td>
</tr>
<tr>
<td>OAR (PBS Fork)</td>
<td>LIG</td>
<td>2.5.7</td>
</tr>
<tr>
<td>Oracle Grid Engine (formerly SGE)</td>
<td>Oracle</td>
<td></td>
</tr>
</tbody>
</table>

* As of Oct. 2017
## Main Job Schedulers

<table>
<thead>
<tr>
<th>Name</th>
<th>Company</th>
<th>Version*</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLURM</td>
<td>SchedMD</td>
<td>17.02.8</td>
</tr>
<tr>
<td>LSF</td>
<td>IBM</td>
<td>10.1</td>
</tr>
<tr>
<td>OpenLava</td>
<td>LSF Fork</td>
<td>2.2</td>
</tr>
<tr>
<td>MOAB/Torque</td>
<td>Adaptative Computing</td>
<td>6.1</td>
</tr>
<tr>
<td>PBS</td>
<td>Altair</td>
<td>13.0</td>
</tr>
<tr>
<td>OAR (PBS Fork)</td>
<td>LIG</td>
<td>2.5.7</td>
</tr>
<tr>
<td>Oracle Grid Engine (formely SGE)</td>
<td>Oracle</td>
<td></td>
</tr>
</tbody>
</table>

*: As of Oct. 2017
UL HPC resource manager: OAR

The OAR Batch Scheduler

- Versatile resource and task manager
  - schedule **jobs** for users on the cluster **resource**
  - OAR resource = a node or part of it (CPU/core)
  - OAR job = execution time (**walltime**) on a set of resources

http://oar.imag.fr
UL HPC resource manager: OAR

The OAR Batch Scheduler

- Versatile resource and task manager
  - schedule **jobs** for users on the cluster **resource**
  - OAR resource = a node or part of it (CPU/core)
  - OAR job = execution time (**walltime**) on a set of resources

OAR main features includes:

- **interactive vs. passive (aka. batch) jobs**
- **best effort jobs**: use more resource, accept their release any time
- **deploy jobs** (**Grid5000 only**): deploy a customized OS environment
  - ... and have full (root) access to the resources
- **powerful resource filtering/matching**
Main OAR commands

- **oarsub**: submit/reserve a job (by default: 1 core for 2 hours)
- **oardel**: delete a submitted job
- **oarnodes**: shows the resources states
- **oarstat**: shows information about running or planned jobs

<table>
<thead>
<tr>
<th>Submission</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>interactive</strong></td>
<td>oarsub [options] -I</td>
</tr>
<tr>
<td><strong>passive</strong></td>
<td>oarsub [options] scriptName</td>
</tr>
</tbody>
</table>

- Each created job receive an identifier JobID
  - Default passive job log files: OAR.JobID.std{out,err}
- You can make a reservation with `-r "YYYY-MM-DD HH:MM:SS"`
Backup slides

Main OAR commands

- `oarsub` submit/reserve a job (by default: **1 core for 2 hours**)
- `oardel` delete a submitted job
- `oarnodes` shows the resources states
- `oarstat` shows information about running or planned jobs

```
<table>
<thead>
<tr>
<th>Submission</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>interactive</strong></td>
</tr>
<tr>
<td>oarsub [options] -I</td>
</tr>
<tr>
<td><strong>passive</strong></td>
</tr>
<tr>
<td>oarsub [options] scriptName</td>
</tr>
</tbody>
</table>
```

- Each created job receive an identifier **JobID**
  - Default passive job log files: OAR.JobID.std{out,err}
- You can make a reservation with `-r "YYYY-MM-DD HH:MM:SS"`

Direct access to nodes by `ssh` is forbidden: use `oarsh` instead
OAR job environment variables

Once a job is created, some environments variables are defined:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$OAR_NODEFILE</td>
<td>Filename which lists all reserved nodes for this job</td>
</tr>
<tr>
<td>$OAR_JOB_ID</td>
<td>OAR job identifier</td>
</tr>
<tr>
<td>$OAR_RESOURCE_PROPERTIES_FILE</td>
<td>Filename which lists all resources and their properties</td>
</tr>
<tr>
<td>$OAR_JOB_NAME</td>
<td>Name of the job given by the &quot;-n&quot; option of oarsub</td>
</tr>
<tr>
<td>$OAR_PROJECT_NAME</td>
<td>Job project name</td>
</tr>
</tbody>
</table>

Useful for MPI jobs for instance:

$> mpirun -machinefile $OAR\_NODEFILE /path/to/myprog

... Or to collect how many cores are reserved per node:

$> cat $OAR\_NODEFILE | uniq -c
OAR job types

<table>
<thead>
<tr>
<th>Job Type</th>
<th>Max Walltime (hour)</th>
<th>Max #active_jobs</th>
<th>Max #active_jobs_per_user</th>
</tr>
</thead>
<tbody>
<tr>
<td>interactive</td>
<td>12:00:00</td>
<td>10000</td>
<td>5</td>
</tr>
<tr>
<td>default</td>
<td>120:00:00</td>
<td>30000</td>
<td>10</td>
</tr>
<tr>
<td>besteffort</td>
<td>9000:00:00</td>
<td>10000</td>
<td>1000</td>
</tr>
</tbody>
</table>

cf /etc/oar/admission_rules/*.conf

- **interactive**: useful to test / prepare an experiment
  - you get a shell on the first reserved resource

- **best-effort vs. default**: nearly unlimited constraints **YET**
  - a besteffort job can be killed as soon as a default job as no other place to go
  - enforce checkpointing (and/or idempotent) strategy
Characterizing OAR resources

Specifying wanted resources in a hierarchical manner

- Use the `-l` option of `oarsub`. Main constraints:
  - `enclosure=N` number of enclosure
  - `nodes=N` number of nodes
  - `core=N` number of cores
  - `walltime=hh:mm:ss` job’s max duration

Specifying OAR resource properties

- Use the `-p` option of `oarsub`:
  Syntax: `-p "property=’value’"`
  - `gpu={'YES,NO}'` has (or not) a GPU card
  - `host=’fqdn’` full hostname of the resource
  - `network_address=’hostname’` Short hostname of the resource
  - `nodeclass={'k,b,h,d,r}'` Class of node
  (Chaos only)
OAR (interactive) job examples

