Latest versions available on Github:

UL HPC tutorials:  https://github.com/ULHPC/tutorials
UL HPC School:  http://hpc.uni.lu/hpc-school/
Summary

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
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**:
  - a set of **requested resources**:
    - number of computing resources: **nodes** (including all their CPUs and cores) or **CPUs** (including all their cores) or **cores**
    - number of accelerators (GPUs)
    - amount of **memory**: either per node or per (logical) CPU
    - the **(wall)time** needed for the user’s tasks to complete their work
  - a set of **constraints** limiting jobs to nodes with specific features
  - 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 [...]

```
(access)$ srun -p interactive --qos qos --interactive --pty bash -i
(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
```

```
$ 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 workload manager

#### 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>152</td>
<td>0-2:0:0</td>
<td>5-0:0:0</td>
<td>unlimited</td>
</tr>
<tr>
<td>bigmem</td>
<td>4</td>
<td>0-2:0:0</td>
<td>5-0:0:0</td>
<td>unlimited</td>
</tr>
<tr>
<td>gpu</td>
<td>24</td>
<td>0-2:0:0</td>
<td>5-0:0:0</td>
<td>unlimited</td>
</tr>
<tr>
<td>interactive</td>
<td>8</td>
<td>0-1:0:0</td>
<td>0-4:0:0</td>
<td>2</td>
</tr>
<tr>
<td>long</td>
<td>8</td>
<td>0-2:0:0</td>
<td>30-0:0:0</td>
<td>2</td>
</tr>
</tbody>
</table>
### SLURM workload manager

#### 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>152</td>
<td>0-2:0:0</td>
<td>5-0:0:0</td>
<td>unlimited</td>
</tr>
<tr>
<td>bigmem</td>
<td>4</td>
<td>0-2:0:0</td>
<td>5-0:0:0</td>
<td>unlimited</td>
</tr>
<tr>
<td>gpu</td>
<td>24</td>
<td>0-2:0:0</td>
<td>5-0:0:0</td>
<td>unlimited</td>
</tr>
<tr>
<td>interactive</td>
<td>8</td>
<td>0-1:0:0</td>
<td>0-4:0:0</td>
<td>2</td>
</tr>
<tr>
<td>long</td>
<td>8</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>Max cores</th>
<th>Max jobs/user</th>
</tr>
</thead>
<tbody>
<tr>
<td>qos-besteffort</td>
<td>no limit</td>
<td></td>
</tr>
<tr>
<td>qos-batch</td>
<td>2344</td>
<td>100</td>
</tr>
<tr>
<td>qos-bigmem</td>
<td>no limit</td>
<td>10</td>
</tr>
<tr>
<td>qos-gpu</td>
<td>no limit</td>
<td>10</td>
</tr>
<tr>
<td>qos-interactive</td>
<td>168</td>
<td>10</td>
</tr>
<tr>
<td>qos-long</td>
<td>168</td>
<td>10</td>
</tr>
</tbody>
</table>
You have some **private** QoS not accessible to all users.

<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>2344</td>
<td>100</td>
</tr>
<tr>
<td>qos-batch-001</td>
<td>private</td>
<td>1400</td>
<td>100</td>
</tr>
<tr>
<td>qos-batch-002</td>
<td>private</td>
<td>256</td>
<td>100</td>
</tr>
<tr>
<td>qos-batch-003</td>
<td>private</td>
<td>256</td>
<td>100</td>
</tr>
<tr>
<td>qos-bigmem</td>
<td>ALL</td>
<td>no limit</td>
<td>10</td>
</tr>
<tr>
<td>qos-gpu</td>
<td>ALL</td>
<td>no limit</td>
<td>10</td>
</tr>
<tr>
<td>qos-interactive</td>
<td>ALL</td>
<td>168</td>
<td>10</td>
</tr>
<tr>
<td>qos-interactive-001</td>
<td>private</td>
<td>56</td>
<td>10</td>
</tr>
<tr>
<td>qos-long</td>
<td>ALL</td>
<td>168</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
  → shorter walltime jobs highly encouraged
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
  → jobs may wait in the queue with QOS*Limit reason set
    ✓ e.g. QOSGrpCpuLimit if group limit for CPUs was reached
Default partition: **batch**, meant to receive most user jobs
   - we hope to see majority of user jobs being able to scale
   - shorter walltime jobs highly encouraged

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
   - jobs may wait in the queue with *QOS*\(^*\)\textit{Limit} reason set
     - e.g. **QOSGrpCpuLimit** if group limit for CPUs was reached

