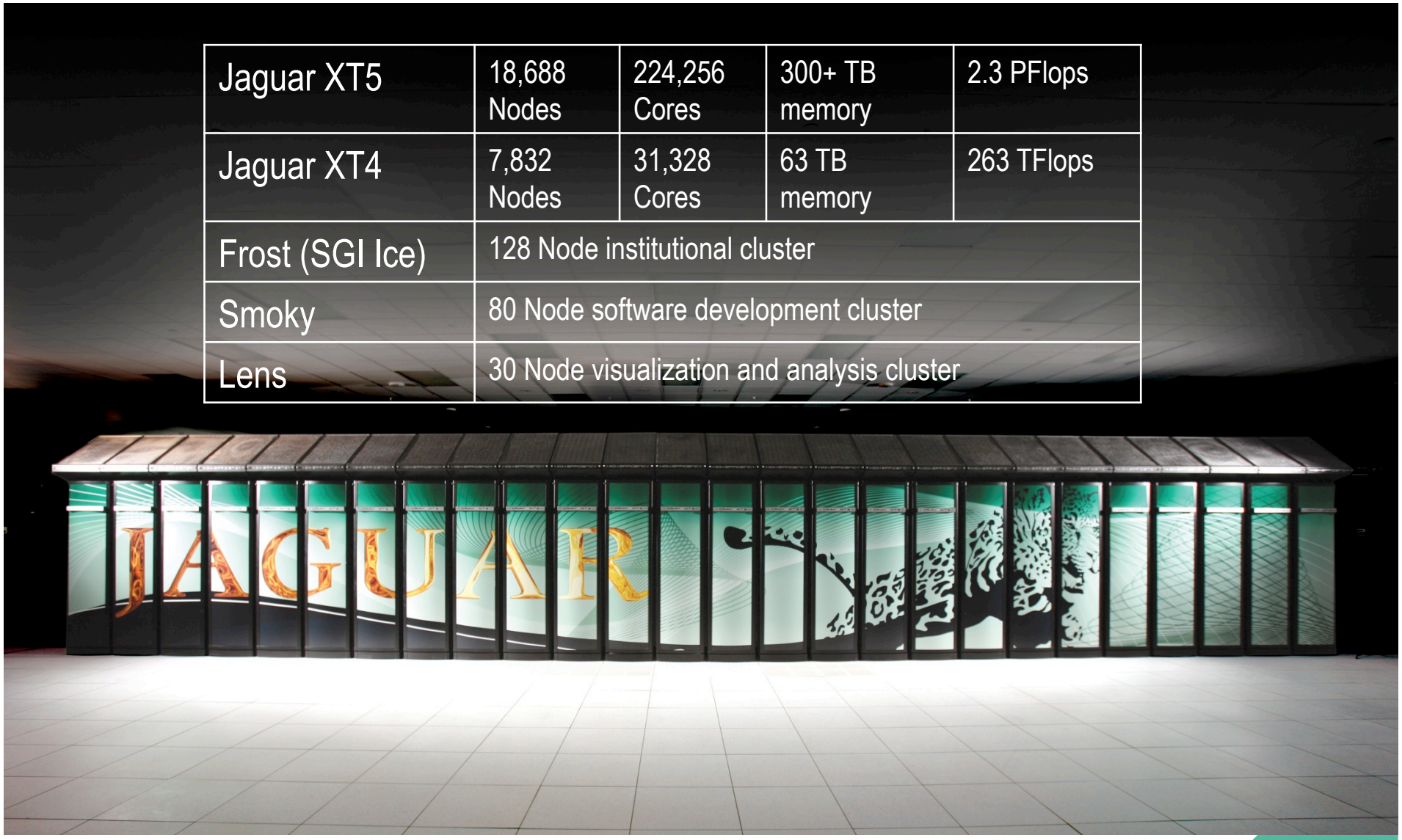
Efficient Object Storage Journaling in a Distributed Parallel File System

Presented by Sarp Oral

Sarp Oral, Feiyi Wang, David Dillow, Galen Shipman, Ross Miller, and Oleg Drokin

A Demanding Computational Environment

Jaguar XT5	18,688 Nodes	224,256 Cores	300+ TB memory	2.3 PFlops
Jaguar XT4	7,832 Nodes	31,328 Cores	63 TB memory	263 TFlops
Frost (SGI Ice)	128 Node institutional cluster			
Smoky	80 Node software development cluster			
Lens	30 Node visualization and analysis cluster			



Spider

Fastest Lustre file system in the world

Demonstrated bandwidth of 240 GB/s on the *center wide* file system

Largest scale Lustre file system in the world

Demonstrated stability and concurrent mounts on major OLCF systems

- Jaguar XT5
- Jaguar XT4
- Opteron Dev Cluster (Smoky)
- Visualization Cluster (Lens)

Over 26,000 clients mounting the file system and performing I/O

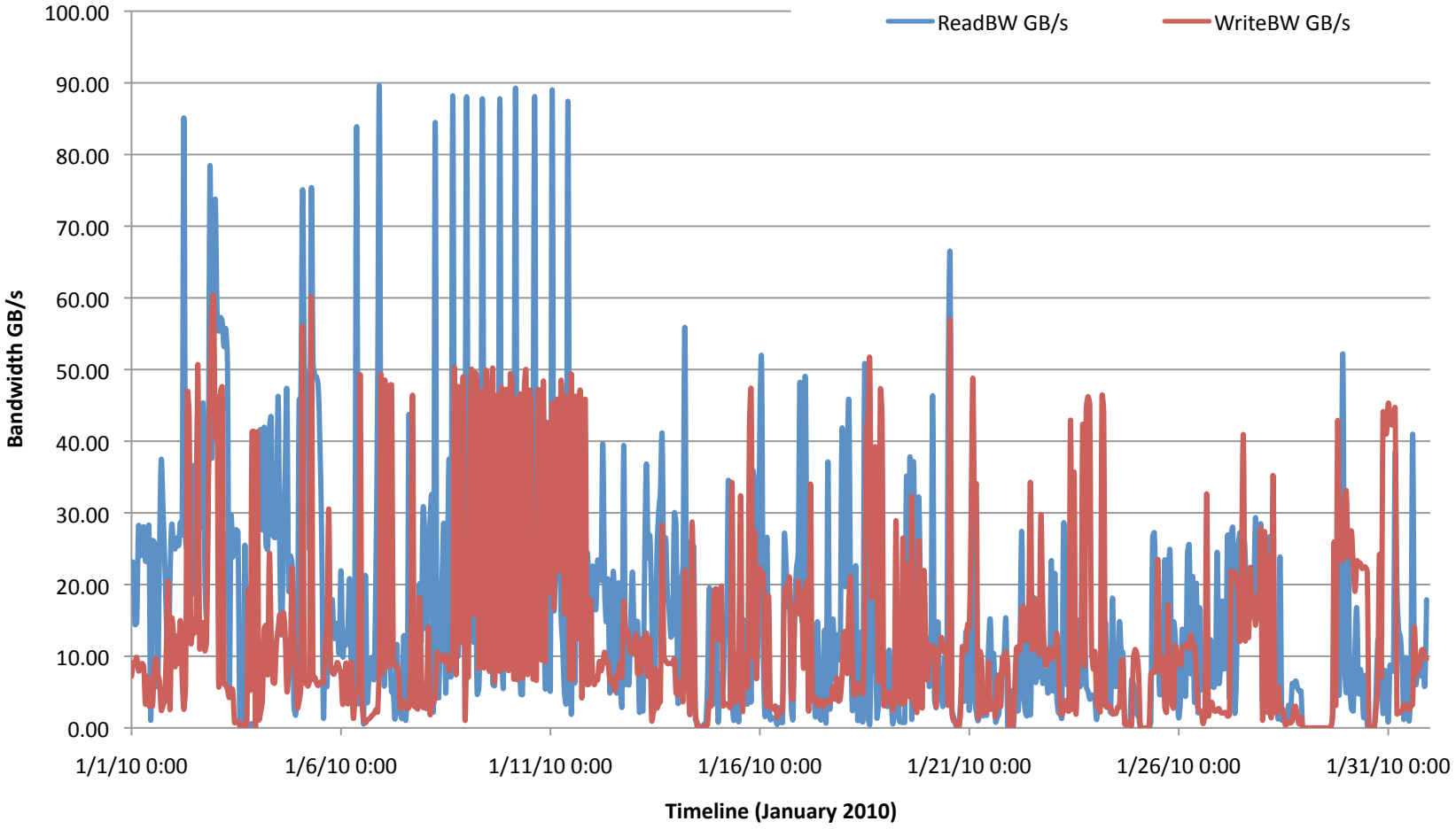
General availability on Jaguar XT5, Lens, Smoky, and GridFTP servers