- 2 cores on 3 nodes (same enclosure) for 3h15:  
  Total: 6 cores

(frontend)$> oarsub -I -l /enclosure=1/nodes=3/core=2,walltime=3:15
OAR (interactive) job examples

- 2 cores on 3 nodes (same enclosure) for 3h15: Total: 6 cores
  (frontend)$> oarsub -I -l /enclosure=1/nodes=3/core=2,walltime=3:15

- 4 cores on a GPU node for 8 hours Total: 4 cores
  (frontend)$> oarsub -I -l /core=4,walltime=8 -p "gpu='YES'"
OAR (interactive) job examples

- **2 cores on 3 nodes (same enclosure) for 3h15:**
  
  ```
  (frontend)$> oarsub -I -l /enclosure=1/nodes=3/core=2,walltime=3:15
  ```
  
  Total: 6 cores

- **4 cores on a GPU node for 8 hours**

  ```
  (frontend)$> oarsub -I -l /core=4,walltime=8 -p "gpu='YES'"
  ```
  
  Total: 4 cores

- **2 nodes among the h-cluster1-* nodes (Chaos only)**

  ```
  (frontend)$> oarsub -I -l nodes=2 -p "nodeclass='h'"
  ```
  
  Total: 24 cores
OAR (interactive) job examples

- 2 cores on 3 nodes (same enclosure) for 3h15:
  
  Total: 6 cores

  (frontend)$> oarsub -I -l /enclosure=1/nodes=3/core=2,walltime=3:15

- 4 cores on a GPU node for 8 hours

  Total: 4 cores

  (frontend)$> oarsub -I -l /core=4,walltime=8 -p "gpu='YES’"

- 2 nodes among the h-cluster1-* nodes

  (Chaos only) Total: 24 cores

  (frontend)$> oarsub -I -l nodes=2 -p "nodeclass='h’"

- 4 cores on 2 GPU nodes + 20 cores on other nodes

  Total: 28 cores

  $> oarsub -I -l "{gpu='YES’}/nodes=2/core=4+{gpu='NO’}/core=20"
2 cores on 3 nodes (same enclosure) for 3h15:  
(frontend)$> oarsub -I -l /enclosure=1/nodes=3/core=2,walltime=3:15

4 cores on a GPU node for 8 hours  
(frontend)$> oarsub -I -l /core=4,walltime=8 -p "gpu='YES'"

2 nodes among the h-cluster1-* nodes  
(Chaos only) (frontend)$> oarsub -I -l nodes=2 -p "nodeclass='h'"

4 cores on 2 GPU nodes + 20 cores on other nodes  
$> oarsub -I -l \{gpu='YES'\}/nodes=2/core=4+\{gpu='NO'\}/core=20"

A full big SMP node  
$> oarsub -t bigsmp -I 1 node=1

Total: 6 cores
Total: 4 cores
Total: 24 cores
Total: 28 cores
Total: 160 cores on gaia-74
Some other useful features of OAR

**Connect to a running job**

(frontend)$> oarsub -C JobID

**Cancel a job**

(frontend)$> oardel JobID

**Status of a jobs**

(frontend)$> oarstat -state -j JobID

**View the job**

(frontend)$> oarstat
(frontend)$> oarstat -f -j JobID

**Get info on the nodes**

(frontend)$> oarnodes
(frontend)$> oarnodes -l
(frontend)$> oarnodes -s

**Run a best-effort job**

(frontend)$> oarsub -t besteffort ...
OAR Practical session

Demo Time
- gaia or chaos UL cluster access
- Interactive / Passive job submission
Backup slides

Designing efficient OAR job launchers

Resources/Example
https://github.com/ULHPC/launcher-scripts

- UL HPC grant access to **parallel computing** resources
  - ideally: OpenMP/MPI/CUDA/OpenCL jobs
  - if serial jobs/tasks: run them efficiently

- Avoid to submit purely serial jobs to the OAR queue a
  - waste the computational power (11 out of 12 cores on gaia).
  - use whole nodes by running at least 12 serial runs at once

- **Key**: understand difference between Task and OAR job
Backup slides

Designing efficient OAR job launchers

Resources/Example

https://github.com/ULHPC/launcher-scripts

- UL HPC grant access to **parallel computing** resources
  - ideally: OpenMP/MPI/CUDA/OpenCL jobs
  - if serial jobs/tasks: run them efficiently
- Avoid to submit purely serial jobs to the OAR queue a
  - waste the computational power (11 out of 12 cores on gaia).
  - use whole nodes by running at least 12 serial runs at once
- **Key**: understand difference between **Task** and **OAR job**

For more information...

- Incoming Practical Session
  - HPC workflow with sequential jobs (C,python,java etc.)
# Simple interactive job

```
(access)$> oarsub -I
(node)$> echo $OAR_JOBID
4239985
(node)$> echo $OAR_NODEFILE
/var/lib/oar//4239985
(node)$> cat $OAR_NODEFILE | wc -l
8
(node)$> cat $OAR_NODEFILE
moonshot1-39
moonshot1-39
moonshot1-39
moonshot1-39
moonshot1-39
moonshot1-40
moonshot1-40
moonshot1-40
moonshot1-40
```
View existing job

# View YOUR jobs (remove -u to view all)

```
(access)$> oarstat -u
Job id   Name    User       Submission Date   S Queue
         ------    --------    ------------------- - ----------
        4239985  svarrette  2017-10-23 12:33:41 R default
```

---

V. Plugaru & UL HPC Team (University of Luxembourg)
(access)$> oarstat -f -j 4239985
Job_Id: 4239985
[...]
state = Running
wanted_resources = -l "{type = 'default'}/ibpool=1/host=2,walltime=2:0:0"
types = interactive, inner=4236343, moonshot
assigned_resources = 3309+3310+3311+3312+3313+3314+3315+3316
assigned_hostnames = moonshot1-39+moonshot1-40
queue = default
launchingDirectory = /home/users/svarrette
stdout_file = OAR.4239985.stdout
stderr_file = OAR.4239985.stderr
jobType = INTERACTIVE
properties = (((bigmem='NO' AND bigsmp='NO') AND dedicated='NO') AND os='debian8')
walltime = 2:0:0
initial_request = oarsub -I -l nodes=2 -t moonshot -t inner=4236343
message = R=8,W=2:0:0,J=I,T=inner|interactive|moonshot (Karma=1.341)
Access to an existing job: Attempt 1

# Get your job ID...

(access)\$> oarstat -u

- **Attempt 1**: Get assigned resources and ...
Access to an existing job: Attempt 1

# Get your job ID...
(access)$> oarstat -u

Attempt 1: Get assigned resources and ... ssh to it!

