

Terascale I/O Solutions

Nathan Stone

Pittsburgh Supercomputing Center

Terascale Performance Analysis Workshop

Specific Solutions

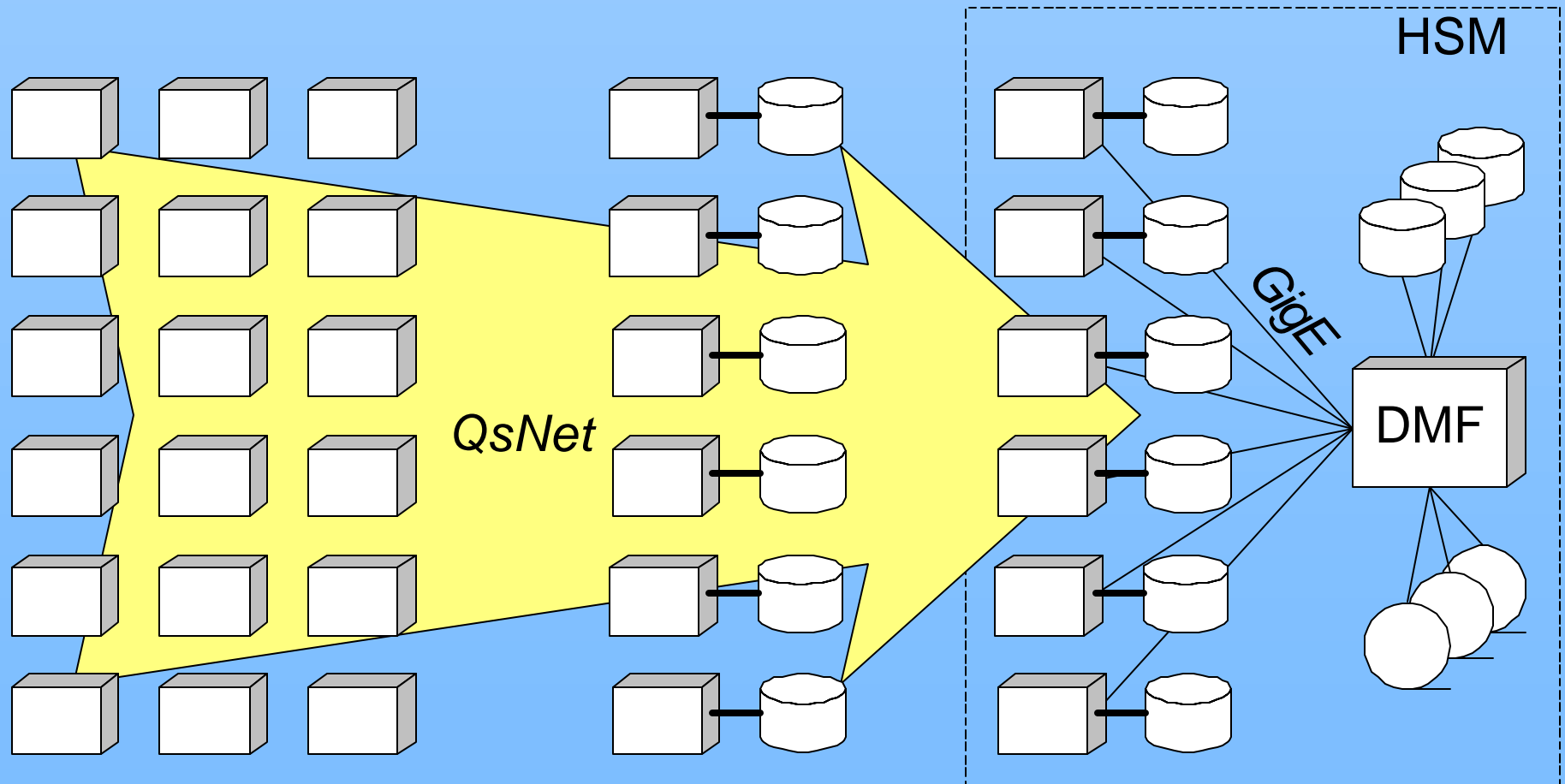
- TCSCOMM
 - High-performance communication library
- TCSIO
 - High-performance file-migration infrastructure
- CPR
 - Automated application-level checkpoint/recovery
- HSM
 - A new/novel hardware infrastructure
- SLASH
 - Distributed cache management and HSM gateway

