Lustre® at Blue Waters

LUG 2013

Nathan Rutman (nathan_rutman@xyratex.com)

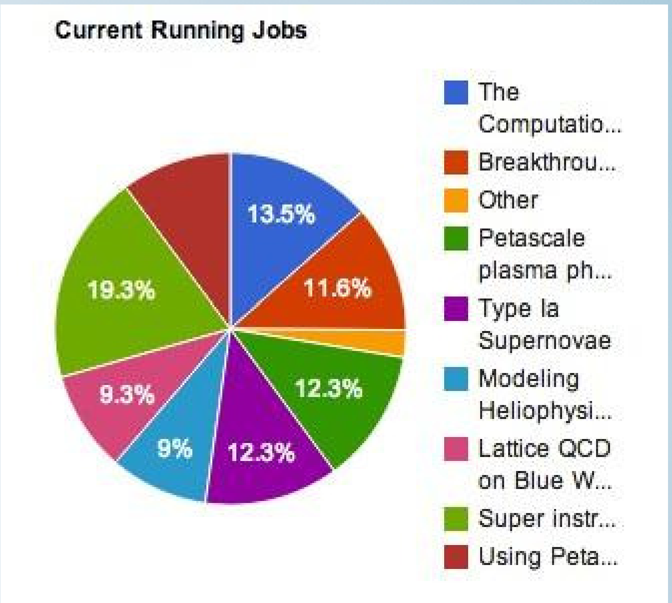
xyratex

History

- 2007 - University of Illinois and NCSA select IBM to build Blue Waters, plan for 1 petaflop by end of 2012
- Aug 2011 - IBM withdraws, project was "more complex and required significantly increased financial and technical support"
- Nov 2011 - NSF approves NCSA's Cray-based deployment of Blue Waters
- July 2012 - Cray builds and installs system in 9 months, begin software/hardware scaling work
- Nov 2012 - Full system available for NSF research

Stats

- Compute
  - 237 Cray XE6 cabinets
  - 32 Cray XK7 cabinets
  - 25766 clients
  - 1.5 PB memory
  - sustained petaflop computing
  - 11.6 PF peak



- Storage
  - 25 PB Lustre storage on Sonexion hardware
  - 1.1 TB/s total
  - 22 PB scratch
  - 1.0 TB/s /scratch

Storage

- 3 Filesystems
  - home
  - project
  - scratch
    - 360 OSSs, 1440 OSTs
- Rack
  - 6 SSU's / 12 OSS's
  - Top-of-rack IB switch
- SSU
  - 2 OSS's in active/active pair
  - 4 OSTs per OSS
  - Each OST is MDRAID 6 8+2
  - 82 7.2K fatsas 2TB drives
- MDS
  - 72-drive RAID 10 300GB 15K
  - 128 GB, 32 cores

Network

- Gemini 3D torus
- Fine-grained routing from clients to Cray Gemini-IB LNET routers
- Top-of-rack IB switch in each Sonexion rack
- TOR switches connected to Cray routers
  - 16 LNET routers : 12 OSSs to match router throughput

- 2 Cray routers per FS for MDS traffic, connected to Director-class core switch
- Core switch connected to each TOR

