Lawrence Livermore National Laboratory

SLURM Grid Ideas

2011 SLURM User Group Meeting

September 23, 2011

Don Lipari

Livermore Computing

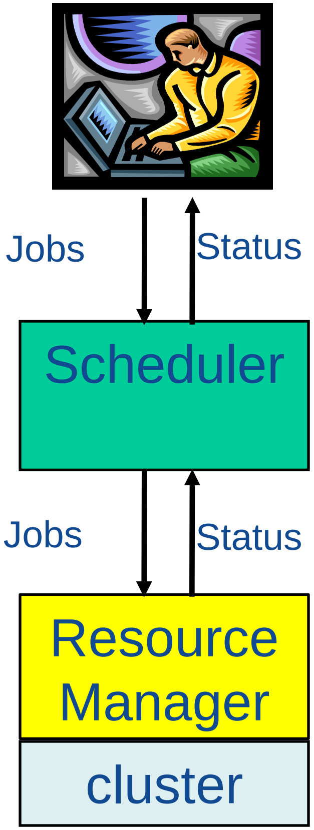
Lawrence Livermore National Laboratory, P. O. Box 808, Livermore, CA 94551

This work performed under the auspices of the U.S. Department of Energy by
Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344

LLNL-PRES-498173

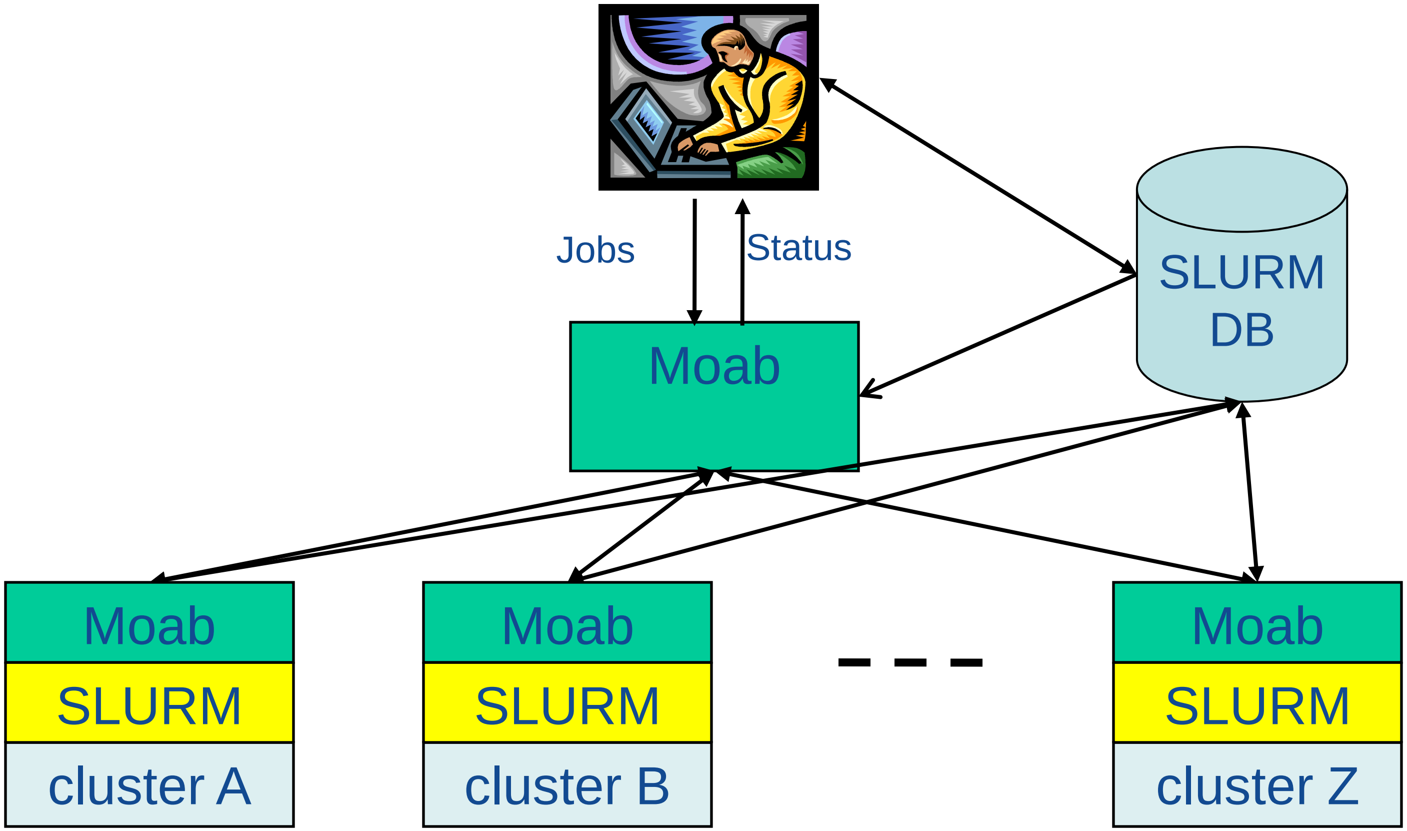
Scheduler and Resource Manager

- Determines job priorities
- Implements scheduling policy
- Enforces limits
- Workload management



