

Batch System Deployment on a Production Terascale Cluster

*Texas Advanced
Computing Center*

Karl W. Schulz, Kent Milfeld, Chona S. Guiang,
Avijit Purkayastha, Tommy Minyard, and
John R. Boisseau

Cray, Inc | John Casu



Texas Advanced Computing Center
THE UNIVERSITY OF TEXAS AT AUSTIN



Outline

- Attributes of Ideal Batch System
- Cluster Overview
- LSF Batch System Deployment
 - Configuration
 - System Integration / Custom Developed Utilities (LSF API)
 - Possible Configuration Pitfalls
 - Scalability Results
 - Outstanding Items to be Resolved
- LSF Report Card



Ideal Batch System Components

- Fault Tolerant
 - Advanced Scheduling Options
 - Easily Configurable, Extensible (programming API)
 - Scalable
 - Robust
 - Interface Easily with External Accounting Infrastructure
 - User Support
 - Grid Computing Aware/Enabled
 - Free
- Trade Off Between These Characteristics Led to the Deployment of LSF HPC on our 600 Processor Xeon Cluster



Cluster Overview: *Lonestar*

Hardware Configuration	
Compute Nodes (282)	Dell PowerEdge 1750, 3.06 GHz FSB 533 MHz 2GB Memory (266 MHz, dual channel) 36 GB SCSI Disk
I/O Nodes (16)	Dell PowerEdge 2650 3.06 GHz FSB 533MHz (4GB Mem.)
Front-End Nodes (2)	Dell PowerEdge 2650 3.06 GHz FSB 533MHz (4GB Mem.)
High Speed Interconnect	Myrinet 2000 (2Gb/s) CLOS topology
NFS/Management Network	Dell PowerConnect 5224 / 3248 (GigE switch hierarchy)
Disk RAIDs (16)	Power Vault 220S, 14x72GB SCSI disk/enclosure



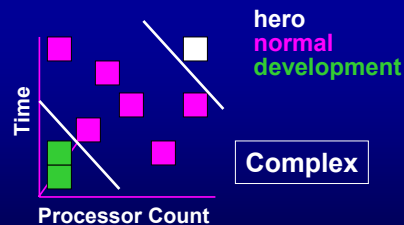
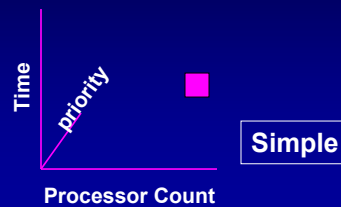
Cluster Overview: *Lonestar*

Base Software Stack	
Kernel	2.4.20-30 (with PAPI kernel patches)
Compilers	Intel (7.1), gcc (2.9.6 & 3.2.3), PGI (5.1.3)
Debugging	TotalView 6.3.1
Hardware Performance	PAPI 3.0
Profiling, Monitoring	Ompitrace 1.1/Paraver 3.1, Tau (pdtools 3.0)
Batch System	LSF HPC 5.1
Cluster Management	Cray Rx 3.1 (a variant of NPACI Rocks)
Parallel I/O	ROMIO/PVFS 3.5.1



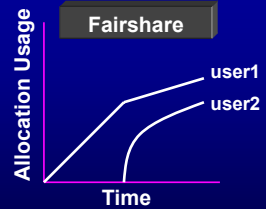
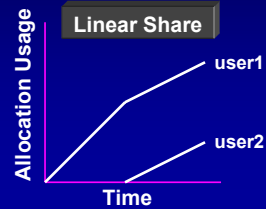
Basic Batch System Configuration

- Queue definitions adopt a “keep it simple” strategy by defining only three queues for default users:
 - normal*, and *high*
 - 256 processor limit, 48 hour run limit
 - High queue has increased scheduling priority
 - development*
 - 16 processor limit, 30 minute run limit
 - interactive execution
- Remaining special queues:
 - hero*, 512 processors max, pre-approved users only
 - sysrest*, maximum processor count, staff administrators only



Basic Batch System Configuration

- We use LSF scheduler with *global Fairshare* and *Backfill* enabled
- Pre-execution script (*esub*) is used as a global job filter:
 - Provides simple syntax-error checking
 - Require user-specified runlimits
 - Avoid job flooding by restricting max number of jobs
 - Enforce accounting policies
 - Interface with external project-based accounting routines
 - Only allow jobs that can afford the resources
- Post-execution script (*eexec*) is used to clean up temporary file space after each job completes