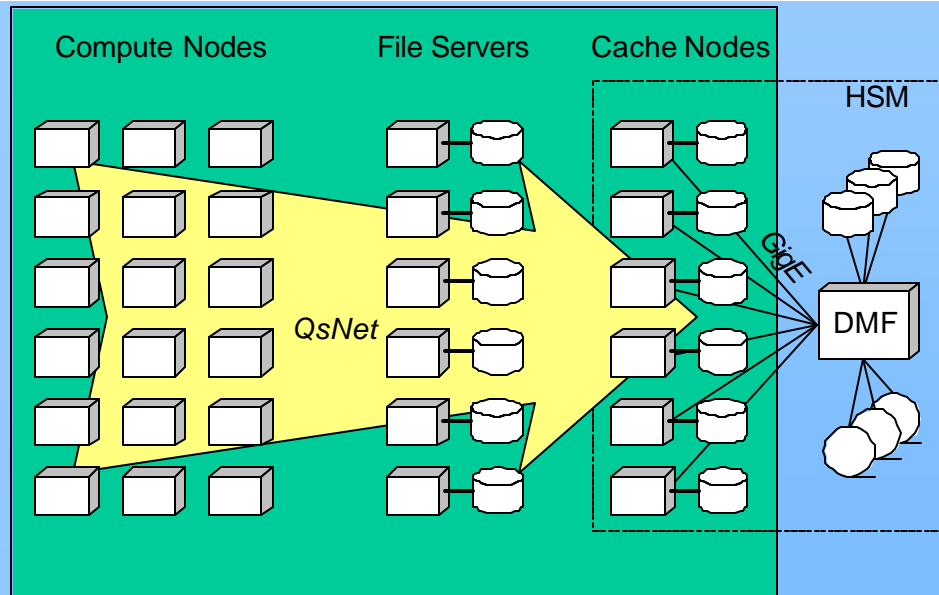
Infrastructure Domains

Compute Nodes

File Servers

Cache Nodes



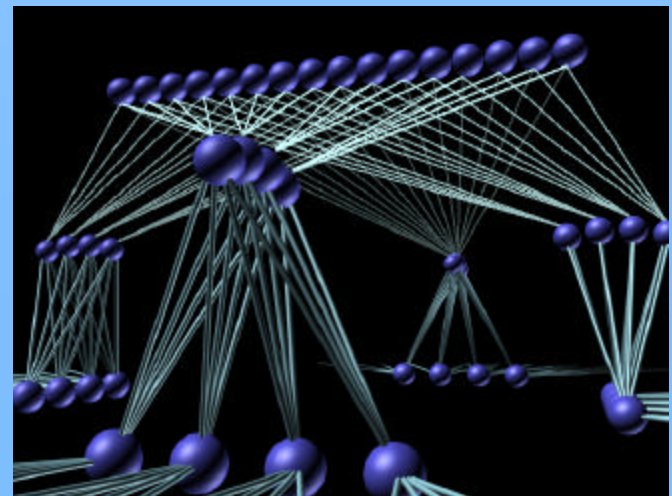


TCSCOMM

High-performance communication library

The QsNet Interconnect

- High-bandwidth, low-latency network
 - 2 major components:
 - The Elan network cards
 - ELAN3 = copper
 - The Elite switch cards
 - Full “fat-tree” topology
 - non-contending paths
 - Full bi-sectional bandwidth
 - 250 MB/s at 5 us latency per “rail”



The Elan System Libraries

- Two system libraries are available for using the Elan cards:
 - The libelan library:
 - higher level – **used by MPI**
 - requires RMS to set memory layout, contexts/capabilities
 - The libelan3 library:
 - lower level – **used by TCSCOMM**
 - applications can create custom contexts/capabilities, memory layouts.

TCSCOMM: Motivation

- To use the QsNet for non-MPI processes
 - running *outside of RMS*
- TCSCOMM must use *libelan3* in order to:
 - Create custom “capabilities”
 - Provide heterogenous communications
 - Different hardware architectures and memory layouts
 - Tru64 and Linux (IA-32 and IA-64)
 - Be as fast as possible

TCSCOMM Design

- Two-stage method to transfer data:
 - **Queued DMA**
 - small packets
 - used to send data transfer requests
 - from sender to receiver
 - **getdma**
 - used to transfer large blocks of memory
 - Invoked by the receiver
 - *i.e.* the receiver synchronizes the data transfers