- Manages the computing resources of a single cluster
- Launches job across node(s)
- Provides job status and reports resource usage

Moab Grid Scheduling



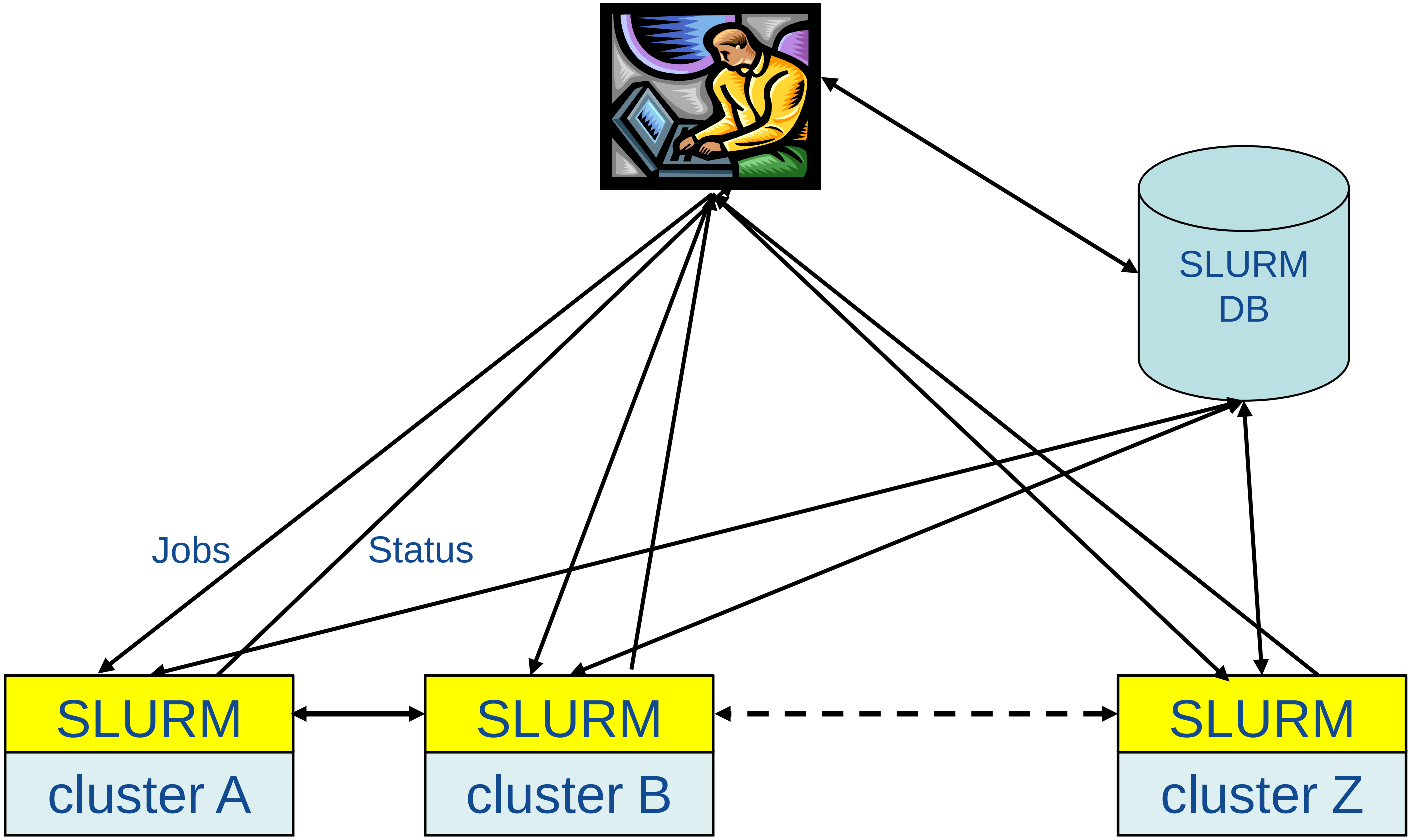
Grid Advantages

- Users can submit jobs to, and obtain status from, any cluster in the grid.
- Users can target their jobs to multiple clusters, and run on the soonest available cluster.
- Users can submit jobs that can depend on any job(s) on any cluster within the grid.
  - Includes existing support for: after, afterany, afterok, afternotok, singleton
- Job IDs are unique across the grid.
  - Users reference their job by ID.
  - They do not have to remember which machine it is on.