# Collect the assigned resources
(access)$> oarstat -f -j 4239985 | grep hostname
assigned_hostnames = moonshot1-39+moonshot1-40

(access)$> ssh moonshot1-39
[...]
==================================================================
!/\ WARNING: Direct login by ssh is forbidden.

Use oarsub(1) to reserve nodes, and oarsh(1) to connect to your reserved nodes, typically by:
    OAR_JOB_ID=<jobid> oarsh <nodename>
==================================================================
Access to an existing job

- **Using oarsh:**

```
# Get your job ID...
(access)$> oarstat -u
# ... get the hostname of the nodes allocated ...
(access)$> oarstat -f -j 4239985 | grep hostname
# ... and connect to it with oarsh
(access)$> OAR_JOB_ID=4239985 oarsh moonshot1-39
```
Backup slides

Access to an existing job

● **Using oarsh:**

```bash
# Get your job ID...
(access)$> oarstat -u
# ... get the hostname of the nodes allocated ...  
(access)$> oarstat -f -j 4239985 | grep hostname 
# ... and connect to it with oarsh
(access)$> OAR_JOB_ID=4239985 oarsh moonshot1-39
```

● (better) **Using oarsub -C:**

```bash
# Get your job ID...
(access)$> oarstat -u
# ... and connect to the FIRST node of the reservation
(access)$> oarsub -C 4239985
```
Backup slides

**MPI jobs**

- Intel MPI

  (node)>
  module load toolchain/intel
  # ONLY on moonshot node have no IB card: export I_MPI_FABRICS=tcp
  (node)>
  mpirun -hostfile $OAR_NODEFILE /path/to/mpiprog

- OpenMPI:

  (node)>
  module load mpi/OpenMPI
  (node)>
  mpirun -hostfile $OAR_NODEFILE -x PATH -x LD_LIBRARY_PATH
       /path/to/mpiprog

- For more details: See MPI sessions

  https://github.com/ULHPC/launcher-scripts/blob/devel/bash/MPI/mpi_launcher.sh
OAR Launcher Scripts

$> oarsub -S <scriptname>  # -S: interpret #OAR comments

- Our launcher scripts on Github: https://github.com/ULHPC/launcher-scripts
  ➔ see in particular our generic launcher compliant w. OAR & SLURM

- Example:

```
#!/bin/bash
#OAR -l nodes=2/core=1,walltime=1
#OAR -n MyNamedJob
# Prepare UL HPC modules
if [ -f /etc/profile ]; then
  . /etc/profile
fi

module load toolchain/intel
/path/to/prog <ARGS>
```
**Slurm Workload Manager**

- **Simple Linux Utility for Resource Management**
  - Development started in 2002
    - Initially as a simple resource manager for Linux clusters
  - Has evolved into a capable job scheduler through use of opt. plugins
  - About 500,000 lines of C code today.
  - Supports AIX, Linux, Solaris, other Unix variants
- Used on many of the world’s largest computers
Slurm Workload Manager

- **Simple Linux Utility for Resource Management**
  - Development started in 2002
  - Initially as a simple resource manager for Linux clusters
  - Has evolved into a capable job scheduler through use of opt. plugins
  - About 500,000 lines of C code today.
  - Supports AIX, Linux, Solaris, other Unix variants
- Used on many of the world's largest computers

- **Now deployed on new UL HPC clusters**
  - Starting iris cluster (2017)
**Slurm Design Goals**

- **Small and simple**
- **Highly scalable** and **Fast**
  - managing 1.6 million core IBM BlueGene/Q,
  - tested to 33 million cores using emulation
  - throughput: up to 600 jobs p.s. & 1000 job submissions p.s.
- **Modular:**
  - plugins to support ≠ scheduling policies, MPI libraries...
- **Secure and Fault-tolerant**
  - highly tolerant of system failures
- **Power Management** and detailed monitoring

- **Open source:** GPL v2, active world-wide development
- **Portable:** written in C with a GNU autoconf configuration engine
Backup slides

Slurm Docs and Resources

- User and Admins latest documentation:
  - [http://slurm.schedmd.com/documentation.html](http://slurm.schedmd.com/documentation.html)

- Detailed man pages for commands and configuration files
  - [http://slurm.schedmd.com/man_index.html](http://slurm.schedmd.com/man_index.html)

- All SLURM related publications and presentations:
  - [http://slurm.schedmd.com/publications.html](http://slurm.schedmd.com/publications.html)

ULHPC Documentation & comparison to OAR

[https://hpc.uni.lu/users/docs/scheduler.html](https://hpc.uni.lu/users/docs/scheduler.html)
Puppet module ULHPC/slurm

Developed by the UL HPC Team – see https://forge.puppet.com/ULHPC/

Used in production on iris cluster

see also Slurm Control Repo Example

<table>
<thead>
<tr>
<th>Puppet Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>slurm</td>
<td>The main slurm class, piloting all aspects of the configuration</td>
</tr>
<tr>
<td>slurm::slurmdbd</td>
<td>Specialized class for Slurmdbd, the Slurm Database Daemon.</td>
</tr>
<tr>
<td>slurm::slurmctld</td>
<td>Specialized class for Slurmctld, the central management daemon of Slurm.</td>
</tr>
<tr>
<td>slurm::slurmd</td>
<td>Specialized class for Slurmd, the compute node daemon for Slurm.</td>
</tr>
<tr>
<td>slurm::munge</td>
<td>Manages MUNGE, an authentication service for creating and validating credentials.</td>
</tr>
<tr>
<td>slurm::pam</td>
<td>Handle PAM aspects for SLURM (Memlock for MPI etc.)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Puppet Defines</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>slurm::download</td>
<td>takes care of downloading the SLURM sources for a given version passed as resource name</td>
</tr>
<tr>
<td>slurm::build</td>
<td>building Slurm sources into packages (_i.e. RPMs for the moment)</td>
</tr>
<tr>
<td>slurm::install::packages</td>
<td>installs the Slurm packages, typically built from slurm::build</td>
</tr>
<tr>
<td>slurm::acct::*</td>
<td>adding (or removing) accounting resources to the slurm database</td>
</tr>
</tbody>
</table>
SLURM Architecture