Cutting edge resiliency at scale

Demonstrated resiliency features on Jaguar XT5

- DM Multipath
- Lustre Router failover

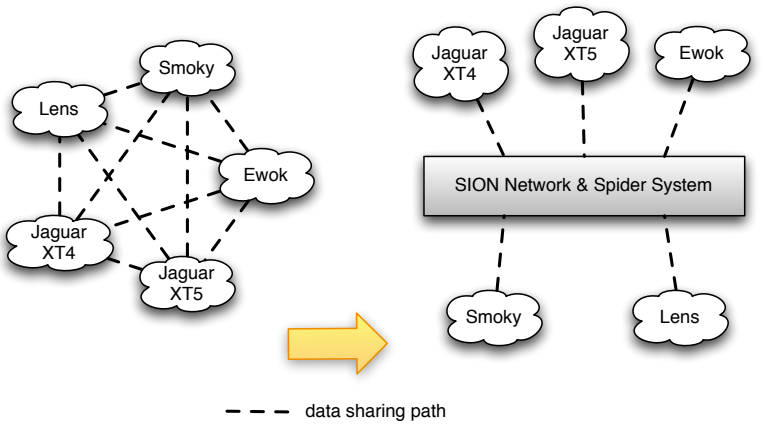
Designed to Support Peak Performance



Max data rates (hourly) on ½ of available storage controllers

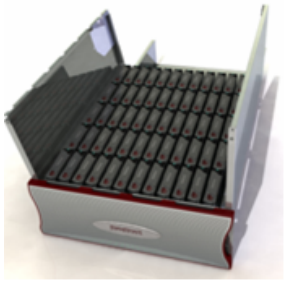
Motivations for a Center Wide File System

- Building dedicated file systems for platforms does not scale
  - Storage often 10% or more of new system cost
  - Storage often not poised to grow independently of attached machine
  - Different curves for storage and compute technology
  - Data needs to be moved between different compute islands
    - Simulation platform to visualization platform
  - Dedicated storage is only accessible when its machine is available
  - Managing multiple file systems requires more manpower



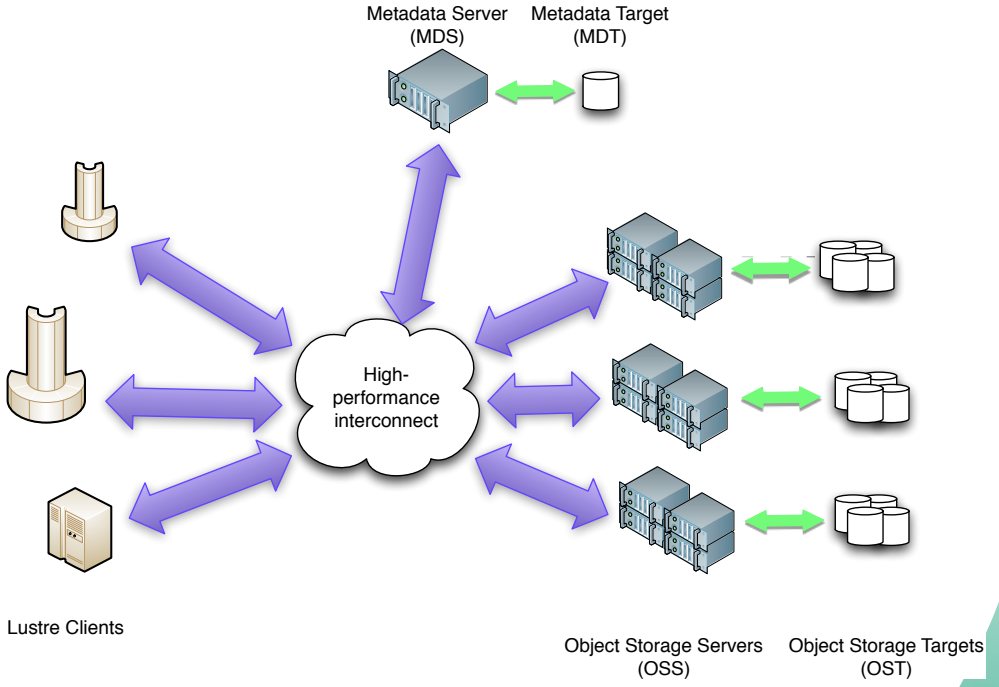
Spider: A System At Scale

- Over 10.7 PB of RAID 6 formatted capacity
- 13,440 1 TB drives
- 192 Lustre I/O servers
- Over 3 TB of memory (on Lustre I/O servers)
- Available to many compute systems through high-speed SION network
  - Over 3,000 IB ports
  - Over 3 miles (5 kilometers) cables
- Over 26,000 client mounts for I/O
- Peak I/O performance is 240 GB/s
- Current Status
  - in production use on all major OLCF computing platforms