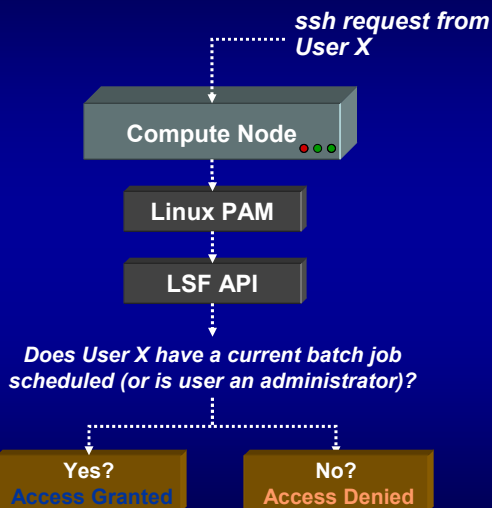
System Integration

- Compute-Node Access Management
 - Users log onto front end nodes and interact with compute nodes through the batch system
 - On a cluster with a large, diverse set of users, allowing continuous *ssh* access to all compute nodes is probably unwise
 - Allowing remote access does, however, allow experienced users to monitor their jobs at a very detailed level
- *A Compromise*: login access granted only to users who have actively scheduled jobs on a specific node
 - Requires direct integration with the batch system to ascertain which nodes are assigned to individual users
 - Integration accomplished using LSF API



System Integration: Access Management

- ssh access controlled via the development of an additional Linux-PAM module (*pluggable authentication module*)
- PAM module built on top of LSF API
- Key administrators granted constant access



System Integration: Access Management

- To implement, the additional PAM module is simply added to ssh authentication (see `/etc/pam.d/sshd`):

```
##PAM-1.0
auth required /lib/security/pam_stack.so service=system-auth
auth required /lib/security/pam_nologin.so
account required /lib/security/pam_stack.so service=system-auth
password required /lib/security/pam_stack.so service=system-auth
session required /lib/security/pam_stack.so service=system-auth
session required /lib/security/pam_limits.so
session optional /lib/security/pam_console.so
account required /lib/security/pam_TACC_LSF.so
```

- Developed module grants access to all LSF administrators; login access to the entire cluster can be trivially controlled via single configuration file on front-end



System Integration: Job Monitoring Utility

- LSF has a capable set of utilities for submitting, stopping, deleting, and resuming batch jobs
- However, we also need a concise summary utility to aid users in managing their runs and to monitor the overall state of the system; helpful to answer the common age-old questions:
 - Is my job submitted? Is it running?
 - Is the machine busy?
 - How big of a job should I submit to get in quickly?
 - Are there any advanced reservations?
 - Who's hogging the development queue?
- We developed a job-status utility with an interface similar to the *Maui showq* command
- Our *lsf_showq* utility is built using LSF API to query the batch system directly



Example *lsf_showq* Output

ACTIVE JOBS				Summary of Advanced Reservations for the Next Week			
JOBID	JOBNAME	USERNAME	STATE	PROC	WCLIMIT	QUEUETIME	
14694	equillda	user1	Running				
14701	v	user2	Running				
14707	v	user3	Running	16	19:11:02	Tue Feb 3	17:49:36
14708	jet08	user4	Running	32	0:38:36	Tue Feb 3	18:17:10
14713	rti	user5	Running	64	3:58:25	Tue Feb 3	20:36:59
14714	cyl	user6	Running	128	23:16:36	Tue Feb 3	21:55:10
6 Active jobs				272 of 556 Processors Active (48.92%)			
IDLE JOBS							
JOBID	JOBNAME	USERNAME	STATE	PROC	WCLIMIT	QUEUETIME	
14716	bigjob	user7	Idle	512	0:15:00	Tue Feb 3	22:18:57
14719	smalljob	user7	Idle	256	0:15:00	Tue Feb 3	22:35:31
2 Idle jobs							
BLOCKED JOBS							
JOBID	JOBNAME	USERNAME	STATE	PROC	WCLIMIT	QUEUETIME	
14717	hello	user7	Held	16	0:15:00	Tue Feb 3	22:19:07
14718	hello	user7	Held	32	0:15:00	Tue Feb 3	22:19:15
4 Blocked jobs							
Total Jobs: 12		Active Jobs: 6		Idle Jobs: 2		Blocked Jobs: 4	
ADVANCED RESERVATIONS				Summary of Running, Idle, and Blocked Jobs			
RESV ID	PROC	RESERVATION					
karl#79	556	Tue Mar 23 09:00:00 2004	-				