Grid vs. Cloud

- There are a fixed number of clusters within a grid.
- The OS and environment of the grid's clusters remains fixed over time.

SLURM Grid Scheduling



Show-Stoppers to Deploying the SLURM Grid

- There are three main deficiencies in SLURM’s v2.3 grid implementation:
  - Target clusters are selected at job submission time and cannot be changed with a dynamic workload (newly submitted jobs and jobs finishing early)
  - Does not support job dependencies off-cluster
  - Job IDs are not unique across the grid. Users are unable to status a job without knowing which cluster it is on.

Detailed Requirements

- Administrators can configure multiple clusters into a single grid.
  - By default, this includes all the clusters defined in the Accounting Storage db.
- Users can submit jobs to, and obtain status from, any cluster in the grid.
- Users can target their jobs to multiple clusters, and run on the soonest available cluster.
- If a partition is specified, only clusters containing that partition will be candidates.
- If AccountingStorageEnforce=associations is enabled, only clusters containing the requested account/user association will be candidates.
- The “soonest available” decision must be periodically re-computed to accommodate:
  - Changing workload as clusters’ queues change unpredictably (e.g., jobs finish prematurely or high priority jobs are submitted).
  - Loss of communication to other clusters.
  - Resources fail or are removed from service.

More Detailed Requirements

- Users can submit jobs that can depend on any job(s) on any cluster within the grid.
  - Includes existing support for: after, afterany, afterok, afternotok, singleton
- Job IDs targeted to multiple clusters must be unique across the grid.
  - This avoids the need for job IDs of the form cluster.jobID.
  - We need to give the user a way to status jobs by job ID and not have to hunt for a job cluster-by-cluster.
- Users must have the ability to add and remove target machines and job dependencies.
- Multi-factor job priority determination must remain cluster-specific.
- Design must avoid a single point-of-failure.
- Design must be resilient and tolerant of system or communication failures.

Enterprise Scheduling Design Option 1
Peer-to-Peer

- The slurmdbd is enhanced to dispense multi-cluster job IDs for all clusters in the grid.
  - Jobs that users submit to multiple clusters will request the next job ID from the slurmdbd.
  - Jobs that users submit to a single cluster will not request a job ID from the slurmdbd. Instead, it will receive the next job ID in that cluster’s sequence (as it does now).
  - If the slurmdbd is down, or fails to respond within a reasonable time, the job will be submitted to just the soonest target candidate (as it does now).
  - Admins will be expected to configure each cluster’s, and the slurmdbd’s, FirstJobId and MaxJobId into mutually exclusive ranges.

Peer-to-Peer (page 2)

- Jobs that users submit to multiple clusters are sent, including the script, to each cluster using the same job ID received from the slurmdbd.
- The cluster that is predicted, at the time of submission, to run the job the soonest is tagged the “leading” candidate. Job submissions to the remaining clusters are tagged, “fall-back” candidates.
  - The job on the leading candidate cluster has a list of all the fall-back candidate clusters for that job.
  - Jobs on fall-back candidate clusters have only the name of the leading candidate cluster.
  - (No resources are ever reserved for fall-back candidate jobs ?)
- If a job on the leading candidate cluster reaches the top of its queue, messages are sent to all the fall-back candidate clusters to cancel the fall-back jobs.

Peer-to-Peer (page 3)

- If a fall-back candidate job reaches the top of its queue, and the leading candidate has not run the job yet and has a predicted start time later than the fall-back candidate cluster, it can make the request to take over the leadership, (including the candidate list) and then run the job.
  - The new leading candidate cluster would then issue a message to all the other fall-backs (including the former leader) to cancel their fall-back jobs.
  - If the request for leadership is denied or goes unanswered, the fall-back candidate job is cancelled (after some configurable period).
  - If the leading candidate cluster is removed from service or becomes unresponsive, all of the fall-back candidate clusters will eventually cancel their fall-back jobs.
- Admins would be given the ability to manually transfer leading status to a fall-back cluster. This could be done in preparation for taking the leading cluster out-of-service.
- Users will have the option to delete a target cluster from their job.
  - A message would be sent to cancel the job from the removed machine, while updating the fall-back list that the leader maintains.
  - Giving users the ability to add a target cluster to their job is conceivable. The script would have to be copied over to the new target.