One daemon per node

slurmd

Cluster-wide control daemon

slurmctld (primary)

slurmctld (backup)

User commands

scancel  scontrol  sinfo  smap  squeue  srun
Backup slides

SLURM Entities

<table>
<thead>
<tr>
<th>Entity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computing node</td>
<td>Computer used for the execution of programs</td>
</tr>
<tr>
<td>Partition</td>
<td>Group of nodes into logical sets</td>
</tr>
<tr>
<td>Job</td>
<td>Allocation of resources assigned to a user for some time</td>
</tr>
<tr>
<td>Job Step</td>
<td>Sets of (possible parallel) tasks with a job</td>
</tr>
</tbody>
</table>

![Diagram showing SLURM Entities]
Nodes hierarchy

- NUMA [base]board
  - Socket/Core/Thread
- Memory
- Generic Resources GRES (e.g. GPUs)
Users submit jobs to a partition (queue)
Backup slides

SLURM Entities example

Partition “batch”

Job 1
Job 2
Job 3

Node: tux123
Socket 0

Core 1
Core 2
Core 3
Core 4
Core 5
Core 6

Job allocation

Jobs are allocated resources
Backup slides

SLURM Entities example

- Jobs spawn steps, which are allocated resources from within the job’s allocation

```
#!/bin/bash
srun -n4 --exclusive a.out &
srun -n2 --exclusive a.out &
wait
```

Node: tux123
Socket 0

Step 0
Step 1

Partition “batch”

Job 1
Job 2
Job 3
Node State Information

- NUMA boards, Sockets, Cores, Threads
- CPUs
  - can treat each core or each thread as a CPU for scheduling purposes
- Memory size
- Temporary disk space
- Features (arbitrary string, e.g. OS version)
- Weight (scheduling priority, ...)
  - can favor least capable node that satisfies job requirement
- Boot time
- CPU Load
- State (e.g. drain, down, etc.)
- Reason, time and user ID
  - e.g. “Bad PDU [operator@12:40:10T12/20/2011]"
Queues/Partitions State Information

- Associated with specific set of nodes
  - Nodes can be in more than one partition (not the case in iris)
- Job size and time limits
- Access control list (by Linux group) / QoS
- Preemption rules
- State information (e.g. drain)
- Over-subscription and gang scheduling rules
Backup slides

Job State

Submission → Pending → Configuring (node booting) → Resizing → Running → Suspended → Completing → Completed (zero exit code) → Preempted → Failed (non-zero exit code) → TimeOut (time limit reached) → NodeFail
Slurm Daemons

- **slurmctld**: Central controller (typically one per cluster)
  - Optional backup with automatic fail over
  - Monitors state of resources
  - Manages job queues and Allocates resources

- **slurmd**: Compute node daemon
  - Typically one per compute node, one or more on front-end nodes
  - Launches and manages tasks
  - Small and very light-weight (low memory and CPU use)

**Common configuration file**: `/etc/slurm/slurm.conf`
- Other interesting files: `/etc/slurm/{topology, gres}.conf`

- **slurmdbd**: Database daemon (typically one per site)
  - Collects accounting information
  - Uploads configuration information (limits, fair-share, etc.)
Predefined **Queues/Partitions**:

- **batch** (Default)  
  Maximum: 30 nodes, 5 days walltime
- **interactive**  
  Maximum: 2 nodes, 4h walltime, 10 jobs
- **long**  
  Maximum: 2 nodes, 30 days walltime, 10 jobs

- Corresponding Quality of Service (QOS)
- Possibility to run **besteffort** jobs via the qos-besteffort QOS
- Accounts associated to supervisor (multiple associations possible)
- Proper group/user accounting
User jobs have the following key characteristics:

- set of requested resources:
  - number of computing resources: nodes (including all their CPUs and cores) or CPUs (including all their cores) or cores
  - amount of memory: either per node or per CPU
  - (wall)time needed for the user’s tasks to complete their work

- a requested node partition (job queue)
- a requested quality of service (QoS) level which grants users specific accesses
- a requested account for accounting purposes

By default...

- users submit jobs to a particular partition, and under a particular account (pre-set per user).
Slurm Commands: General Info

- Man pages available for all commands, daemons and config. files
  - `--help` option prints brief description of all options
  - `--usage` option prints a list of the options
  - `-v | -vv | -vvv`: verbose output

- Commands can be run on any node in the cluster
- Any failure results in a non-zero exit code
- APIs make new tool development easy
  - Man pages available for all APIs
- Almost all options have two formats
  - A single letter option (e.g. `-p batch` for partition ‘batch’)
  - A verbose option (e.g. `--partition=batch`)
- Time formats: `DD-HH:MM::SS`
**User Commands: Job/step Allocation**

- **sbatch**: Submit script for later execution (batch mode)
  - allocate resources (nodes, tasks, partition, etc.)
  - Launch a script containing `srun`s for series of steps on them.

- **salloc**: Create job allocation & start a shell to use it
  - allocate resources (nodes, tasks, partition, etc.),
  - either run a command or start a shell.
  - Request launch `srun` from shell. (interactive commands within one allocation)

- **srun**: Create a job allocation (if needed) and launch a job step (typically an MPI job)
  - allocate resources (number of nodes, tasks, partition, constraints, etc.)
  - launch a job that will execute on them.