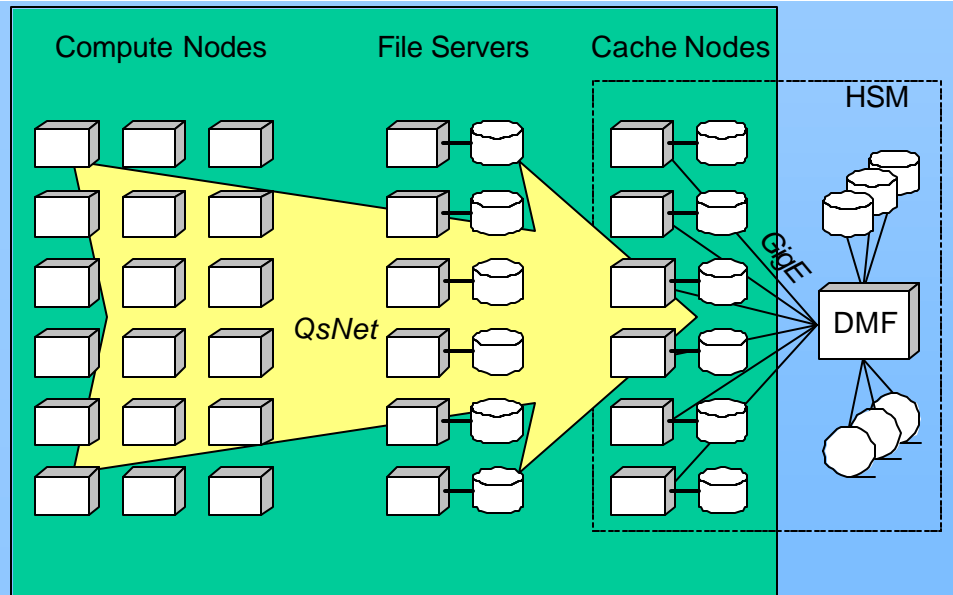
TCSCOMM: Performance

- TCSCOMM **load-balances** across multiple “rails” of QsNet networks
- We routinely observe **400 MB/sec** transfer rates
 - For heterogenous transfers, we observe
 - **360 MB/sec** for dual rail
 - **200 MB/sec** for single rail

TCSCOMM-Enabled Applications

Distributed Rendering –Joel Welling (PSC)

- A network layer for the WireGL and Chromium packages based on TCSCOMM (which will be used in the same role as existing TCP/IP and Myrinet network layers)
 - Running in a heterogeneous environment
 - calculations on Tru64 compute nodes, rendering on IA32 workstations.
 - High-performance visualization cluster!



TCSIO

*High-performance file-migration
infrastructure*

TCSIO: Motivation

- Advanced features
 - *e.g.* Checkpoint/Recovery
- 3rd party operations
 - Run I/O tasks while applications resume computation
- High-throughput file migration service
- Bypass file systems
 - Performance was/is an issue in systems of this size
- Maximize disk resource utilization

TCSIO: Implementation Details

- “I/O Daemons”
 - Wherever disk resources are shared
- Command-line clients
 - tcscp, tcls, tcsrm, ...
- C/C++/Fortran API
 - Delivers advanced features to user apps, if desired
- Configuration files
 - For dynamic control of basic features

TCSIO: Interface

- We tried to make it as simple as possible...

```
>tcscp input.dat iam300:/local/
```

(a la “rcp”)

- But we wanted to provide some advanced features:

But of course,

```
> ,tcscp input_{rank}.dat {compute}:/local/
```

we're not the first

one to think of (PE rank substitution, host aliasing, ...)