Preemptible **besteffort** **QOS** available for **batch** and **interactive** partitions (but not **yet** for **bigmem**, **gpu** or **long**)
   - meant to ensure maximum resource utilization especially on **batch**
   - should be used together with restartable software

**QOSs** specific to particular group accounts exist (discussed later)
   - granting additional accesses to platform contributors
SLURM workload manager

SLURM - design for iris (IV)

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

- Resource selection: **consumable resources**
  - cores and memory as consumable (per-core scheduling)
  - GPUs as consumable (4 GPUs per node in the gpu partition)
  - block distribution for cores (best-fit algorithm)
  - default memory/core: 4GB (4.1GB maximum, rest is for OS)
    ✓ gpu and bigmem partitions: 27GB maximum
SLURM workload manager

SLURM - design for iris (IV)

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

- Resource selection: **consumable resources**
  - cores and memory as consumable (per-core scheduling)
  - GPUs as consumable (4 GPUs per node in the gpu partition)
  - block distribution for cores (best-fit algorithm)
  - default memory/core: 4GB (4.1GB maximum, rest is for OS)
    ✓ gpu and bigmem partitions: 27GB maximum

- 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 workload manager

SLURM - design for iris (IV)

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

- Resource selection: **consumable**
  - cores and memory as consumable (per-core scheduling)
  - GPUs as consumable (4 GPUs per node in the gpu partition)
  - block distribution (best-fit algorithm)
  - default memory: 4GB (4.1GB maximum, rest is for OS)
    - ✓ gpu and bigmem partitions: 27GB maximum

- User process tracking with **cgroups**
  - cpuset 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 =
(PriorityWeightAge) * (age_factor) +
(PriorityWeightFairshare) * (fair-share_factor) +
(PriorityWeightJobSize) * (job_size_factor) +
(PriorityWeightPartition) * (partition_factor) +
(PriorityWeightQOS) * (QOS_factor) +
SUM(TRES_weight_cpu * TRES_factor_cpu,
   TRES_weight_<type> * TRES_factor_<type>,
   ...)

For complete details see: slurm.schedmd.com/priority_multifactor.html

- **TRES** - Trackable RESources
  ↦ CPU, Energy, Memory and Node tracked by default

- **GRES** - Generic RESources
  ↦ GPU

Corresponding weights/reset periods **tuned with your feedback**
  ↦ we require (your & your group’s) usage pattern to optimize them
  ↦ the target is high interactivity (low time spent by the jobs waiting)
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 disabled 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 disabled 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
  ✓ (at minimum:) replace `mpirun` with SLURM’s `srun`
  ✓ if you compile/install your own MPI you’ll need to configure it

→ Many examples at: [https://hpc.uni.lu/users/docs/slurm_launchers.html](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
  - (at minimum:) replace mpirun with SLURM’s srun
  - if you compile/install your own MPI you’ll need to configure it

Many examples at: 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
ULHPC customization 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/$jobid.$rstcnt/[tmp,var_tmp]
  - transparent for apps. ran through srun
  - apps. ran with ssh cannot be attached, will see base /tmp!

- **X11 forwarding (GUI applications)**
  - Some issue prevents us to use –x11 option of SLURM on iris
    - workaround in the tutorial/FAQ
    - create job using salloc and then use ssh -Y
Software licenses in SLURM

- ARM (ex. 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`