- **sattach**: attach to running job for debuggers
Backup slides

User & Admin Commands: System Information

- **sinfo**: Report system status (nodes, partitions etc.)
- **squeue**: display jobs[steps] and their state
- **scancel**: cancel a job or set of jobs.
- **scontrol**: view and/or update system, nodes, job, step, partition or reservation status
- **sstat**: show status of running jobs.
- **sacct**: display accounting information on jobs.
- **sprio**: show factors that comprise a jobs scheduling priority
- **smap**: graphically show information on jobs, nodes, partitions
  → not available on iris
Slurm Admin Commands

- **sacctmgr**: setup accounts, specify limitations on users and groups.
- **sshare**: view sharing information from multifactor plugin.
- **sreport**: display information from accounting database on jobs, users, clusters.
- **sview**: graphical view of cluster. Display and change characteristics of jobs, nodes, partitions.
  - not yet available on iris cluster
- **strigger**: show, set, clear event triggers. Events are usually system events such as an equipement failure.
# Slurm vs. OAR Main Commands

<table>
<thead>
<tr>
<th>Action</th>
<th>SLURM command</th>
<th>OAR Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Submit passive/batch job</td>
<td><code>sbatch [...] $script</code></td>
<td><code>oarsub [...] $script</code></td>
</tr>
<tr>
<td>Start interactive job</td>
<td><code>srun [...] --pty bash</code></td>
<td><code>oarsub -I [...]</code></td>
</tr>
<tr>
<td>Queue status</td>
<td><code>squeue</code></td>
<td><code>oarstat</code></td>
</tr>
<tr>
<td>User job status</td>
<td><code>squeue -u $user</code></td>
<td><code>oarstat -u $user</code></td>
</tr>
<tr>
<td>Specific job status (detailed)</td>
<td><code>scontrol show job $jobid</code></td>
<td><code>oarstat -f -j $jobid</code></td>
</tr>
<tr>
<td>Job accounting status (detailed)</td>
<td><code>sacct --job $jobid -l</code></td>
<td><code>oardel $jobid</code></td>
</tr>
<tr>
<td>Delete (running/waiting) job</td>
<td><strong><code>scancel $jobid</code></strong></td>
<td><code>oarhold $jobid</code></td>
</tr>
<tr>
<td>Hold job</td>
<td><code>scontrol hold $jobid</code></td>
<td><code>oarresume $jobid</code></td>
</tr>
<tr>
<td>Resume held job</td>
<td><code>scontrol release $jobid</code></td>
<td><code>oarnodes</code></td>
</tr>
<tr>
<td>Node list and their properties</td>
<td><code>scontrol show nodes</code></td>
<td></td>
</tr>
</tbody>
</table>
## Job Specifications

<table>
<thead>
<tr>
<th>Specification</th>
<th>SLURM</th>
<th>OAR</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Script directive</strong></td>
<td>#SBATCH</td>
<td>#OAR</td>
</tr>
<tr>
<td><code>&lt;n&gt;</code> <strong>Nodes request</strong></td>
<td><code>-N &lt;n&gt;</code></td>
<td><code>-l nodes=&lt;n&gt;</code></td>
</tr>
<tr>
<td><code>&lt;n&gt;</code> <strong>Cores/Tasks request</strong></td>
<td><code>-n &lt;n&gt;</code></td>
<td><code>-l core=&lt;n&gt;</code></td>
</tr>
<tr>
<td><code>&lt;c&gt;</code> <strong>Cores-per-node request</strong></td>
<td><code>--ntasks-per-node=&lt;c&gt;</code></td>
<td><code>-l nodes=&lt;n&gt;/core=&lt;c&gt;</code></td>
</tr>
<tr>
<td><code>&lt;c&gt;</code> <strong>Cores-per-task request (multithreading)</strong></td>
<td><code>-c=&lt;c&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;m&gt;</code> <strong>GB memory per node request</strong></td>
<td><code>--mem=&lt;m&gt;GB</code></td>
<td><code>-l walltime=hh[:mm:ss]</code></td>
</tr>
<tr>
<td><strong>Walltime request</strong></td>
<td><code>-t &lt;mm&gt;/days-hh[:mm:ss]</code></td>
<td><code>--array &lt;count&gt;</code></td>
</tr>
<tr>
<td><strong>Job array</strong></td>
<td><code>--array &lt;specification&gt;</code></td>
<td></td>
</tr>
<tr>
<td><strong>Job name</strong></td>
<td><code>-J &lt;name&gt;</code></td>
<td><code>-n &lt;name&gt;</code></td>
</tr>
<tr>
<td><strong>Job dependency</strong></td>
<td><code>-d &lt;specification&gt;</code></td>
<td><code>-a &lt;jobid&gt;</code></td>
</tr>
<tr>
<td><strong>Property request</strong></td>
<td><code>-C &lt;specification&gt;</code></td>
<td><code>-p &quot;&lt;property&gt;=&lt;value&gt;&quot;</code></td>
</tr>
<tr>
<td><strong>Specify job partition/queue</strong></td>
<td><code>-p &lt;partition&gt;</code></td>
<td><code>-t &lt;queue&gt;</code></td>
</tr>
<tr>
<td><strong>Specify job qos</strong></td>
<td><code>--qos &lt;qos&gt;</code></td>
<td></td>
</tr>
<tr>
<td><strong>Specify account</strong></td>
<td><code>-A &lt;account&gt;</code></td>
<td></td>
</tr>
<tr>
<td><strong>Specify email address</strong></td>
<td><code>--mail-user=&lt;email&gt;</code></td>
<td><code>--notify &quot;mail:&lt;email&gt;&quot;</code></td>
</tr>
</tbody>
</table>
Backup slides

Typical Workflow

# Run an interactive job -- make an alias ‘si [...]’

```
$> srun -p interactive --qos qos-interactive --pty bash
```

# Ex: interactive job for 30 minutes, with 2 nodes/4 tasks per node

```
$> si --time=0:30:0 -N 2 --ntasks-per-node=4
```

# Run a [passive] batch job -- make an alias ‘sb [...]’

```
$> sbatch -p batch --qos qos-batch /path/to/launcher.sh
```

# Will create (by default) slurm-<jobid>.out file

<table>
<thead>
<tr>
<th>Environment variable</th>
<th>SLURM</th>
<th>OAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job ID</td>
<td>$SLURM_JOB_ID</td>
<td>$OAR_JOB_ID</td>
</tr>
<tr>
<td>Resource list</td>
<td>$SLURM_NODELIST #List not file!</td>
<td>$OAR_NODEFILE</td>
</tr>
<tr>
<td>Job name</td>
<td>$SLURM_JOB_NAME</td>
<td>$OAR_JOB_NAME</td>
</tr>
<tr>
<td>Submitting user name</td>
<td>$SLURM_JOB_USER</td>
<td>$OAR_USER</td>
</tr>
<tr>
<td>Task ID within job array</td>
<td>$SLURM_ARRAY_TASK_ID</td>
<td>$OAR_ARRAY_INDEX</td>
</tr>
<tr>
<td>Working directory at submission</td>
<td>$SLURM_SUBMIT_DIR</td>
<td>$OAR_WORKING_DIRECTORY</td>
</tr>
<tr>
<td>Number of nodes assigned to the job</td>
<td>$SLURM_NNODES</td>
<td>$(wc -l ${OAR_NODEFILE})</td>
</tr>
<tr>
<td>Number of tasks of the job</td>
<td>$SLURM_NTASKS</td>
<td>$(wc -l ${OAR_NODEFILE})</td>
</tr>
</tbody>
</table>