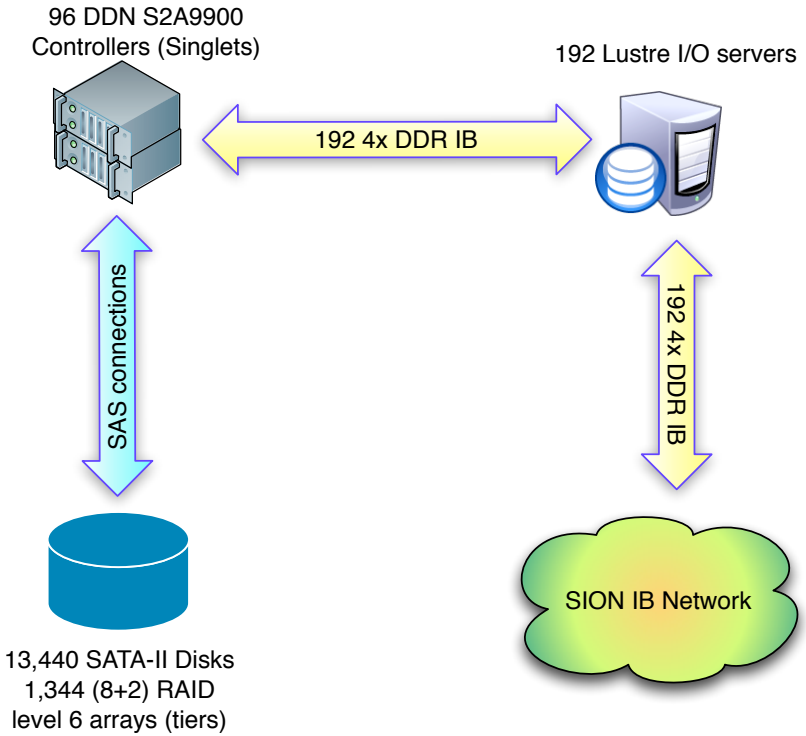
Lustre File System

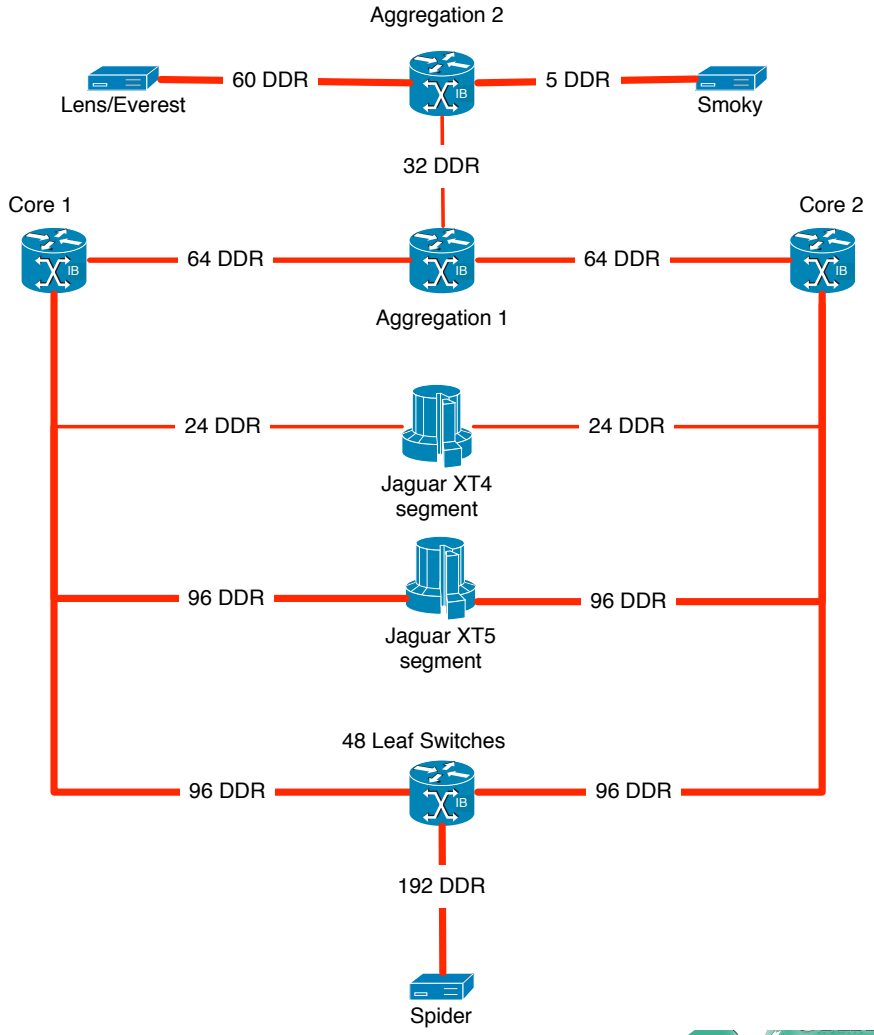
- Developed and maintained by CFS, then Sun, now Oracle
- POSIX compliant, open source parallel file system, driven by DOE Labs
- Metadata server (MDS) manages namespace
- Object storage server (OSS) manages Object storage targets (OST)
- OST manages block devices
  - *ldiskfs* on OSTs
    - V. 1.6 → superset of ext3
    - V. 1.8 + → superset of ext3 or ext4
- High-performance
  - Parallelism by object striping
- Highly scalable
- Tuned for parallel block I/O