```
$> 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
  \( \rightarrow \) with their own name: Name.Surname

All user accounts linked to a bank account
  \( \rightarrow \) including Profs.’s own user

Iris accounting DB contains over
  \( \rightarrow \) 103 group accounts from Faculties, ICs & Externals
  \( \rightarrow \) comprising 877 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 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

- `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
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

- **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 (own) jobs 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>Job efficiency report</td>
<td>seff $jobid</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>
<tr>
<td>Attach to running job</td>
<td>sjoin $jobid [$node]</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 $m memory per node</td>
<td>--mem=$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 $g # of GPUs per node</td>
<td>--gres=gpu:$g</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 required node $feature</td>
<td>-C $feature</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 workload manager

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 $g # of GPUs per node</td>
<td>--gres=gpu:$g</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 required node $feature</td>
<td>-C $feature</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>

- 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
  - beware of running across heterogeneous nodes: use ‘-C’
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><code>-d after:$jobid1:$jobid2</code></td>
</tr>
<tr>
<td>these other jobs have ended</td>
<td><code>-d afterany:$jobid1:$jobid2</code></td>
</tr>
<tr>
<td>these other jobs have ended with no errors</td>
<td><code>-d afterok:$jobid1:$jobid2</code></td>
</tr>
<tr>
<td>these other jobs have ended with errors</td>
<td><code>-d afternok:$jobid1:$jobid2</code></td>
</tr>
<tr>
<td>all other jobs with the same name have ended</td>
<td><code>-d singleton</code></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 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

<table>
<thead>
<tr>
<th>Other scheduling requests</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
salloc -p interactive --qos qos-interactive bash -c 'ssh -Y $(scontrol show hostnames | head -n 1)'
srun -p interactive --qos qos-besteffort --cpu-bind=none -N1 -n4 --pty bash -i
srunc -C skylake -p batch --time=0:10:0 -N1 -c28 --pty bash -i
```
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
salloc -p interactive --qos qos-interactive bash -c 'ssh -Y $(scontrol show hostnames | head -n 1)'
srun -p interactive --qos qos-besteffort --cpu-bind=none -N1 -n4 --pty bash -i
srun -C skylake -p batch --time=0:10:0 -N1 -c28 --pty bash -i
```
> **Interactive jobs**

```bash
srun -p interactive --qos qos-interactive --time=0:30 -N2 --ntasks-per-node=4 --pty bash -i
salloc -p interactive --qos qos-interactive bash -c 'ssh -Y $(scontrol show hostnames | head -n 1)'
```
```
srun -p interactive --qos qos-besteffort --cpu-bind=none -N1 -n4 --pty bash -i
srun -C skylake -p batch --time=0:10:0 -N1 -c28 --pty bash -i
```

> **Batch jobs**

```bash
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=2019-11-23T07:30:00 job.sh
sbatch -p batch --qos qos-besteffort job.sh
sbatch -p gpu --qos qos-gpu --gres=gpu:4 job.sh
sbatch -p bigmem --qos qos-bigmem --mem=2T job.sh
```

**Status and details for partitions, nodes, reservations**

```bash
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
```
Usage examples (II)

Collecting job information, priority, expected start time

scontrol show job $jobid
sprio -l
squeue --start -u $USER

# only available while job is queued + 5 minutes after completion
Collecting job information, priority, expected start time

- `scontrol show job $jobid`
- `sprio -l`
- `squeue --start -u $USER`

# only available while job is queued + 5 minutes after completion

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`
- `sprio -l`
- `squeue --start -u $USER`

# only available while job is queued + 5 minutes after completion

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 & seff tools

- `sacct -j $jobid / sacct -j $jobid -l`
- `sacct -p -j $jobid --format=account,user,jobid,jobname,partition,state,elapsed,elapseddraw, start,end,maxrss,maxvmsize,consumedenergy,consumedenergyraw,nnodes,ncpus,nodelist`
- `sacct --starttime 2018-11-23 -u $USER`
- `seff $jobid`

# very useful to see at a glance: CPU/memory efficiency and max. memory
### Controlling queued and running jobs

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>scontrol hold $jobid</code></td>
<td>Hold the job</td>
</tr>
<tr>
<td><code>scontrol release $jobid</code></td>
<td>Release the job</td>
</tr>
<tr>
<td><code>scontrol suspend $jobid</code></td>
<td>Suspend the job</td>
</tr>
<tr>
<td><code>scontrol resume $jobid</code></td>
<td>Resume the job</td>
</tr>
<tr>
<td><code>scancel $jobid</code></td>
<td>Cancel the job</td>
</tr>
<tr>
<td><code>scancel -n $jobname</code></td>
<td>Cancel job by name</td>
</tr>
<tr>
<td><code>scancel -u $USER</code></td>
<td>Cancel job by user</td>
</tr>
<tr>
<td><code>scancel -u $USER -p batch</code></td>
<td>Cancel job by user in batch</td>
</tr>
<tr>
<td><code>scontrol requeue $jobid</code></td>
<td>Requeue the job</td>
</tr>
</tbody>
</table>

