SLURM At CSCS

SLURM User Group
Barcelona, Spain
9/10-October-2012

Stephen Trofinoff
CSCS
stephen.trofinoff@cscs.ch

General SLURM Timeline

- June 2010—First working port of SLURM to a Cray
- October 2010--Install 2.x.x on Palu a development/production system
- Fall 2010-March 2011 Initial experimentation with the use of SLURM
- April 2011—CSCS goes live with SLURM on premier production system, Rosa (2.3.0-prex).
- Spring 2011-Fall 2011—CSCS begins to migrate other systems from PBS to SLURM; also ports code to Cray XK6 architecture.
- October 2011—Palu decommissioned and replaced with new XK6 named Todi—running SLURM v 2.3.0-pre5.
- November 2011—Rosa upgraded to XE6 running SLURM v. 2.3.0-pre5
- March/April 2012—Upgrade of SLURM level across most of the site to 2.3.4
- March-June 2012—Final PBS systems, Buin & Dole are replaced with Albis & Lema—both running SLURM v. 2.3.4.

General SLURM Timeline

- Spring 2012—Due to particular needs of select users, work with SchedMD to create a basic "Zero-node" job submission scheme.
- Currently, working preparing SLURM code for upcoming Cascade system due in December 2012.
- Currently, working on a basic "least-used node" selection patch for a specific set of users.

Some Past SLURM Work

- XE6 support
- XK6 support
- Zero-Node ('ZN') functionality
- GRES Accounting
- A basic SPANK auto-affinity module (based upon an older LLNL one)
- Transition procedure for preserving jobs across incompatible versions of SLURM

Current SLURM Work

- SLURM preparation for Cascade
- “Load-balancing” node selection using Least-used node.
- SigTerm vs SigKill changes to allow epilogue script to work with ALTD.

Future SLURM Work/Wish List

- Finish additional parts of GRES accounting
- More robust ZN functionality
- SLURM internal cache flush capability via scontrol
- Exclude group/user list for reservations (minor)
- Not having to specify partition when using reservation with nodes not in the default (minor)

The Systems

Name	Arch. Type	Number of Nodes	Number of Processors	Node Layout	GPU	Node Memory
Rosa	Cray XE6	1496	47872	2x16x1	None	32GB
Todi	Cray XK6*	272	4352	1x16x1	1 Fermi/node 186 nodes	32GB
Julier	non-Cray	12	288	2x6x2	None	10-48GB 2-256GB
Pilatus	non-Cray	22**	704	2x8x2	None	64GB
Rothorn	non-Cray	1	256	32x8x1	None	2TB
Albis	Cray	72	1728	2x12x1	None	32GB
Lema	Cray	168	4032	2x12x1	None	32GB
Castor/Pollux	non-Cray	32	384	2x6x1	2 per node	24GB

The Systems (Continued)

Name	Arch. Type	Number of Nodes	Number of Processors	Node Layout	GPU/Node	Node Memory
Eiger	non-Cray	21	300			
		4		2x6x1	2 Fermi GTX480	24GB
		5		2x6x1	1 Geforce GTX285	24GB
		2		2x6x1	2 Tesla s1070	24GB
		2		2x6x1	2 Fermi c2070	24GB
		2		2x12x1	2 Fermi m2050	48GB
		2		2x12x1	2 Fermi c2070	48GB
		4		2x6x1	1 Geforce GTX285	48GB

The Systems Miscellaneous

- Ela—main gateway from outside to systems
- Fojorina01/Fojorina02—Hosts the common slurmdbd for all principal systems
- db.cscs.ch—hosts the central CSCS DB and SLURM DB

The Test Systems

- Gele—A Cray 16-2x16x1-node system w/32GB per node
- Dolomite—set of non-Cray blades, currently using 4-2x6x2 nodes each with ~11GB
- VM's with emulators for XK6 and Cascade
- Use their own DB.