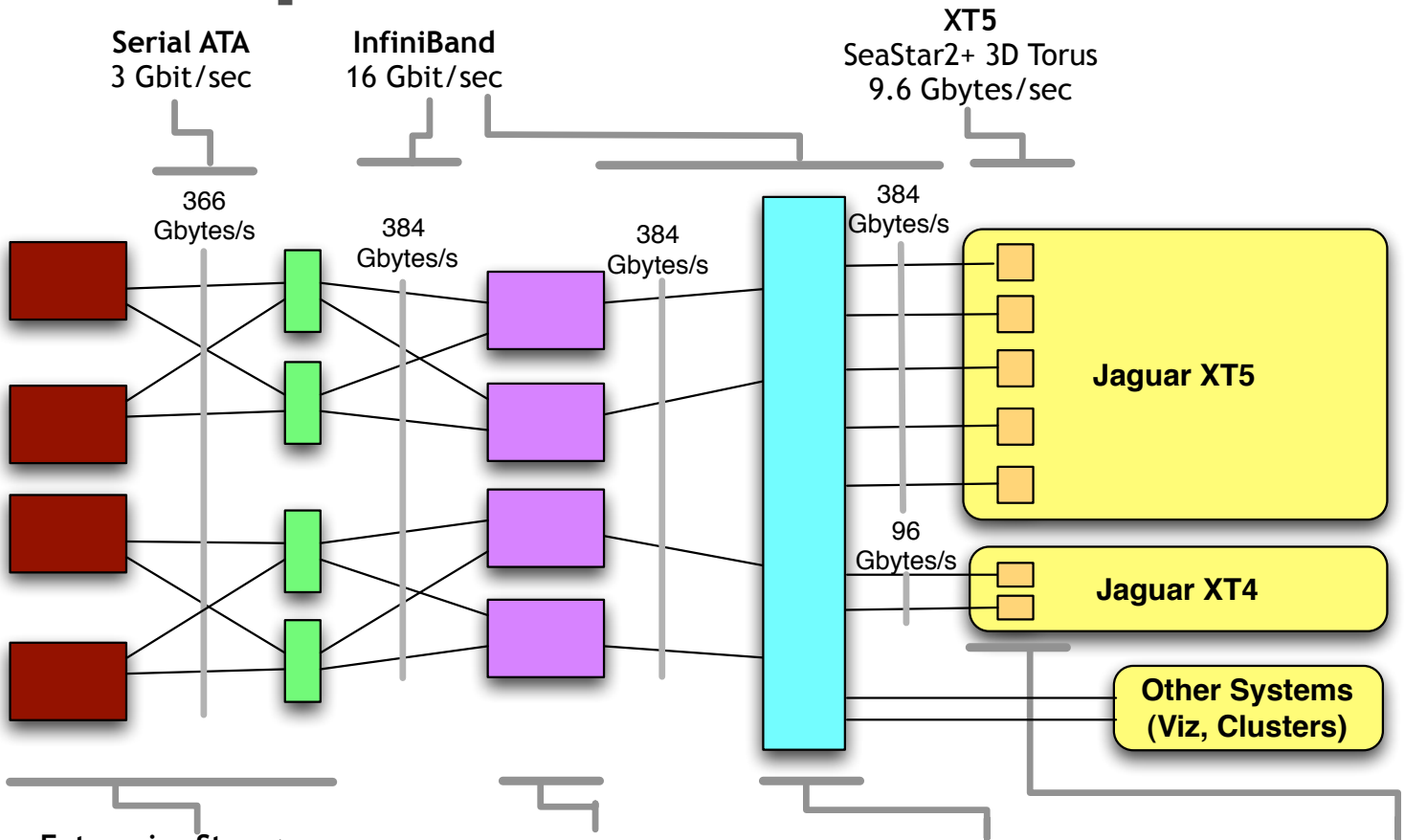
Spider - Overview

- Currently providing high-performance scratch space to all major OLCF platforms





Spider - Speeds and Feeds



Enterprise Storage
controllers and large racks of disks are connected via InfiniBand.

48 DataDirect S2A9900 controller pairs with 1 Tbyte drives and 4 InifiniBand connections per pair

Storage Nodes
run parallel file system software and manage incoming FS traffic.

192 dual quad core Xeon servers with 16 Gbytes of RAM each

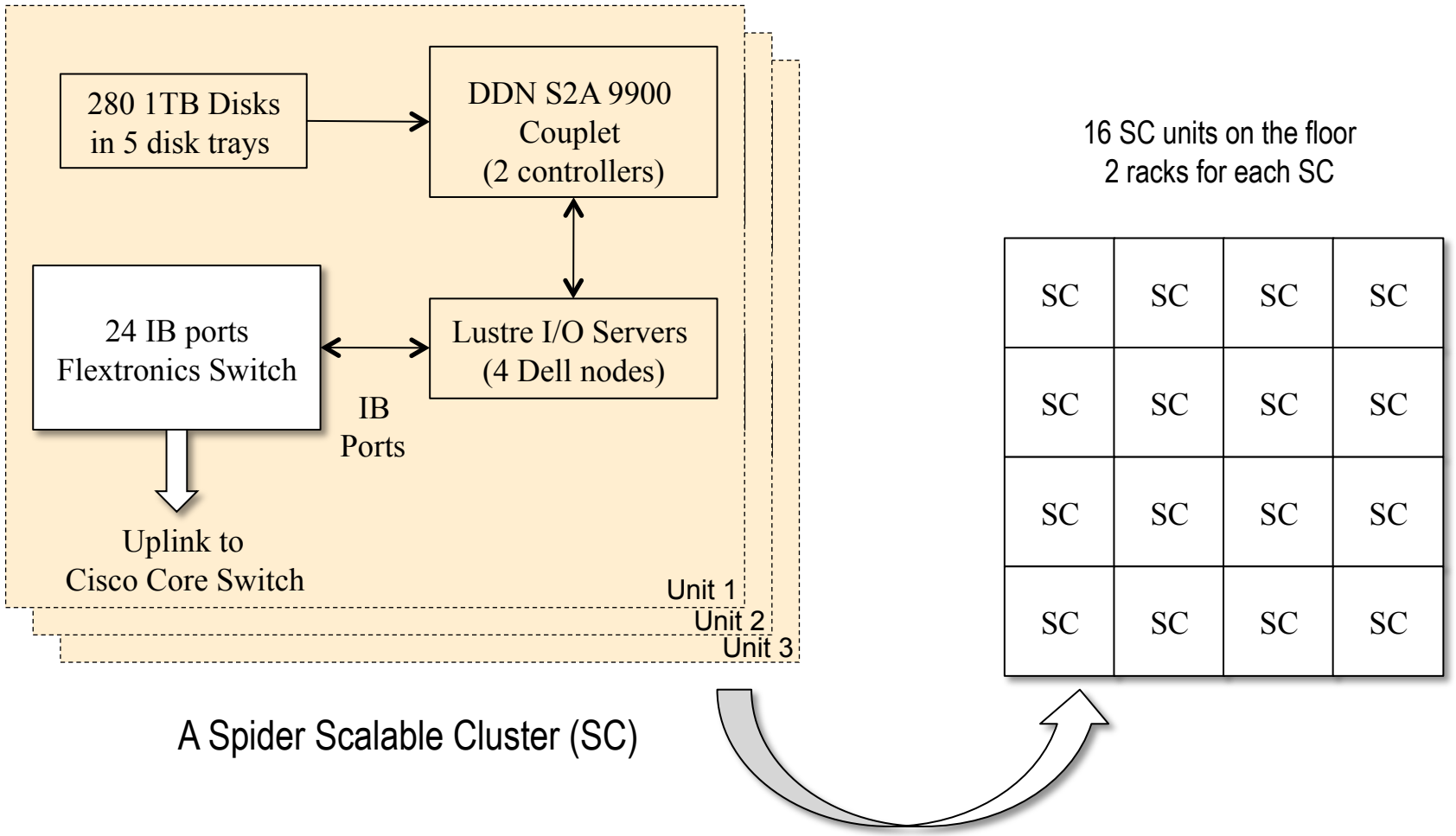
SION Network
provides connectivity between OLCF resources and primarily carries storage traffic.