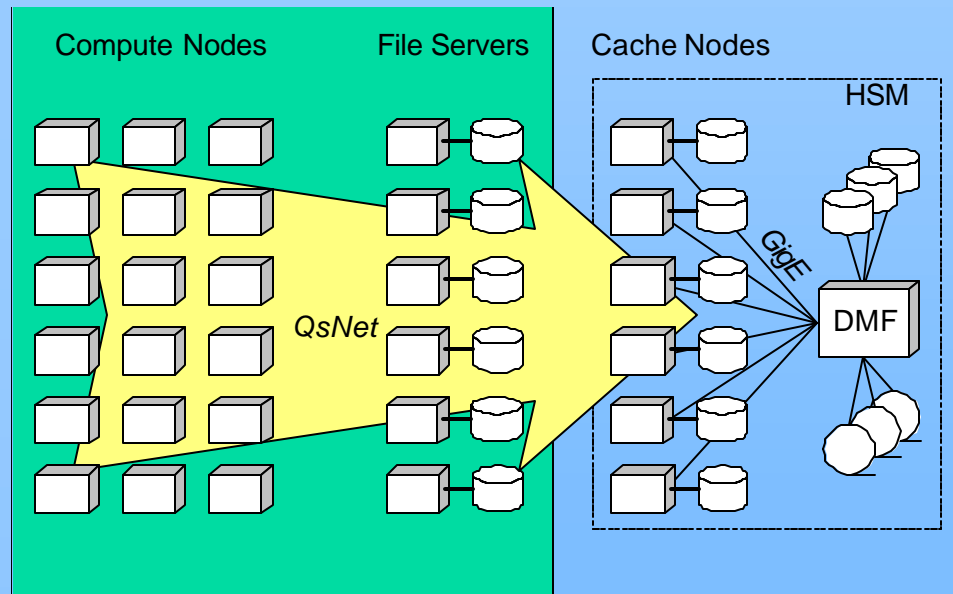
simplicity...



The “Microsoft Keyboard”

TCSIO: Performance

- Measuring aggregate performance to file servers
 - compute node performance is disk-limited
- 2.6/2.3 GB/sec aggregate read/write
(3.6 GB/sec read on a “quiet” machine)
 - measured for 1024 PEs writing a 100 MB file



CPR

*Automated application-level
checkpoint/recovery*

CPR: Motivation

- Large cluster (750+ compute nodes)
- Large (1000+ PEs) long-running (measured in days) jobs
- Simple statistics
 - Single-node MTTF ~ 1 yr implies 2 hardware failures per day

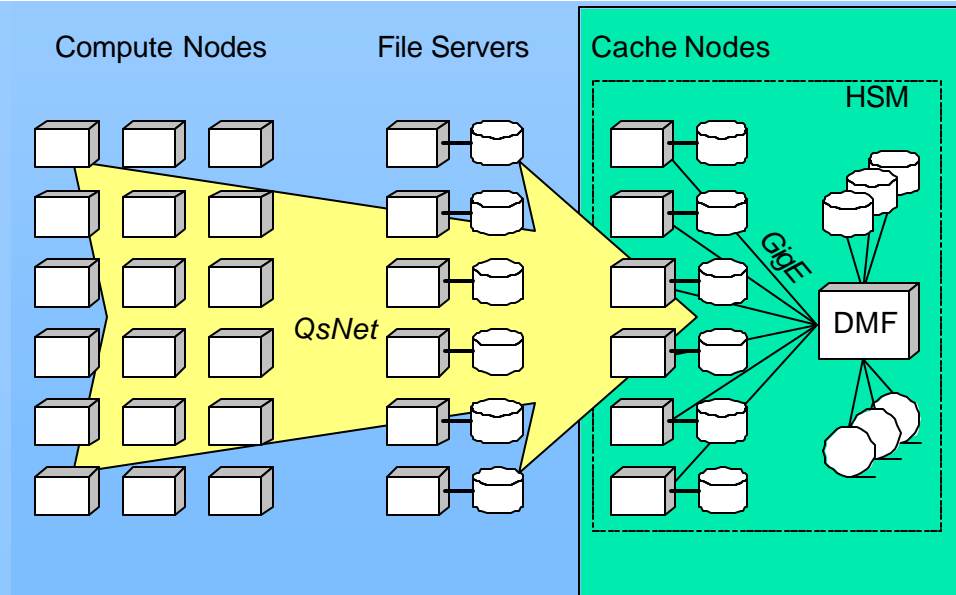
CPR: Details

- Application-level CPR
 - Users must modify source-code
 - Have several simple and computational examples, and a few “early-adopters”
- Requires:
 - Full TCSIO system
 - Central DB (for CPR state tracking)
 - CPR user library – to link into users’ applications
 - Scheduling, accounting, and “event system” integration
 - Necessary for automated restart and recovery

CPR: Integration

- Event Handling system
 - Cluster resource that “notices” hardware failures
 - RMS, on AlphaServer SC
 - Scheduling or other custom system on other resources
 - Notifies the CPR system of node failure
- Scheduling
 - PBS “epilogue” script
 - Checks whether a job is using CPR, failed “abnormally”, and deserves to be restarted
 - Re-queues the job automatically
- Accounting
 - Credit back “lost” computing time, following last complete checkpoint