**Note:** create the equivalent of `$OAR_NODEFILE` in Slurm:

```
<-- srun hostname | sort -n > hostfile
```
Available Node partitions

- **Slurm Command Option**: `-p, --partition=<partition>`
  - Example: `{srun,sbatch} -p batch [...]`
- **Date format**: `-t <minutes>` or `-t <D>-<H>:<M>:<S>`

<table>
<thead>
<tr>
<th>Partition</th>
<th>#Nodes</th>
<th>Default time</th>
<th>Max time</th>
<th>Max nodes/user</th>
</tr>
</thead>
<tbody>
<tr>
<td>batch</td>
<td>80%</td>
<td>0-2:0:0 [2h]</td>
<td>5-0:0:0 [5d]</td>
<td>unlimited</td>
</tr>
<tr>
<td>interactive</td>
<td>10%</td>
<td>0-1:0:0 [1h]</td>
<td>0-4:0:0 [4h]</td>
<td>2</td>
</tr>
<tr>
<td>long</td>
<td>10%</td>
<td>0-2:0:0 [2h]</td>
<td>30-0:0:0 [30d]</td>
<td>2</td>
</tr>
</tbody>
</table>
Quality of Service (QOS)

- Slurm Command Option  
  ```bash
  --qos=<qos>
  ```
- There is **no default QOS** (due to the selected scheduling model)
  - you **MUST** provide upon **any job submission**
  - a default qos is guessed from the partition i.e. `qos-<partition>`

<table>
<thead>
<tr>
<th>QoS</th>
<th>User group</th>
<th>Max cores</th>
<th>Max jobs/user</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>qos-besteffort</td>
<td>ALL</td>
<td>no limit</td>
<td></td>
<td>Preemptible jobs, requeued on preemption</td>
</tr>
<tr>
<td>qos-batch</td>
<td>ALL</td>
<td>1064</td>
<td>100</td>
<td>Normal usage of the batch partition</td>
</tr>
<tr>
<td>qos-interactive</td>
<td>ALL</td>
<td>224</td>
<td>10</td>
<td>Normal usage of the interactive partition</td>
</tr>
<tr>
<td>qos-long</td>
<td>ALL</td>
<td>224</td>
<td>10</td>
<td>Normal usage of the long partition</td>
</tr>
<tr>
<td>qos-batch-rsvd</td>
<td>rsvd</td>
<td>rsvd</td>
<td>100</td>
<td>Reserved usage of the batch partition</td>
</tr>
<tr>
<td>qos-interactive-rsvd</td>
<td>rsvd</td>
<td>rsvd</td>
<td>10</td>
<td>Reserved usage of the interactive partition</td>
</tr>
<tr>
<td>qos-long-rsvd</td>
<td>rsvd</td>
<td>rsvd</td>
<td>10</td>
<td>Reserved usage of the long partition</td>
</tr>
</tbody>
</table>
Accounts

- Every user job runs under a group account
  granting access to specific QOS levels.

<table>
<thead>
<tr>
<th>Account</th>
<th>Parent Account</th>
</tr>
</thead>
<tbody>
<tr>
<td>UL</td>
<td>UL</td>
</tr>
<tr>
<td>FSTC</td>
<td>UL</td>
</tr>
<tr>
<td>FDEF</td>
<td>UL</td>
</tr>
<tr>
<td>FLSHASE</td>
<td>UL</td>
</tr>
<tr>
<td>LCSB</td>
<td>UL</td>
</tr>
<tr>
<td>SNT</td>
<td>UL</td>
</tr>
<tr>
<td>Professor $X</td>
<td>FACULTY / IC</td>
</tr>
<tr>
<td>Group head $G</td>
<td>FACULTY / IC</td>
</tr>
<tr>
<td>Researcher $R</td>
<td>Professor $X</td>
</tr>
<tr>
<td>Researcher $R</td>
<td>Group head $G</td>
</tr>
<tr>
<td>Student $S</td>
<td>Professor $X</td>
</tr>
<tr>
<td>Student $S</td>
<td>Group head $G</td>
</tr>
<tr>
<td>External collaborator $E</td>
<td>Professor $X</td>
</tr>
<tr>
<td>External collaborator $E</td>
<td>Group head $G</td>
</tr>
</tbody>
</table>

```
$> sacctmgr list associations where users=$USER \
   format=Account%30s,User,Partition,QOS
```
Other Features

- **Checkpoint / Restart**
  - Based on DMTCP: Distributed MultiThreaded CheckPointing
  - see the official DMTCP launchers
  - ULHPC example

- Many metrics can be extracted from user jobs
  - with SLURM’s own tools (sacct/sstat)
  - within the jobs with e.g. PAPI
  - easy to bind executions with Allinea Performance Report

- Advanced admission rules
  - to simplify CLI

- Container Shifter / Singularity
  - Work in progress, not yet available on iris
# Simple interactive job

(\texttt{access})\$ \texttt{srunch -p interactive [--qos qos-interactive] --pty bash}
(\texttt{node})\$ \texttt{echo } \$\texttt{SLURM\_NTASKS}
1
(\texttt{node})\$ \texttt{echo } \$\texttt{SLURM\_JOBID}
59900
# Simple interactive job

```bash
(access)\$> srun -p interactive [--qos qos-interactive] --pty bash
(node)\$> echo $SLURM_NTASKS
1
(node)\$> echo $SLURM_JOBID
59900
```

```
$> squeue -u $USER -l  # OR 'sq'
```
## Simple Example of usage

```bash
# Simple interactive job
(access)\$ \texttt{srung -p interactive [--qos qos-interactive] --pty bash}

(node)\$ \texttt{echo \$SLURM_NTASKS}
1

(node)\$ \texttt{echo \$SLURM_JOBID}
59900
```

```bash
\$ \texttt{squeue -u \$USER -l}  # OR 'sq'
```