3000+ port 16 Gbit/sec InfiniBand switch complex

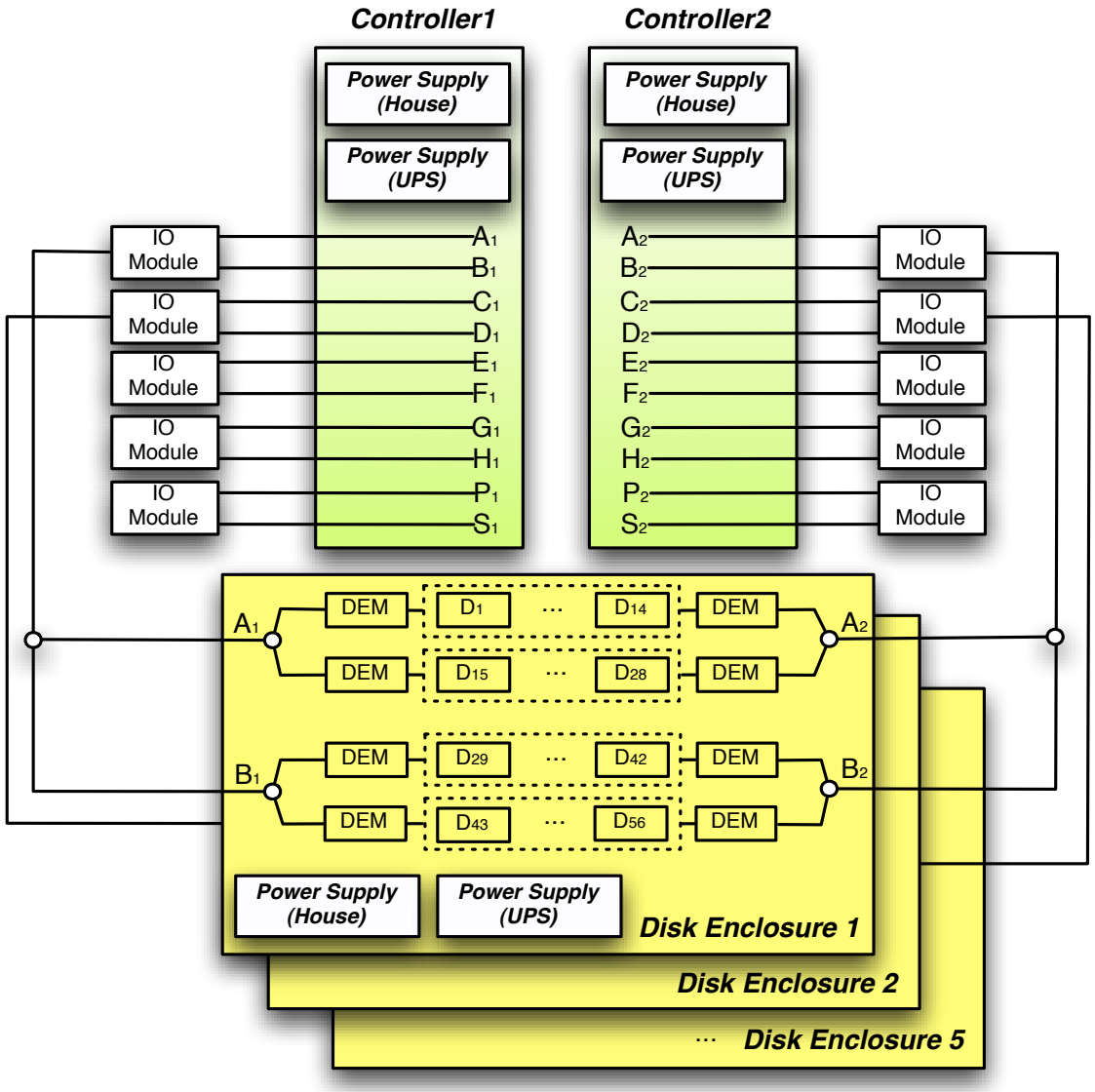
Lustre Router Nodes
run parallel file system client software and forward I/O operations from HPC clients.

192 (XT5) and 48 (XT4) one dual core Opteron nodes with 8 GB of RAM each

Spider - Couplet and Scalable Cluster

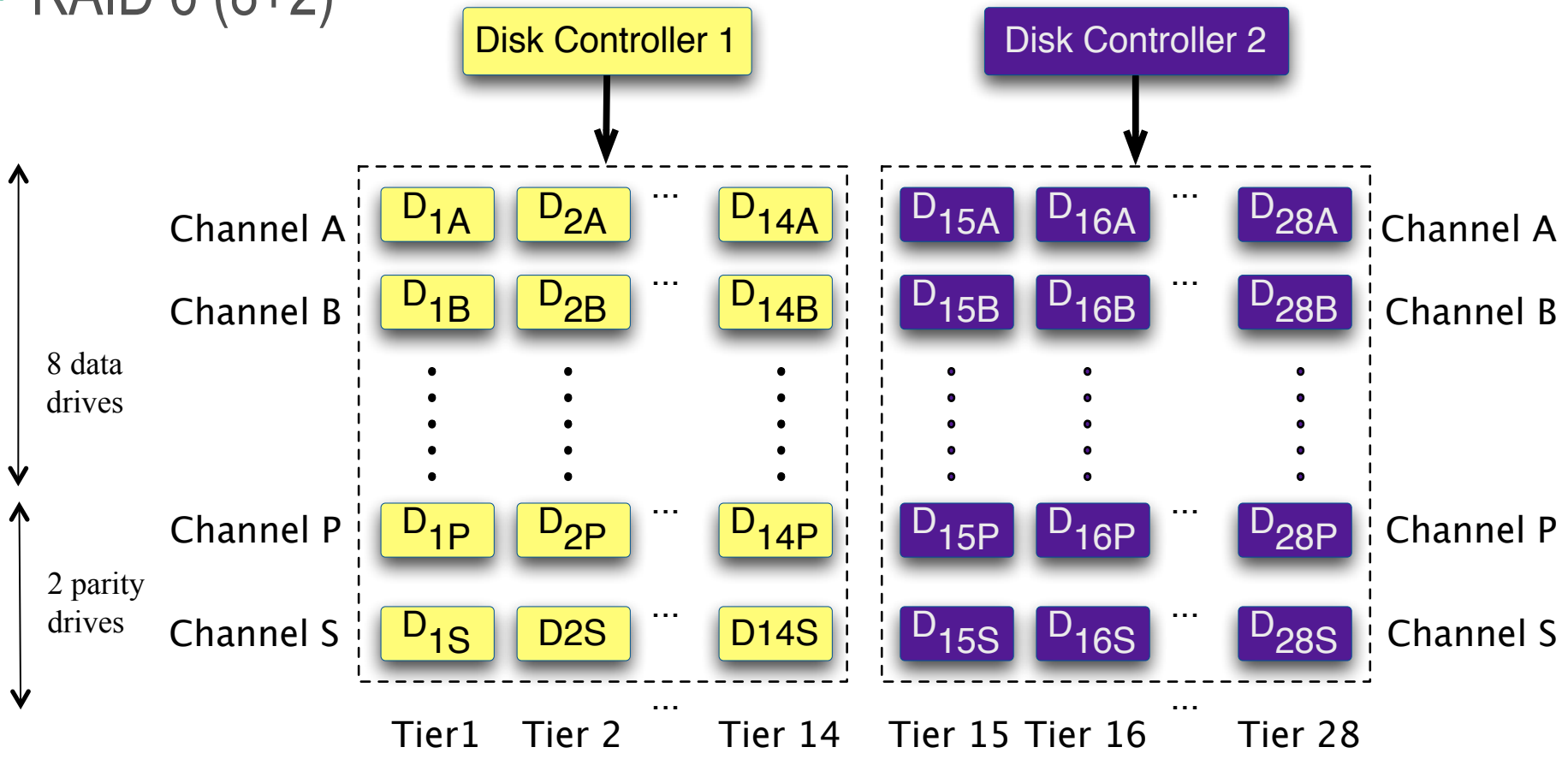


Spider - DDN S2A9900 Couplet



Spider - DDN S2A9900 (cont'd)