...in production since 2001

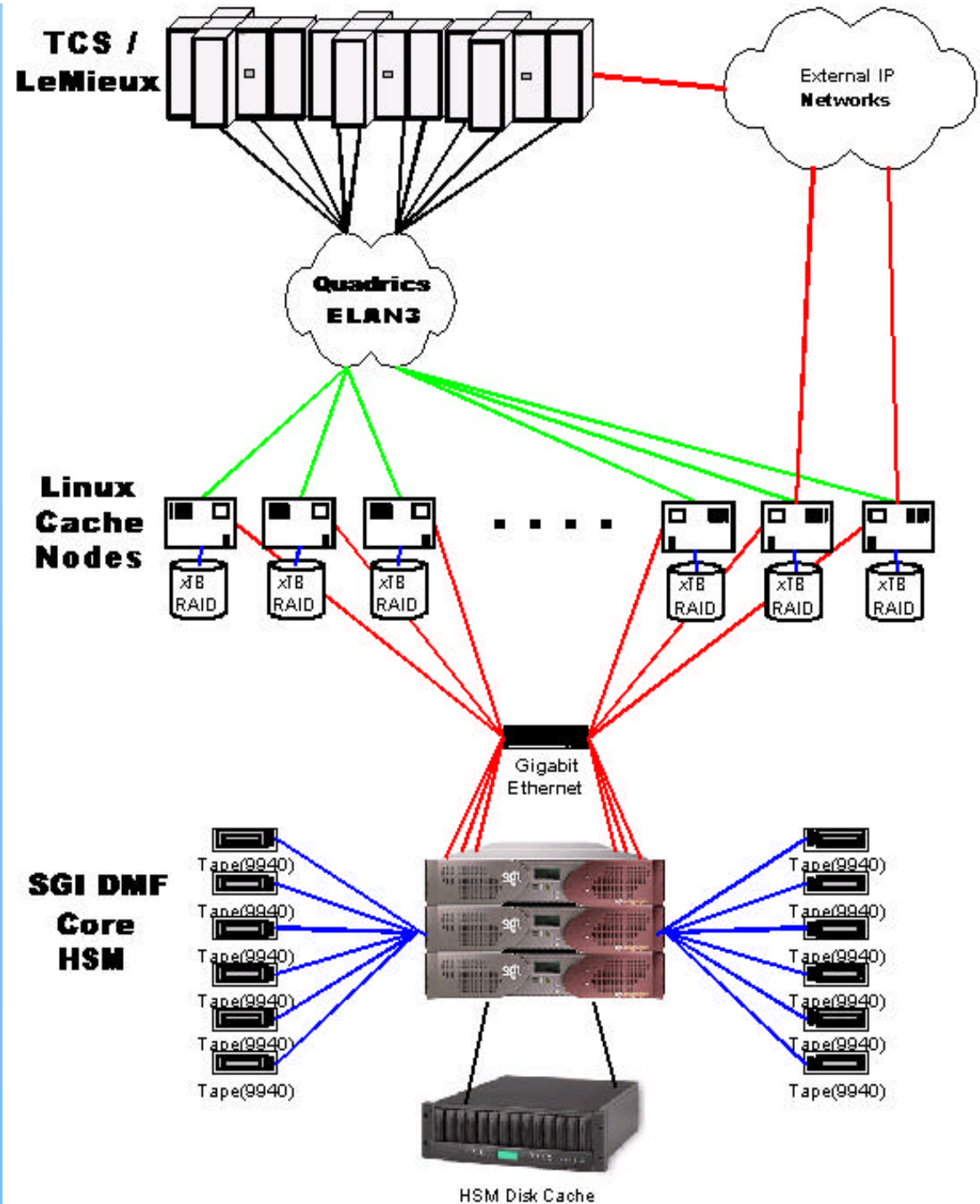


Hierarchical Storage Management

A new/novel hardware infrastructure

HSM: Hardware & Connectivity

- “Core” HSM is SGI/DMF on O300
 - 6-20 TB local disk cache
 - 12 STK 9940B drives
 - 200 GB @ 28 MB/sec
- LCNs are Linux IA32 with 6+ TB of RAIDed cache storage