SLURM Features Used

- Basics
  - partition/node configuration options
  - Cray and Cons_res modules for node selection
  - Backfill scheduler
  - Priority multifactor
- Additional
  - Lua scripts (job submission policy enforcement and group priority)
  - Task/affinity
  - GRES (some systems)
  - Accounting (via slurmdbd and MySQL DB)
  - Zero-Node ('ZN') jobs for post-processing (some systems)
  - Various Prologues/Epilogues (some systems)
- Contribs/Misc
  - PAM module (some systems)
  - SPANK module (some systems)
- Some User/Admin features
  - Advanced reservations
  - Job Chaining

SLURM Features Not Used

- Fairshare
- QOS
- Gang Scheduling
- Preemption
- Command wrappers on Cray

SLURM By System

System	SLURM	Sched	Select	Prolog Epilog	Lua	Priority	ASE	Task/Affinity	SPANK	PAM	Accounting
Rosa	2.3.4 *1,2	backfill	Cray CR_Memory	Task	Yes	multifactor	limits	No			slurmdbd
Todi	2.3.4 *1-5	backfill	Cray CR_Memory	Task	Yes	multifactor	Not set	No			slurmdbd
Julier	2.3.4	backfill	Cons_res CR_CPU_Memory	Task	Yes	multifactor	limits	Sched			slurmdbd
Pilatus	2.3.4 *1-4	backfill	Cons_res CR_CPU_Memory	Task	Yes	multifactor	Assoc.	Sched	Auto Binding		slurmdbd
Eiger	2.3.4	backfill	Cons_res CR_Core	Task (pro-/epi-)	Yes	multifactor	Not set	Sched		Yes	slurmdbd
Rothorn	2.3.4	backfill	Cons_res CR_Core_Memory	Task	No	multifactor	limits	Cpusets			slurmdbd

SLURM By System (Continued)

System	SLURM	Sched	Select	Prolog Epilog	Lua	Priority	ASE	Task Affinity	SPANK	PAM	Accounting
Albis	2.3.4 *???	backfill	Cray CR_Memory	Task	Yes	multifactor	limits	No			slurmdbd
PPAlbis	2.3.4 *???	backfill	Cons_res $ CR_Core	Task	Yes	multifactor	limits	No			slurmdbd
Lema	2.3.4 *???	backfill	Cray CR_Memory	Task	Yes	multifactor	limits	No			slurmdbd
PPLema	2.3.4 *???	backfill	Cons_res $ CR_Core	Task	Yes	multifactor	limits	No			slurmdbd
Castor Pollux	2.3.1 *???	backfill	Cons_res CR_Core_Memory	Task	No	multifactor	N/A	Sched			None

SLURM 2.3.4 Patches In Use

1. Basic Cray XK6
2. Node Memory—Fix bug with “--mem” option on Cray systems.
3. Zero-Node patches—provides limited ability to run “post-processing” jobs on front ends.
4. GRES Count Underflow
5. sacct -N fix