- RAID 6 (8+2)

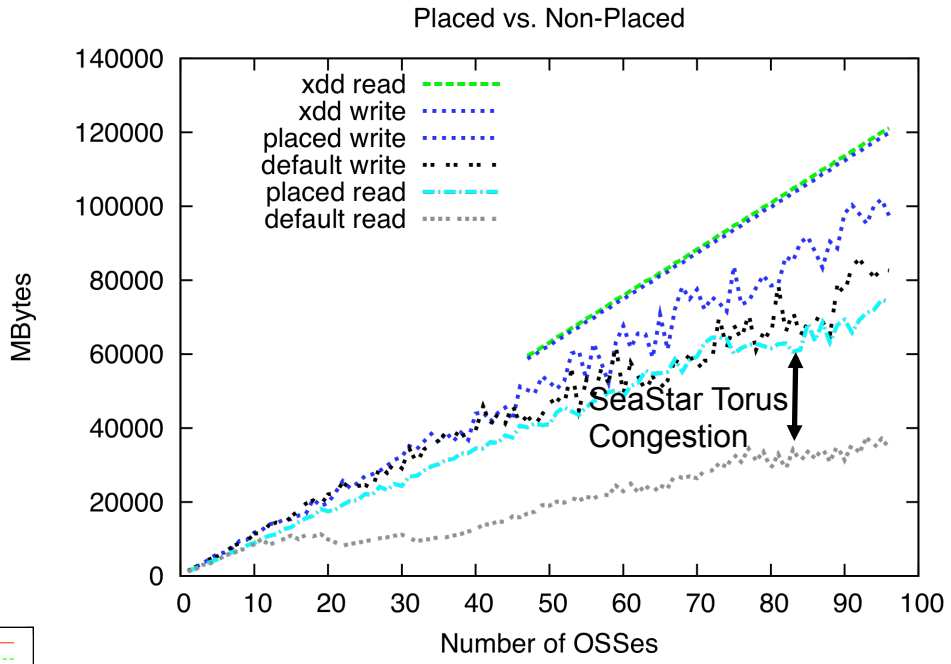


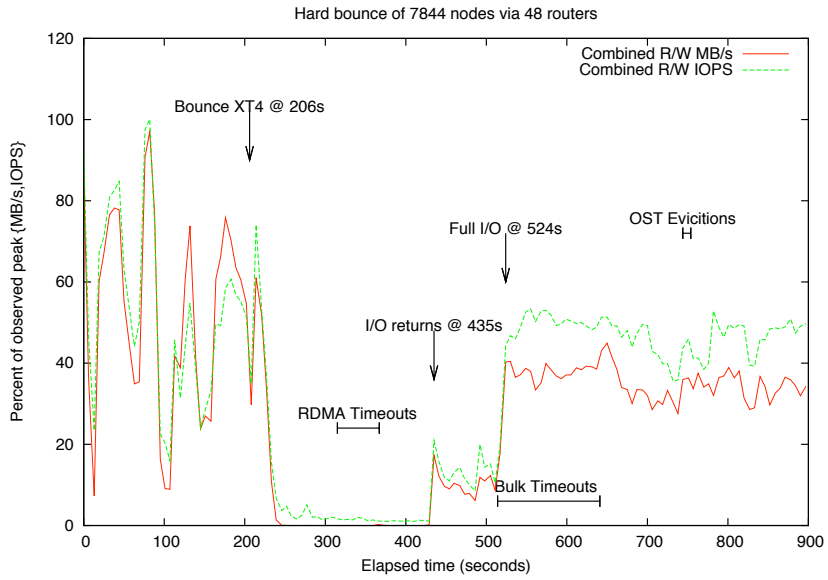
Spider - How Did We Get Here?

- 4 years project
- We didn't just pick up phone and order a center-wide file system
  - No single vendor could deliver this system
  - Trail blazing was required
- Collaborative effort was key to success
  - ORNL
  - Cray
  - DDN
  - Cisco
  - CFS, SUN, and now Oracle

Spider – Solved Technical Challenges

- Performance
  - Asynchronous journaling
  - Network congestion avoidance
- Scalability
  - 26,000 file system clients





- Fault tolerance design
  - Network
  - I/O servers
  - Storage arrays
- Infiniband support on XT SIO

ldiskfs Journaling Overhead

- Even sequential writes exhibit random I/O behavior due to journaling
  - Observed 4-8 KB writes along with 1 MB sequential writes on DDNs
  - DDN S2A9900's are not well tuned for small I/O access
  - For enhanced reliability write-back cache on DDNs are turned off
- Special file (contiguous block space) reserved for journaling on ldiskfs
  - Labeled as journal device
  - Beginning on physical disk layout
- Ordered mode
  - After file data portion committed on disk → journal meta data portion needs to be committed
- *Extra head seek needed for every journal transaction commit!*

Idiskfs Journaling Overhead (Cont'd)

- Block level benchmarking (writes) for 28 tiers → 5608.15 MB/s (baseline)
- File system level benchmark (*obdfilter*) gives 1398.99 MB/s
  - 24.9% of baseline bandwidth
  - One couplet, 4 OSS each with 7 OSTs
  - 28 clients, one-to-one mapping with OSTs

- Analysis
  - Large number of 4KB writes in addition to 1MB writes
  - Traced back to *ldiskfs* journal updates

		Read	Write
Single LUN	Sequential	685.62	235.45
	Random	101.74	96.77
28 LUN	Sequential	5570.15	5608.15
	Random	2753.87	2530.5