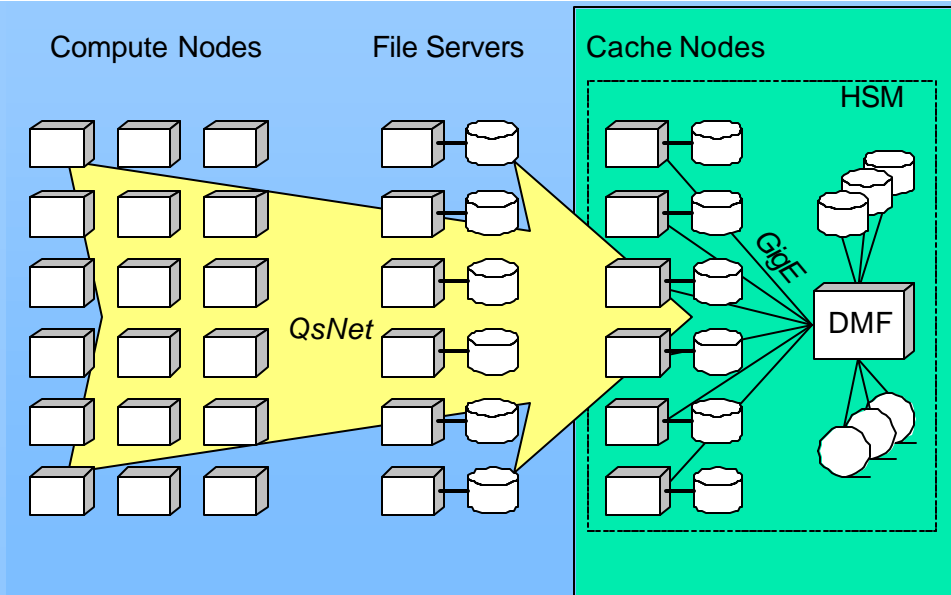
HSM: Motivation

2-layer design provides:

- Full-functionality HSM “core” (**no** user access)
 - DMF tape management
 - XFS file system: low-level file metadata
- Linux Cache Nodes (LCNs)
 - Distributed load
 - Much larger disk caches (xTB) per node
 - Multiple parallel nodes
 - Scalable
 - Cheap!
- Doesn't cost much when retrieving files from tape
- Saves **a lot** when retrieving files from disk!

HSM: Client Applications

- All platforms
 - KFTP (GigE)
 - SCP/SFTP(GigE)
- TCS Specific
 - TCSIO (ELAN3)
- Cray Specific
 - RCP (HIPPI)
- Other PSC systems
 - FAR (File ARchiver utility)



SLASH

Distributed cache management and HSM gateway

SLASH Defined...

- SLASH:
 - Scalable Lightweight Archival Storage Hierarchy
- Assumes a distributed cache
- Throttles archival
 - Not based on user access, but on management daemons
- Arbitrates multiple shared storage systems

SLASH: Details

- SLASH presents a very simple API to applications that access HSM data (*e.g.* TCSIO, FTPd, etc.)
 - `int hsmCacheDownload(...)`
 - `int hsmCacheUploadInit(...)`
 - `int hsmCacheUploadComplete(...)` – for success
 - `int hsmCacheUploadCleanup(...)` – for failure
- Internally, SLASH uses a system of symbolic links to track uploaded files and their state.
- SLASH has a few daemons that asynchronously migrate and/or verify the status of uploaded files, including collision arbitration

HSM Procedure: File Upload

1. Remote user client application initiates communication to **round-robin** allocated LCN
 2. LCN creates or updates a file in the “core” (XFS) file system modifying **extended attributes** via RPC
 3. File is uploaded to a temporary area on LCN
 4. File is asynchronously sent to “core” file system
- **Collisions** from multiple LCNs decided by last version written (based on start of upload)

HSM Procedure: File Download

1. Remote user client application initiates communication to round-robin allocated LCN
 2. “Primary” LCN (the one contacted) determines which LCN (if any) has cached copy of file
 - A. If Primary LCN has the file, it sends it.
 - B. If no LCN copy is available, file is recalled (via NFS over GigE) from the DMF server.
 - C. If another LCN has the file:
 - i. If using TCSIO, 3rd party transfer begins
 - ii. If not TCSIO, file is pulled from “Secondary” LCN via NFS over GigE.
- DMF may transparently stage file from tape if not already on DMF disk cache.

Questions?

Nathan Stone

<nstone@psc.edu>

<http://www.psc.edu/~nstone/>

PSC Advanced Systems Group

http://www.psc.edu/advanced_systems/

Whitepapers for ongoing work at PSC

http://www.psc.edu/publications/tech_reports/