*Example:*

```bash
scontrol hold $jobid
scontrol release $jobid
```
Controlling queued and running jobs

\texttt{scontrol hold $jobid}
\texttt{scontrol release $jobid}
\texttt{scontrol suspend $jobid}
\texttt{scontrol resume $jobid}
\texttt{scancel $jobid}
\texttt{scancel -n $jobname}
\texttt{scancel -u $USER}
\texttt{scancel -u $USER -p batch}
\texttt{scontrol requeue $jobid}

Checking accounting links and QOS available for you

\texttt{sacctmgr show user $USER format=\texttt{user}20s,\texttt{defaultaccount}30s}
\texttt{sacctmgr list association where users=$USER format=account30s,\texttt{user}20s,qos120s}
### 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

#SBATCH --time=00:03:00
#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}"

export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}

srun ./preprocess_app
srun ./main_app

{Submit it overriding some settings: sbatch --time=5:0:0 launcher.sh}
#!/bin/bash -l
#SBATCH -J MyTestJob
#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)"
echo "== Job ID: ${SLURM_JOBID}"
echo "== Node list: ${SLURM_NODELIST}"
echo "== Submit dir. : ${SLURM_SUBMIT_DIR}"
# Your more useful application can be started below!

Use "mem" to request (more) memory per node for low #core jobs
#!/bin/bash -l
#SBATCH -J MyVeryLargeMemoryJob
#SBATCH --mail-type=end,fail
#SBATCH --mail-user=Your.Email@Address.lu
#SBATCH -N 1
#SBATCH -c 64
#SBATCH --mem=2TB
#SBATCH --time=1-00:00:00
#SBATCH -p bigmem
#SBATCH --qos=qos-bigmem

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!

Iris compute nodes in the bigmem partition have 112C/3TB RAM.
#!/bin/bash -l
#SBATCH -J MyJobOnSkylakeCPUs
#SBATCH --mail-type=end,fail
#SBATCH --mail-user=Your.Email@Address.lu
#SBATCH --ntasks-per-node=1
#SBATCH --mem=64GB
#SBATCH --time=1-00:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch
#SBATCH -C skylake
[...]

$> sinfo --format="%N %f"
NODELIST AVAIL_FEATURES
iris-[001-108] broadwell
iris-[109-168,187-190] skylake
iris-[169-186] skylake,volta
iris-[191-196] skylake,volta,volta32
Iris compute nodes in the gpu partition have 4xVolta V100 GPUs and 768GB RAM.
```bash
#!/bin/bash -l
#SBATCH -J MyGPUJob
#SBATCH -N 1
#SBATCH -c 28
#SBATCH --gres=gpu:4
#SBATCH --time=1-0:0:0
#SBATCH -p gpu

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

# Load the Singularity HPC containers module
module load tools/Singularity
# Pull the reference TensorFlow (GPU-enabled) image from Docker hub
singularity pull docker://tensorflow/tensorflow:latest-gpu
# Run the TF container w. Singularity’s nvidia support on your own model
singularity exec --nv tensorflow-latest-gpu.simg python tf-model.py
```
#!/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!

Long walltime possible but you should not (!) rely on it.
Always prefer parallel, short walltime, requeue-able jobs.
#!/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!
System-level checkpoint-restart possible with DMTCP.
#!/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.
#SBATCH workload manager

## Job launchers - MATLAB

```
#!/bin/bash

#SBATCH -N 1
#SBATCH --ntasks-per-node=28
#SBATCH -c 1
#SBATCH --time=0-01:00:00
#SBATCH -p batch
#SBATCH --qos=qos-batch
#SBATCH --cpu-bind=none

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=00:01:00
#SBATCH -p batch
#SBATCH --qos=qos-batch
#SBATCH --cpu-bind=none

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.
As of 2019 the iris cluster is heterogeneous (Broadwell+Skylake-gen systems)

Its core networking is still non-blocking fat-tree.

- Simply requesting #tasks may not be optimal
  - from hardware POV - slight difference in CPU freq. for now
  - from software efficiency POV - best to have arch. opt. builds
- Many elements contribute to an optimal (fast!) execution:
  - correct division of tasks / cores-per-task and application launch
  - memory allocation
  - execution on nodes with GPU accel. and their allocation
- Different MPI implementations will behave differently
  - recent Intel & OpenMPI on iris
  - always prefer launch using srun
SLURM workload manager

Job launchers - IntelMPI