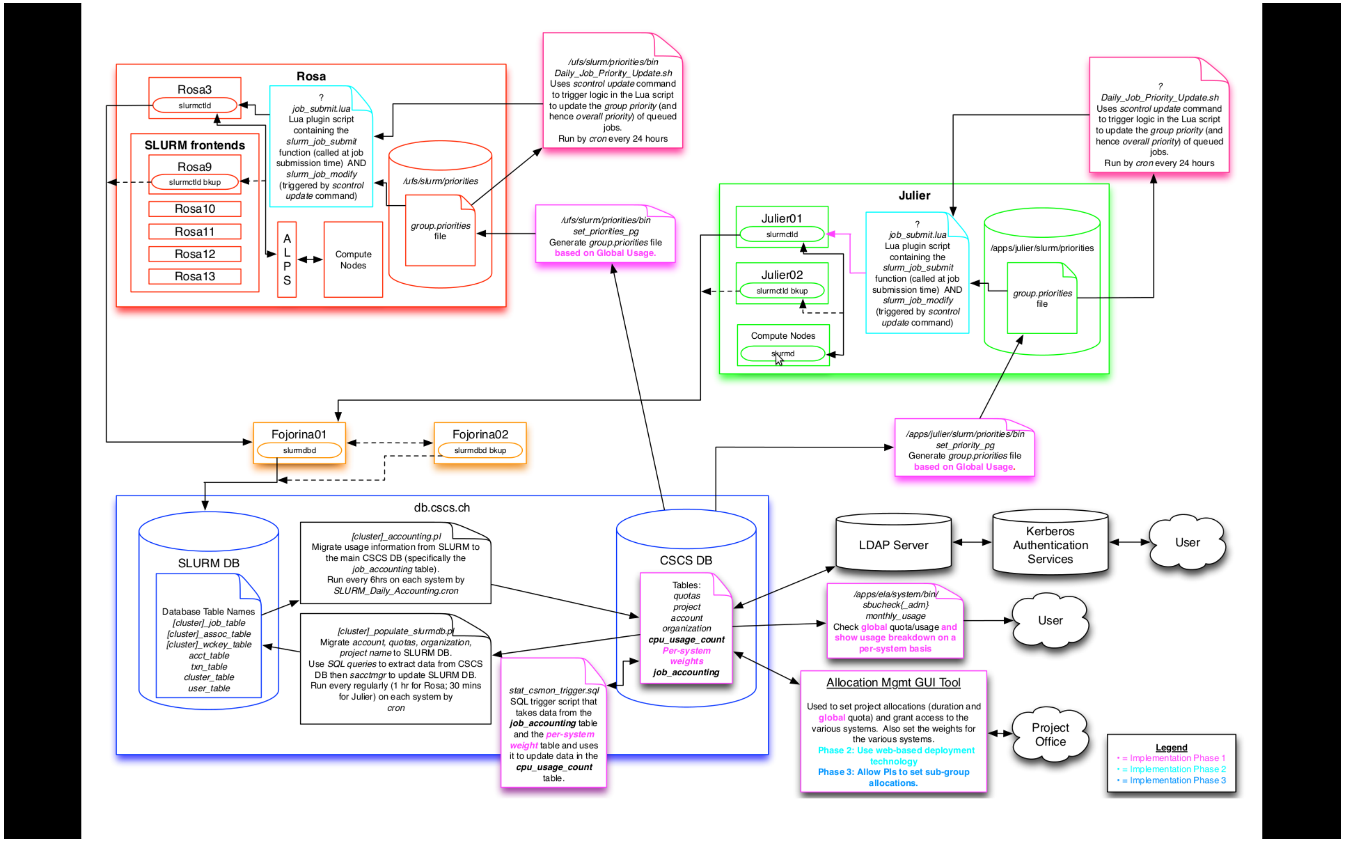
How CSCS Uses SLURM

- SLURM “ecosystem” consists of SLURM and various scripts and utilities built around it
- This ecosystem interacts with the site's general DB—user id's, accounts, allocations
- All systems with accounting have the following cron scripts
  - [cluster]_populate.pl
  - [cluster]_accounting.pl
  - set_priority_pg

How CSCS Uses SLURM

- Time allocations and the associated users granted access by Project Office
- Information stored in the central CSCS DB
- The SLURM "ecosystem" exchanges info w/CSCS DB at several spots including
  - Cron scripts
  - User scripts
- Originally, allocations were on a per-cluster basis
- Now, via our scripts and DB, we provide a "common allocation" that can be/are consumed across various systems
  - CPU hours are weighted depending upon machine/node type
  - Weights can be easily modified over time as needed via DB

How CSCS Uses SLURM



?
job_submit.lua
Lua plugin script containing the *slurm_job_submit* function (called at job submission time) AND *slurm_job_modify* (triggered by *scontrol update* command)

/ufs/slurm/priorities/bin
Daily_Job_Priority_Update.sh
Uses *scontrol update* command to trigger logic in the Lua script to update the *group priority* (and hence *overall priority*) of queued jobs.
Run by *cron* every 24 hours

/ufs/slurm/priorities/bin
set_priorities_pg
Generate *group.priorities* file **based on Global Usage.**

?
Daily_Job_Priority_Update.sh
Uses *scontrol update* command to trigger logic in the Lua script to update the *group priority* (and hence *overall priority*) of queued jobs.
Run by *cron* every 24 hours

?
job_submit.lua
Lua plugin script containing the *slurm_job_submit* function (called at job submission time) AND *slurm_job_modify* (triggered by *scontrol update* command)