- Many metrics during (scontrol)/after job execution (sacct)
  - including energy (J) – but with caveats
  - job steps counted individually
  - enabling advanced application debugging and optimization
- Job information available in easily parseable format (add -p/-P)
Live Job Statistics

```
$> scontrol show job 59900
JobId=59900  JobName=bash
   UserId=<login>(<uid>)  GroupId=clusterusers(666)  MCS_label=N/A
   Priority=6627  Nice=0  Account=ulhpc  QOS=qos-interactive
   JobState=RUNNING  Reason=None  Dependency=(null)
   RunTime=00:04:19  TimeLimit=01:00:00  TimeMin=N/A
   SubmitTime=2017-10-22T23:07:02  EligibleTime=2017-10-22T23:07:02
   StartTime=2017-10-22T23:07:02  EndTime=2017-10-23T00:07:02  Deadline=N/A
   PreemptTime=None  SuspendTime=None  SecsPreSuspend=0
   Partition=interactive  AllocNode:Sid=access1:72734  
   NodeList=iris-002
   NumNodes=1  NumCPUs=1  NumTasks=1  CPUs/Task=1  ReqB:S:C:T=0:0:*:*:
   TRES=cpu=1,mem=4G,node=1
   Socks/Node=*  NtasksPerN:B:S:C=0:0:*:*  CoreSpec=*  
   MinCPUsNode=1  MinMemoryCPU=4G  MinTmpDiskNode=0
   
   Command=bash
   WorkDir=/mnt/irisgpfs/users/<login>
```
### Node/Job Statistics

```
$> sinfo
PARTITION   AVAIL   TIMELIMIT   NODES   STATE   NODELIST
interactive  up      4:00:00   10       idle    iris-[001-010]
long         up      30-00:00:00 2         resv    iris-[019-020]
long         up      30-00:00:00 8         idle    iris-[011-018]
batch*       up      5-00:00:00 5         mix     iris-[055,060-062,101]
batch*       up      5-00:00:00 13        alloc   iris-[053-054,056-059,102-108]
batch*       up      5-00:00:00 70        idle    iris-[021-052,063-100]
```
### Node/Job Statistics

**sinfo**

<table>
<thead>
<tr>
<th>PARTITION</th>
<th>AVAIL</th>
<th>TIMELIMIT</th>
<th>NODES</th>
<th>STATE</th>
<th>NODELIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>interactive</td>
<td>up</td>
<td>4:00:00</td>
<td>10</td>
<td>idle</td>
<td>iris-[001-010]</td>
</tr>
<tr>
<td>long</td>
<td>up</td>
<td>30-00:00:0</td>
<td>2</td>
<td>resv</td>
<td>iris-[019-020]</td>
</tr>
<tr>
<td>long</td>
<td>up</td>
<td>30-00:00:0</td>
<td>8</td>
<td>idle</td>
<td>iris-[011-018]</td>
</tr>
<tr>
<td>batch*</td>
<td>up</td>
<td>5-00:00:00</td>
<td>5</td>
<td>mix</td>
<td>iris-[055,060-062,101]</td>
</tr>
<tr>
<td>batch*</td>
<td>up</td>
<td>5-00:00:00</td>
<td>13</td>
<td>alloc</td>
<td>iris-[053-054,056-059,102-108]</td>
</tr>
<tr>
<td>batch*</td>
<td>up</td>
<td>5-00:00:00</td>
<td>70</td>
<td>idle</td>
<td>iris-[021-052,063-100]</td>
</tr>
</tbody>
</table>

**sacct**

```bash
$> sacct --format=account,user,jobid,jobname,partition,state -j <JOBID>

$> sacct --format=elapsed,elapsedraw,start,end -j <JOBID>

$> sacct --format=maxrss,maxvmsize,consumedenergy,consumedenergyraw,nodelist -j <JOBID>
```
Playing with hostname and task ID label

```bash
$> srun [-N #node] [-n #task] [--ntasks-per-node #n] [] CMD
```

# -n: #tasks

```bash
$> srun -n 4 -l hostname
1: iris-055
2: iris-055
3: iris-055
0: iris-055
```
$> srun [-N #node] [-n #task] [--ntasks-per-node #n] [] CMD

# -n: #tasks
$> srun -n 4 -l hostname
1: iris-055
2: iris-055
3: iris-055
0: iris-055

# -N: #nodes
$> srun -N 4 -l hostname
3: iris-058
2: iris-057
1: iris-056
0: iris-055
Playing with hostname and task ID label

```bash
$> srun [-N #node] [-n #task] [--ntasks-per-node #n] [] CMD
```

- **# -n**: #tasks
  ```bash
  $> srun -n 4 -l hostname
  1: iris-055
  2: iris-055
  3: iris-055
  0: iris-055
  ```

- **# -c**: #cpus/task ~#thread/task
  ```bash
  $> srun -c 4 -l hostname
  0: iris-055
  ```

- **# -N**: #nodes
  ```bash
  $> srun -N 4 -l hostname
  3: iris-058
  2: iris-057
  1: iris-056
  0: iris-055
  ```
Playing with hostname and task ID label

\[ srun [-N \#node] [-n \#task] [--ntasks-per-node \#n] [\textbf{\texttt{CMD}}] \]

\# -n: \#tasks
\[
\texttt{\$> srun -n 4 -l hostname} \\
1: iris-055 \\
2: iris-055 \\
3: iris-055 \\
0: iris-055
\]

\# -c: \#cpus/task \#thread/task
\[
\texttt{\$> srun -c 4 -l hostname} \\
0: iris-055
\]

\# -N: \#nodes
\[
\texttt{\$> srun -N 4 -l hostname} \\
3: iris-058 \\
2: iris-057 \\
1: iris-056 \\
0: iris-055
\]

\[
\texttt{\$> srun -N 2 -n 4 -l hostname} \\
3: iris-056 \\
0: iris-055 \\
1: iris-055 \\
2: iris-055 \\
\texttt{\$> srun -N 2 --ntasks-per-node 2 -l hostname} \\
3: iris-056 \\
2: iris-056 \\
1: iris-055 \\
0: iris-055
\]
Backup slides