```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/intel

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

IntelMPI is configured to use PMI2 for process management (optimal).
Bare mpirun works but not recommended.
SLURM workload manager

Job launchers - OpenMPI

```
#!/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 works but 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 -C skylake
#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.
A note on CPU affinity

- Processes pinned by default to cores (CPUs in SLURM docs.)
- srun aware of requested tasks/cores configuration and pins processes/threads accordingly
- Many options to control task affinity exist, see:
  - https://slurm.schedmd.com/srun.html#OPT_cpu-bind
  - https://slurm.schedmd.com/srun.html#OPT_hint
- Can be disabled with srun --cpu-bind=none

If not disabled for 'interactive' jobs, all your processes will be pinned to 1st core!
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 still in use as the workload manager of Gaia and Chaos but clusters will be decommissioned by the end of this year
  - celebrating 4,506,192 jobs on Gaia! (2019-06-19)
  - celebrating 1,701,137 jobs on Chaos! (2018-06-19)

- Many of its features are common to other workload managers, incl. SLURM (462,613 jobs on Iris as of 2019-06-19)
  - 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 --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>
<tr>
<td>Join a running job</td>
<td>oarsub -C $jobid</td>
<td>sjoin $jobid [$nodeid]</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>Queue request</td>
<td>-q $queue</td>
<td>-p $partition</td>
</tr>
<tr>
<td>Nodes request</td>
<td>-l nodes=$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=$ccount, --ntasks-per-node=$ccount</td>
<td>-N $ncount --ntasks-per-node=$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=&quot;$value&quot;</td>
<td>-C $specification</td>
</tr>
<tr>
<td>Jobs on GPU nodes</td>
<td>-t gpu</td>
<td>-p gpu</td>
</tr>
<tr>
<td>Jobs on large memory nodes</td>
<td>-t bigmem</td>
<td>-p bigmem</td>
</tr>
<tr>
<td>Besteffort jobs</td>
<td>-t besteffort</td>
<td>--qos qos-besteffort</td>
</tr>
<tr>
<td>Email on job state change</td>
<td>--notify mail:$email</td>
<td>--mail-user=$email</td>
</tr>
</tbody>
</table>

Job specifications will need most adjustment on your side. Iris is more homogeneous than Gaia/Chaos for now. Running things in an optimal way is easier.
## 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><code>$OAR_JOB_ID</code></td>
<td><code>$SLURM_JOB_ID</code></td>
</tr>
<tr>
<td>Resource list</td>
<td><code>$OAR_NODEFILE</code></td>
<td><code>$SLURM_NODELIST</code> #List not file! See below.</td>
</tr>
<tr>
<td>Job name</td>
<td><code>$OAR_JOB_NAME</code></td>
<td><code>$SLURM_JOB_NAME</code></td>
</tr>
<tr>
<td>Submitting user name</td>
<td><code>$OAR_USER</code></td>
<td><code>$SLURM_JOB_USER</code></td>
</tr>
<tr>
<td>Task ID within job array</td>
<td><code>$OAR_ARRAY_INDEX</code></td>
<td><code>$SLURM_ARRAY_TASK_ID</code></td>
</tr>
<tr>
<td>Working directory at submission</td>
<td><code>$OAR_WORKING_DIRECTORY</code></td>
<td><code>$SLURM_SUBMIT_DIR</code></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

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
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...

Q&A & practical
ulhpc-tutorials.readthedocs.io/en/latest/scheduling/advanced
Thank you for your attention...

Questions?

High Performance Computing @ uni.lu

Prof. Pascal Bouvry
Dr. Sebastien Varrette
Valentin Plugaru
Sarah Peter
Hyacinthe Cartiaux
Clement Parisot
Dr. Frederic Pinel
Dr. Emmanuel Kieffer

University of Luxembourg, Belval Campus
Maison du Nombre, 4th floor
2, avenue de l’Université
L-4365 Esch-sur-Alzette
mail: hpc@uni.lu

Running jobs with SLURM

1 Introduction
2 SLURM workload manager
    SLURM concepts and design for iris
3 OAR and SLURM
4 Conclusion

http://hpc.uni.lu