Enterprise Scheduling Design Option 2
Meta-Scheduling Elf

- Elf is a non-critical, independent service
- Jobs that users submit to multiple clusters will request the next job ID from the elf.
- Multi-cluster jobs are sent to the elf and the soonest candidate cluster at submit time.
- Elf periodically invokes will-run on candidate clusters to look for a cluster that can run the job sooner.
    - Must include the submit time so candidate can compute job priority as if job were submitted in the past and not now.
- If a sooner candidate is found, the elf moves the job from the original target to the sooner candidate.

Enterprise Scheduling Design Option 3
Full Blown Meta-Scheduler

- FBMS is resembles the Moab grid master
- Jobs that are submitted for multiple clusters are relayed to the FBMS where they are assigned a multi-cluster Job ID.
- FBMS maintains its own copy of the job queues of all the clusters in the enterprise.
- FBMS dispatches jobs just-in-time to target clusters.
- The FBMS takes on the scheduling of all clusters.
- The slurmctld's scheduling capability is deactivated.
- Job status now reported through the FBMS.
  - squeue reports local cluster.
  - fbms_queue reports all jobs in grid.

Job Dependencies Across Clusters

- Option A - Parent job knows dependent jobs and the condition for dependence and alerts clusters with pending dependent jobs when its condition is met.
- Option B - Dependent job periodically polls for condition of parent job
- Peer-to-peer and elf design could implement option B utilizing the Accounting Storage db or by direct communication
- FBMS would know the status of all jobs and release job dependencies when conditions were met.

Grid Options Compared

	Peer-to-Peer	Elf	FBMS
Reliability	Excellent	Excellent	Good (w/ backup design)
Complexity	slurmctld	elf + slurmctld	FBMS only
Scalability	good	better	best
Slurmctld performance penalty	fair	good	best
Scheduling	slurmctld	slurmctld	FBMS
Job Migration	All clusters have a copy of job	Elf moves jobs when needed	Dispatched to cluster at run time
Workload change	complicated	complicated	easy
User modifies candidate clusters	easy as long as a cluster is dropped	easy	easy
Job dependency	complicated	complicated	easy
Job dependency change	complicated	complicated	easy
Job ID generator	slurmdbd	Elf	FBMS

Yet Another Design Option
Ubiquitous Meta Schedulers

- A meta-scheduler daemon (umsd) to accompany every slurmctld.
- The umsd has three primary functions:
  - Selecting local, multi-cluster jobs to run once they reach to the top of the local queue.
  - Determine when job dependencies are met and release dependent jobs on the local cluster.
  - Return job information for all jobs within the grid to ms-squeue client commands.
- Requires no changes to the slurmctld's or SLURM client commands.

Ubiquitous Meta Scheduler

- All job reports (submit/run/cancel/complete) that the slurmctld sends the slurmdbd are reflected by the slurmdbd to all umsd’s.
- Multi-cluster jobs are submitted to all candidates like the peer-to-peer model.
- slurmdbd dispenses the multi-cluster job IDs.
- The local umsd detects when a multi-cluster job reaches (or nears) the top of its local queue.
- That umsd permits (and commits) the job to run on its local cluster and issues an scancel of the same job ID to all of the other clusters.
- The slurmdbd receives notice of the started job as well as the job cancellations from all the other clusters.

Ubiquitous Meta-Scheduler (cont.)

- squeue is unchanged - reports jobs from local cluster
- ms-squeue interrogates the local umsd for job reports from all clusters
- Jobs with the same ID that are candidates for multiple clusters appear as separate records in the slurmdb.
- Using its global view of all jobs running across the grid, the meta-scheduler decides when dependencies are met for jobs from local cluster and releases dependency.

umsd Option Compared

	Peer-to-Peer	Elf	FBMS	umsd
Reliability	Excellent	Excellent	Good	Excellent
Complexity	slurmctld	elf + slurmctld	FBMS only	umsd only
Scalability	good	better	best	better?
Slurmctld performance penalty	fair	good	best	best
Scheduling	slurmctld	slurmctld	FBMS	slurmctld
Job Migration	All clusters have a copy of job	Elf moves jobs when needed	Dispatched to cluster at run time	All clusters have a copy of job
Workload change	complicated	complicated	easy	easy
User modifies candidate clusters	easy as long as a cluster is dropped	easy	easy	easy as long as a cluster is dropped
Job dependency	complicated	complicated	FBMS	umsd
Job dependency change	complicated	complicated	FBMS	umsd
Job ID generator	slurmdbd	Elf	FBMS	slurmdbd