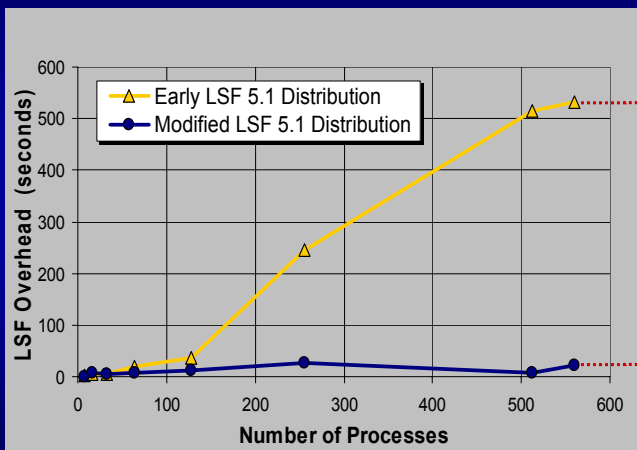
LSF Batch System: Scalability

- Initial large processor runs (> 128 procs) appeared to incur substantial overhead to start and stop individual jobs
- LSF overhead was quantified by running test jobs both with and without the batch system
 - Processor counts ranged from 8 to 564
 - Wall-clock time of simple MPI job measured by hand (via mpich-gm version of *mpirun*)
 - Same MPI job was submitted through the batch system

$$\text{Overhead} \equiv (\text{wall time})_{\text{LSF}} - (\text{wall time})_{\text{mpirun}}$$



LSF Batch System: Scalability



{ Almost 9 minutes of overhead originally

{ Modified version has less than 30 seconds overhead



Possible Pitfalls (Fairshare)

- Configuration of Global Fairshare
 - If you enable global Fairshare as described in the documentation, you may encounter problems with the successful recovery of held jobs
 - Problem arises because LSF adds users to the share partition as jobs are submitted
 - If enough time elapses while a job is held (e.g., during a maintenance window), the job will no longer have a fairshare partition associated with it.
 - The fix is to add users to the fairshare policy at the *job scheduling* stage (not at job submission)

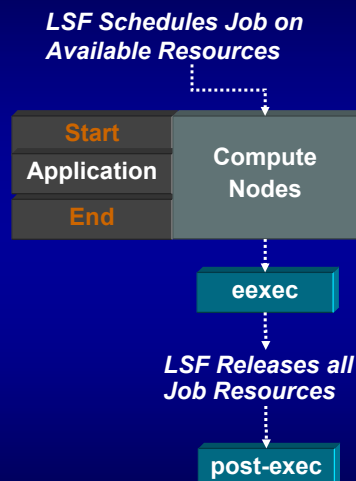


(please refer to Appendix for specific configuration syntax)



Possible Pitfalls (Post-Execution Script)

- A common task of any post-execution script is to clean up after a user's job: e.g. flush the temporary disk space on each compute node
- LSF provides a "post-execution" mechanism on a queue-level basis
- We recommend to instead use the *eexec* script option
 - *eexec* can be configured to run at either the beginning or end of a job
 - Unlike the post-execution script, *eexec* is executed prior to releasing the resources of a given job
 - Prevents subtle problem of purging temporary file systems of a previous job when a new job is already allocated to the same node



(please refer to Appendix for specific configuration syntax)



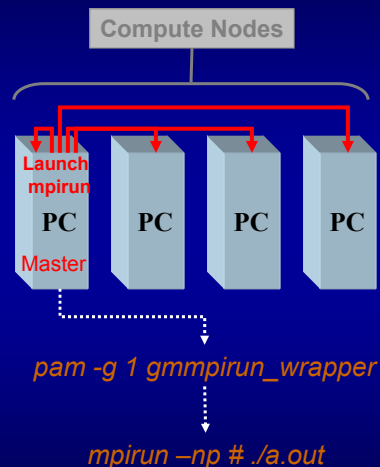
Outstanding Issues to be Addressed

- Advanced Reservations (AR)
 - AR does not presently work in conjunction with “Exclusive” node usage
 - Exclusive usage is a common necessity for multi-processor systems (e.g. 2-way, 4-way Linux servers) for scheduling memory-demanding applications
 - Net result is that you cannot make any advanced reservations on exclusive-usage nodes, regardless of the job runlimit
 - Current workaround is to schedule all system maintenance ARs during machine downtime
 - Possible for recurring maintenance
 - Impossible to make near-term reservations on running system
 - ARs also have the tendency to lose processor counts over extended periods of time



Outstanding Issues: Fault Tolerance

- LSF provides MPI fault tolerance via the use of a parallel application manager (*PAM*) which spawns MPI jobs
- PAM keeps track of children MPI tasks and tracks when a child process exits or is killed
- Benefit is that if one or more MPI procs fail, LSF can clean up the remaining MPI PIDs on remaining compute nodes
- A single point of failure still exists; namely if the master compute node dies, then PAM is not available to clean-up the remaining MPI procs
- PAM failure can result in hung processes which can lead to processor overload



Summary / Report Card

- LSF HPC 5.1 has been successfully deployed on our production Xeon cluster
- LSF API has been used extensively to integrate the batch system into the overall user environment
- Possible configuration pitfalls have been presented along with some outstanding issues to be addressed on Linux clusters

Real Time Configuration	A
Extensible (eg. via programming API)	A
Scalability	A
Grid Infrastructure	A
Documentation	B -
User Support	B
Fault Tolerance	C
Advanced Reservations	C



Thanks for Your Time

Questions?