/apps/julier/slurm/priorities/bin
set_priority_pg
Generate *group.priorities* file **based on Global Usage.**

Database Table Names
[cluster]_job_table
[cluster]_assoc_table
[cluster]_wckey_table
acct_table
txn_table
cluster_table
user_table

[cluster]_accounting.pl
Migrate usage information from SLURM to the main CSCS DB (specifically the *job_accounting* table).
Run every 6hrs on each system by *SLURM_Daily_Accounting.cron*

[cluster]_populate_slurmdb.pl
Migrate *account, quotas, organization, project name* to SLURM DB.
Use *SQL queries* to extract data from CSCS DB then *sacctmgr* to update SLURM DB.
Run every regularly (1 hr for Rosa; 30 mins for Julier) on each system by *cron*

Tables:
quotas
project
account
organization
cpu_usage_count
Per-system weights
job_accounting

stat_csmon_trigger.sql
SQL trigger script that takes data from the ***job_accounting*** table and the ***per-system weight*** table and uses it to update data in the ***cpu_usage_count*** table.

/apps/ela/system/bin/
sbucheck{_adm}
monthly_usage
Check **global** quota/usage **and show usage breakdown on a per-system basis**

Allocation Mgmt GUI Tool

Used to set project allocations (duration and **global** quota) and grant access to the various systems. Also set the weights for the various systems.
Phase 2: Use web-based deployment technology
Phase 3: Allow PIs to set sub-group allocations.

Legend
- • = Implementation Phase 1
- • = Implementation Phase 2
- • = Implementation Phase 3

SLURM Job Priorities at CSCS

- CSCS maintains the concept of a group priority
- Implemented via cron and lua scripts
- Uses the “nice” value component of multifactor priority equation
- Equation for the group priority factor:
  - group_priority(within budget) = weight_nice*(used/quota – time_factor)
  - group_priority(over budget) = weight_nice*over_used + time_factor*penalty
  - Where:
    - weight_nice = 1000 (constant)
    - time_factor = (now - start_time)/(end_time – start_time)
    - over_used = MAX(used/quota,5.0)
    - Penalty = [based upon project type where]

Project Type	alps	lup	cp	small	test
Penalty	10	100	100	1000	1000

SLURM Job Priorities at CSCS

- Maintain concept of local and global usage for a given group
- Most systems, deny jobs from over-budget accounts
- On Rosa, “bottom-feeding” jobs allowed (over-budget but have the lowest of possible priorities)
- Use cron script to periodically update priorities for pending jobs

Julier Limits

- Additional limits were edicted for Julier
- MaxCPUs, MaxJobs, MaxSubmit
- Partition definition doesn't handle these
- Used association records instead
- Each user winds up having one record for each partition of each account to which they belong on Julier

Automatic Binding on Pilatus

- Some internal users demanded a simplified automatic binding of a specific pattern
- Solution—created a SPANK module
- Started with old auto-affinity LLNL module, stripped it down and then changed some of the logic
- Binding is now as follows:
  - Only 1 HW thread (vCPU)/core is used
  - Fill across a socket before using next socket
  - Tasks must fit on a single socket (No crossing boundaries)
  - Can have more than one task per socket if all these tasks fulfill above condition
  - If not enough sockets exist to place all tasks of the node completely, according to above rules, job is rejected.

Some Challenges Along the Way

- Dropped Cluster Problem (due to packets from controller to slurmdbd being too large)
- Unable to launch jobs when only one FE is down (August 2011)
- SLURM & ALPS out-of-sync causing SLURM to get stuck (August/September 2011). SchedMD fixed this.
- Various instances of garbage being written to various fields in the SLURM DB. Had to manually fix some DB entries in some cases.
- Backfill not processing enough jobs at a time on Rosa (Fall 2011?) SchedMD provided a patch that fixed this.
- Understanding proper use of various affinity options (especially at software thread level).

Summary

- CSCS has diverse array of small to mid-size systems
- Successfully manage these resources with SLURM
- Maintain a SLURM “ecosystem” of SLURM instances, DB and scripts to provide both CSCS and users with the desired resource management functionality