Job submission with salloc

```bash
$> salloc [-N #node] [-n #task] [--ntasks-per-node #n]
```

```
$> salloc -N 4
salloc: Granted job allocation 59955
salloc: Waiting for resource configuration
salloc: Nodes iris-[055,060-062] are ready for job
$> env | grep SLURM
$> hostname
access1.iris-cluster.uni.lux
$> srun -l hostname
0: iris-055
2: iris-061
1: iris-060
3: iris-062
```
Reservations and scontrol features

$>\text{scontrol} \text{ show job <JOBID>}$

# Job info
Backup slides

Reservations and scontrol features

$> scontrol show job <JOBID>          # Job info

$> scontrol show {partition,topology}
Reservations and scontrol features

- `$> scontrol show job <JOBID>`  
  # Job info

- `$> scontrol show {partition,topology}`

- `$> scontrol show reservations`  
  # Show existing reservations

- `$> scontrol create reservation`  
  ReservationName=<name>  
  accounts=<account_list>  
  corecnt=<num>  
  duration=[days]-hours:minutes:seconds  
  endtime=yyyy-mm-dd[thh:mm[:ss]]  
  features=<feature_list>  
  flags=maint,overlap,ignore_jobs,daily,weekly  
  licenses=<license>  
  nodecnt=<count>  
  nodes=<node_list>  
  partitionname=<partition(s)>  
  starttime=yyyy-mm-dd[thh:mm[:ss]]  
  users=<user_list>  
  endtime=yyyy-mm-dd[thh:mm[:ss]]`
Basic Slurm Launcher Examples

Documentation

https://hpc.uni.lu/users/docs/slurm_launchers.html

See also **PS1**, **PS2** and **PS3**

```bash
#!/bin/bash -l
# Request one core for 5 minutes in the batch queue

#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --time=0-00:05:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

[...]
#!/bin/bash -l
# Request two cores on each of two nodes for 3 hours

#SBATCH -N 2
#SBATCH --ntasks-per-node=2
#SBATCH --time=00:03:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

echo "== Starting run at $(date)"
echo "== Job ID: ${SLURM_JOBID}"  
echo "== Node list: ${SLURM_NODELIST}"  
echo "== Submit dir. : ${SLURM_SUBMIT_DIR}"
#!/bin/bash -l
# Request one core and half the memory available on an iris cluster node for one day
#
#SBATCH -J MyLargeMemorySequentialJob
#SBATCH --mail-type=end,fail
#SBATCH --mail-user=Your.Email@Address.lu
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --mem=64GB
#SBATCH --time=1-00:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

echo "== Starting run at $(date)"
echo "== Job ID: ${SLURM_JOBID}"
echo "== Node list: ${SLURM_NODELIST}"
echo "== Submit dir. : ${SLURM_SUBMIT_DIR}"
pthreads/OpenMP Slurm Launcher

```bash
#!/bin/bash -l
# Single node, threaded (pthreads/OpenMP) application launcher,
# using all 28 cores of an iris cluster node:

#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH -c 28
#SBATCH --time=0-01:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}
/path/to/your/threaded.app
```
#!/bin/bash -l
# Single node, multi-core parallel application (MATLAB, Python, R...) 
# launcher, using all 28 cores of an iris cluster node:

#SBATCH -N 1
#SBATCH --ntasks-per-node=28
#SBATCH -c 1
#SBATCH --time=0-01:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

module load base/MATLAB
matlab -nodisplay -nosplash < /path/to/inputfile > /path/to/outputfile
Official SLURM guide for Intel MPI

```bash
#!/bin/bash -l
# Multi-node parallel application IntelMPI launcher,
# using 128 distributed cores:

#SBATCH -n 128
#SBATCH -c 1
#SBATCH --time=0-01:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

module load toolchain/intel
export I_MPI_PMI_LIBRARY=/usr/lib64/libpmi.so
srun -n $SLURM_NTASKS /path/to/your/intel-toolchain-compiled-application
```
OpenMPI Slurm Launchers

- Official SLURM guide for Open MPI

```bash
#!/bin/bash -l
# Multi-node parallel application openMPI launcher,
# using 128 distributed cores:

#SBATCH -n 128
#SBATCH -c 1
#SBATCH --time=0-01:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

module load toolchain/foss
mpirun -n $SLURM_NTASKS /path/to/your/foss-toolchain-compiled-application
```
#!/bin/bash -l
# Multi-node hybrid application IntelMPI+OpenMP launcher,  
# using 28 threads per node on 10 nodes (280 cores):

#SBATCH -N 10
#SBATCH --ntasks-per-node=1
#SBATCH -c 28
#SBATCH --time=0-01:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch

module load toolchain/intel
export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}
export I_MPI_PMI_LIBRARY=/usr/lib64/libpmi.so
srun -n $SLURM_NTASKS /path/to/your/parallel-hybrid-app
**Typical Workflow on UL HPC resources**

**Preliminary setup**

1. Connect to the frontend
   - ssh, screen
2. Synchronize your code
   - scp/rsync/svn/git
3. Reserve a few interactive resources
   - oarsub -I [...]
   - on iris: srun -p interactive [...]
   - gcc/icc/mpicc/nvcc...
   - mpirun/srun/python/sh...
   - <launcher>.{sh,py}
4. (eventually) build your program
5. Test on small size problem
6. Prepare a launcher script
Typical Workflow on UL HPC resources

**Preliminary setup**

1. **Connect to the frontend**
   - `ssh`, `screen`
2. **Synchronize your code**
   - `scp/rsync/svn/git`
3. **Reserve a few interactive resources**
   - `oarsub -I [...]`
   - or, `srun -p interactive [...]`
   - ✓ *(eventually)* build your program
   - ✓ Test on small size problem
   - ✓ Prepare a launcher script

**Real Experiment**

1. **Reserve passive resources**
   - `oarsub [...] <launcher>`
   - or, on iris: `sbatch -p {batch|long} [...] <launcher>`
2. **Grab the results**
   - `scp/rsync/svn/git ...`