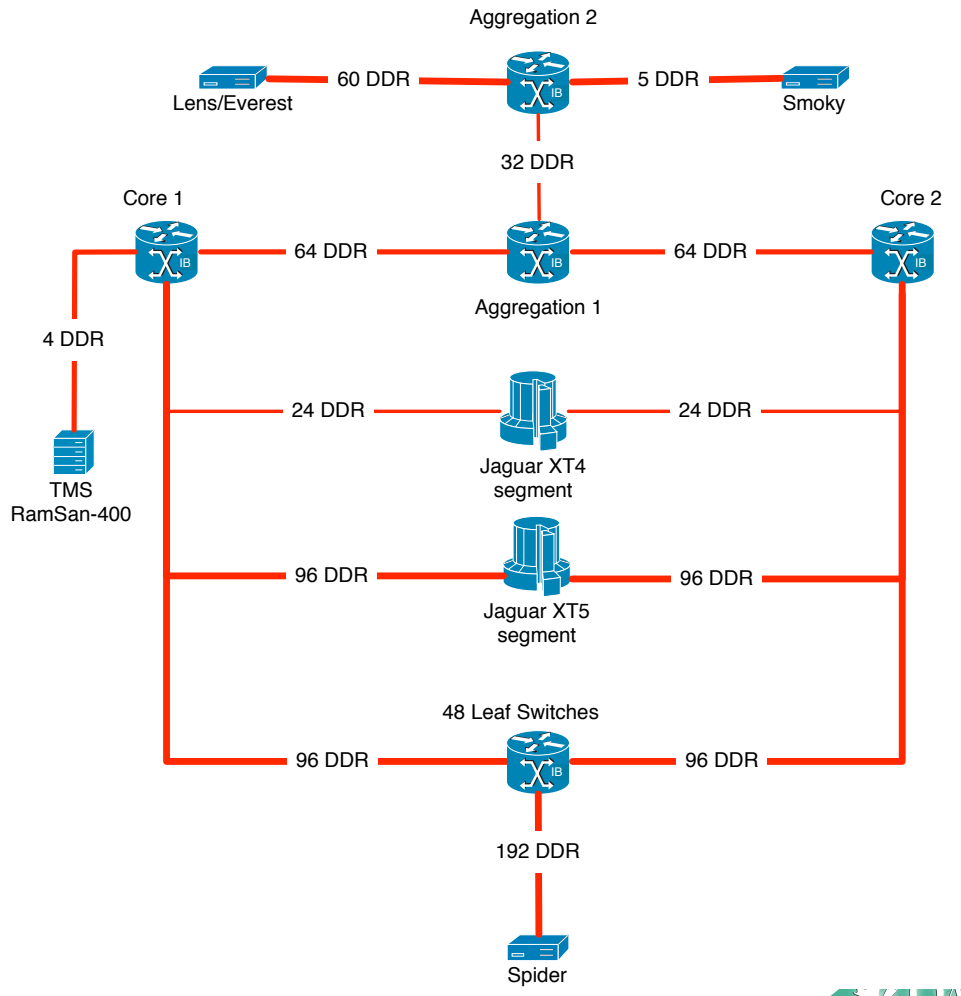
Minimizing extra disk head seeks

- Hardware solutions
  - External journal on an internal SAS tier
  - External journal on a network attached solid state device
- Software solution
  - Asynchronous journal commits

Configuration	Bandwidth MB/s (single couplet)	Delta % from baseline
Block level (28 tiers)	5608.15	0%
Internal journals, SATA	1398.99	**24.9%**
External, internal SAS tier	1978.82	35.2%
External, sync to RAMSAN, solid state	3292.60	58.7%
Internal, async journals, SATA	5222.95	**93.1%**

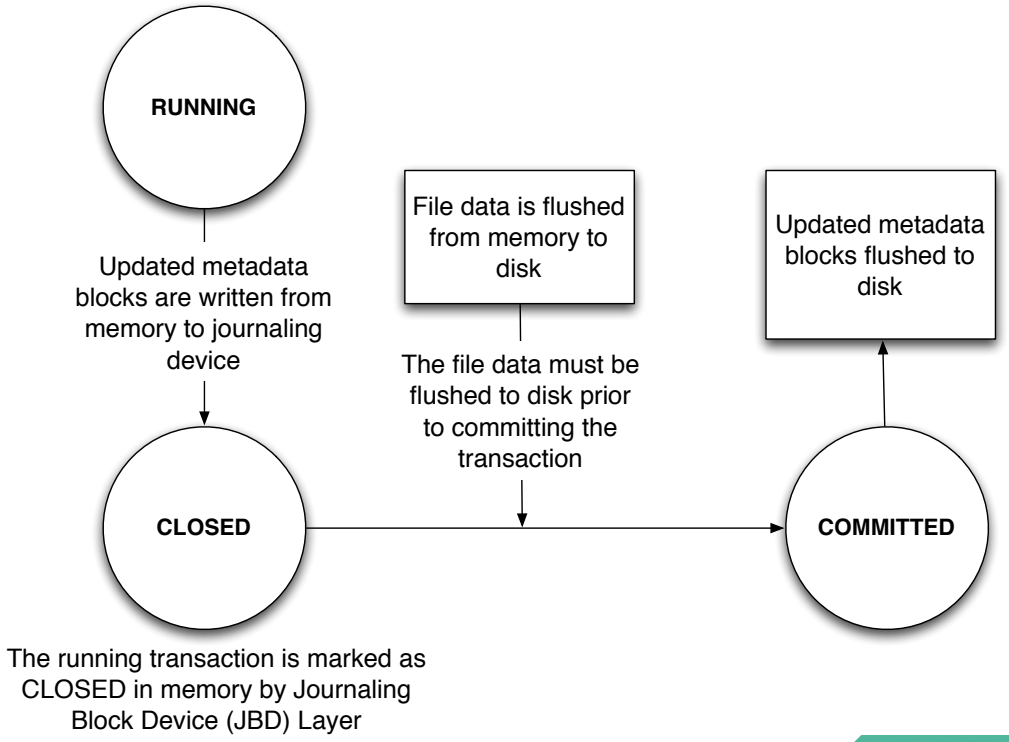
External journals on a solid state device

- Texas Memory Systems' RamSan-400
  - Loaned by Vion Corp.
  - Non-volatile SSD
  - 3 GB/s block I/O
  - 400,000 IOPS
  - 4 IB DDR ports w/ SRP
- 28 LUNs
  - One-to-one mapping with DDN LUNs
  - Obtained 58.7% of baseline performance
  - Network round-trip latency or inefficiency on external journal code path might culprit



Synchronous Journal Commits

- Running and closed transactions
  - *Running transaction* accepts new threads to join in and has all its data in memory
  - *Closed transaction* starts flushing updated metadata to journaling device. After flush is complete, the transaction state is marked as committed
  - *Current running* transaction can't be closed and committed until closed transaction fully commits to journaling device
- Congestion points
  - Slow disk
  - Journal size (1/4 of journal device)
  - Extra disk head seek for journal transaction commit
  - Write I/O operation for new threads is blocked on currently closed transaction that is committing



Asynchronous Journal Commits

- Change *how Lustre uses the journal*, *not the operation of journal*
- Every server RPC reply has a special field (*default*, *sync*)
  - id of the last transaction on stable storage
  - Client uses this to keep a list of completed, but not committed operations
  - In case of a server crash these could be resent (replayed) to the server
- Clients pin dirty and flushed pages to memory (*default, sync*)
  - Released only when server acks these are committed to stable storage
- Relax the commit sequence (*async*)
  - Add *async* flag to the RPC
  - Reply clients immediately after file data portion of RPC is committed to disk

Asynchronous Journal Commits (cont'd)