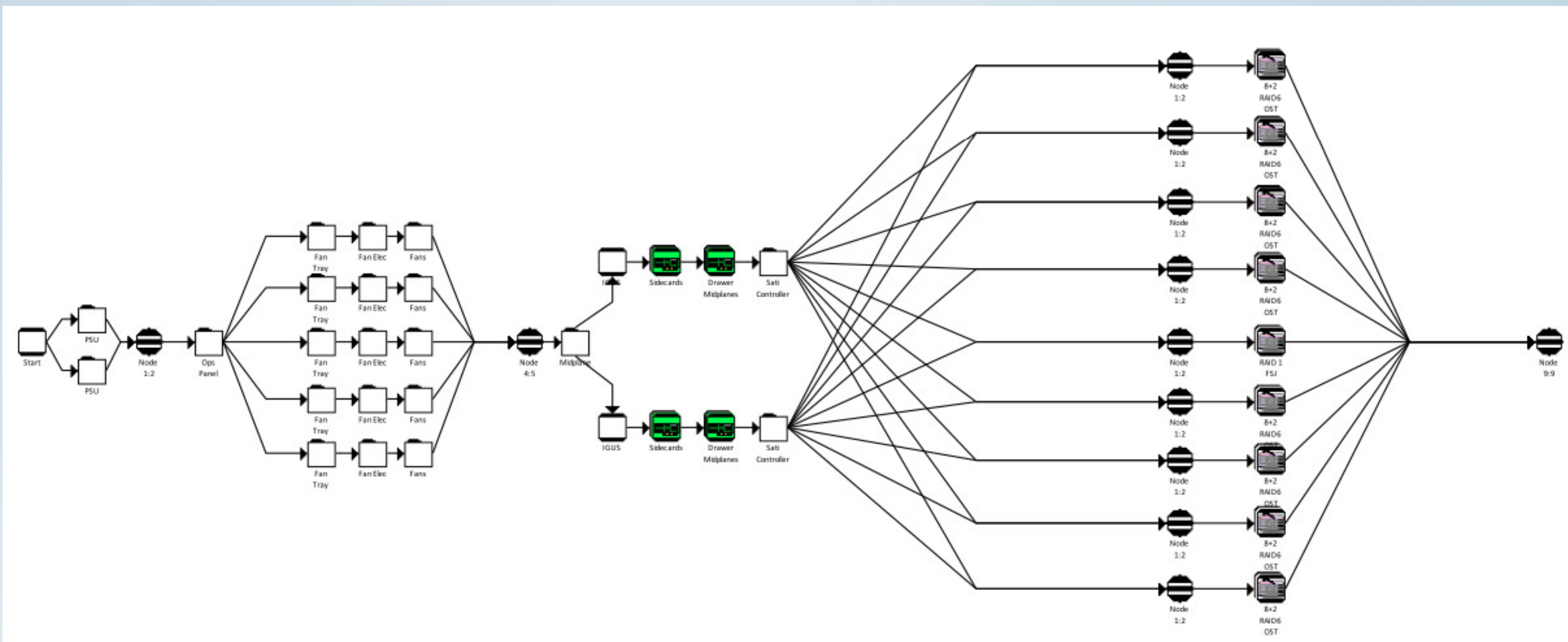
Software

- Lustre 2.1 servers
  - diskless boot from mgmt node
- Lustre 1.8.7 clients

- First large-scale Lustre 2.1 deployment

Reliability

- 99% availability
- Full component redundancy
- Redundant path analysis for each component (mgmt node, controller, SSU, rack, MDS, etc.)



Early scaling

- HA timings
  - fsck
- Fine-grained routing
  - large routing tables
- fcntl(F_UNLCK) failure
  - 1.8-2.1 compat
- Block bitmap reads
- MDS problems
  - high CPU
  - "sluggishness"
  - all MDS threads stuck
  - "pulsing"
  - cascading client timeouts
  - infinite recovery loops

MDS behavior

- Confluence of many factors
  - Memory allocation race
  - Inaccurate thread accounting
  - Blocking callback behavior in the face of filled network buffers
  - Poor utilization of existing buffers

- Details presented in Paris at LAD '12
- ...but I'll talk about three of them anyhow.

Problems and Solutions

- Memory Allocation race (MRP-633)
- Non-visibility of buffer usage (MRP-644)
- Inaccurate "queued" accounting (MRP-622)
- Bad thread accounting
  - Mishandled srv_hpreq_ratio (MRP-661, LU-1963)
  - Missing OOM-killed thread accounting (MRP-648)
  - Number of HP threads (MRP-664)
- ELC on MDS threads (MRP-655)
- Cancel retries (MRP-477)
- Request buffer size (MRP-670)
- Nonblocking lock callbacks (MRP-663)
- LNET issues
  - Router buffer sizing
  - Network Priority
  - Unavailable router passthrough (MRP-658)

xyratex

Threads Blocking on Locks

- Sometimes we see all server threads blocked on lock callback
- Some reasons:
  1. blocking callback never gets sent
  2. router drops bl callback
  3. client doesn't correctly cancel
  4. cancel gets lost in network
  5. router drops cancel
  6. cancel response takes too long to reach mds
  7. backed up on router
  8. queued on mds
- Only the HP thread is left

Memory allocation race

- Symptoms
  - no rpc processing
  - no disk activity
  - high cpu load
  - sluggish MDS console response
- What's happening
  - rqbd buffers fill with incoming reqs
  - each thread sees bufs are below low-water, spinlock to create more
  - threads start to race, bogging down all cpus
- What should we do?
  - if anyone else is in the middle of allocating, skip it.
- Results
  - no more sluggishness, but the MDS can't keep up with incoming request rate

xyratex

Request buffers

We assume buffers are being created, but hard to see.

(stat's req_qdepth was really just req_in's)

```
mdt.snx11003-MDT0000.mdt.req_buffer_stats=
network buffers:
  created: 106496
  available: 1723
  idle: 0
  history: 0
  max allowed: 610801
  size: 113386
  total memory used: 11515 MB
requests:
  active: 1022
  hp active: 0
  incoming: 0
  queued: 103751
```

Request buffer size

- Symptoms
  - created - avail = active + queued
  - So 1:1 buffers to rpc's
- What's happening
  - srv_max_req_size = MDS_MAXREQSIZE + 1024 =113kB
  - srv_buf_size = MDS_MAXREQSIZE + 1024
  - We must have enough remaining space in a buffer to accept a srv_max_req_size for LNET to accept it.
  - So we can only have a single req per rqbd.
- What should we do?
  - If we doubled the buffer size and the average message size is 4k, we could put 113k/4k = 25 messages before the "last" one.

Request buffer size (con't)

- Results
  - Actual message size ~1.4kB average

```
mdt.snx11003-MDT0000.mdt.req_buffer_stats=
network buffers:
  created: 2048
  available: 1884
  idle: 0
  history: 0
  max allowed: 305400
  size: 226772
  total memory used: 442 MB
requests:
  active: 510
  hp active: 0
  incoming: 0
  queued: 24998
```

Conclusions

- Fine-grained routing - solid
- I/O data path - solid
- MD handling path - solid
- ptlrpc request handling code - shakier than we would like
- lock callback code - not robust against req handling

- Lustre 2.1 at scale -- solid now on BW
- For any large deployment -- be prepared to do some work!
- Lustre success vs GPFS failure

Thank You

nathan_rutman@xyratex.com

xyratex