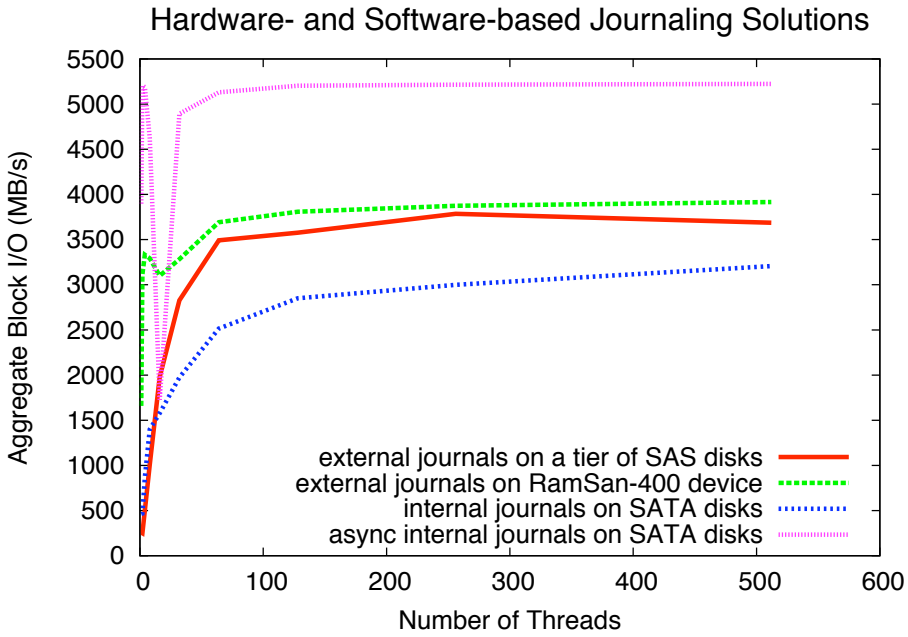
1. Server gets destination object id and offset for write operation
2. Server allocates necessary number of pages in memory and fetch data from remote client into pages
3. Server opens a transaction on the back-end file system.
4. Server updates file metadata, allocates blocks and extends file size
5. Server closes transaction handle
6. Server writes pages with file data to disk synchronously
7. If *async* flag set → server completes operation *asynchronously*
   - Server sends a reply to client
   - JBD flushes updated metadata blocks to journaling device, writes commit record
8. If *async* flag is *NOT* set → server completes operation *synchronously*
   - JBD flushes updated metadata blocks to journaling device, writes commit record
   - Server sends a reply to client

Asynchronous Journal Commits (cont'd)

- Async journaling achieves 5,223 MB/sec (at file system level) or 93% of baseline

- Cost effective
  - Requires only Lustre code change
  - Easy to implement and maintain

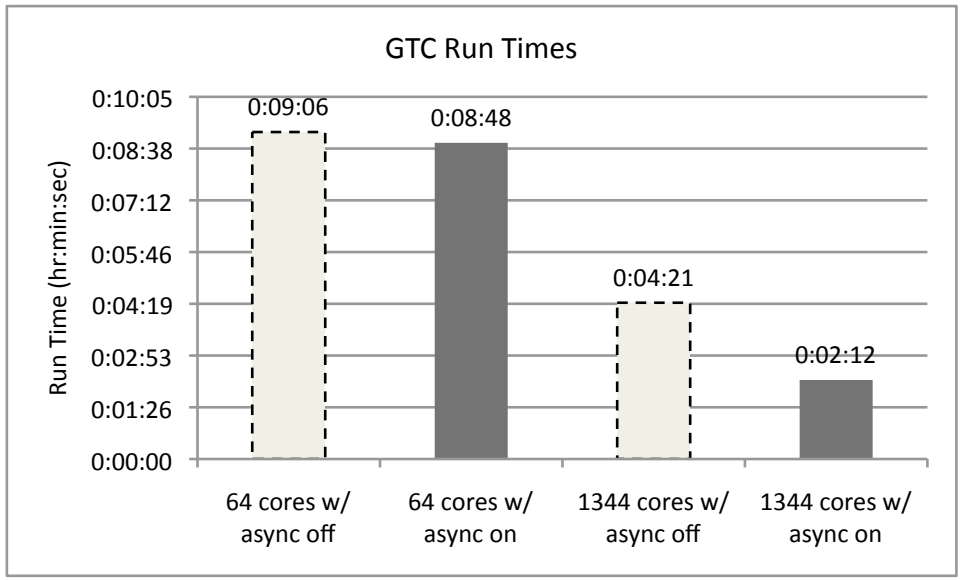
- Temporarily increases client memory consumption
  - Clients have to keep more data in memory until the server acks the commit



- Does not change the failure semantics or reliability characteristics
  - The guarantees about file system consistency at the local OST remain unchanged

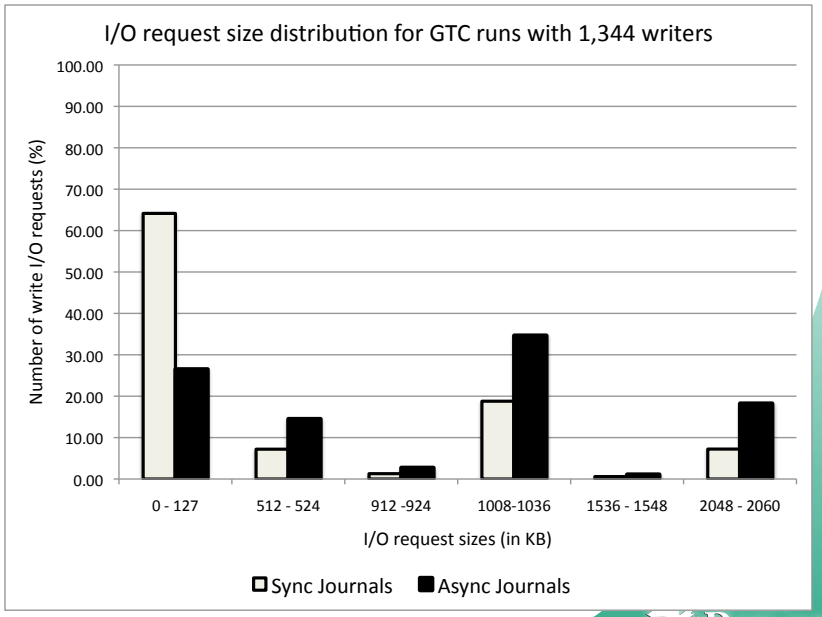
Asynchronous Journal Commits Application Performance

- Gyrokinetic Toroidal Code (GTC)



- Up to 50% reduction in runtime
  - Might not be typical
  - Depends on the application

- Reduced number of small I/O requests
  - 64% to 26.5%



Conclusions

- A system at this scale, we can't just pick up the phone and order one
- No problem is small when you scale it up
- At Lustre file system level we obtained 24.9% of our baseline block level performance
  - Tracked to *ldiskfs* journal updates
- Solutions
  - External journals on an internal SAS tier; achieved 35.2%
  - External journals on network attached SSD; achieved 58.7%
  - Asynchronous journal commits; achieved 93.1%
    - Removed a bottleneck from critical write path
    - Decreased 4 KB I/O DDNs observed by 37.5%
    - Cost-effective, easy to deploy and maintain
    - Temporarily increases client memory consumption
    - Doesn't change failure characteristics or semantics

Questions?

Contact info

Sarp Oral
oralhs at ornl dot gov
Technology Integration Group
Oak Ridge Leadership Computing Facility
Oak Ridge National Laboratory