Robust Storage Management in the Machine Room and Beyond

Presented by

Sudharshan Vazhkudai

Computer Science Research Group
Computer Science and Mathematics Division

In collaboration with

ORNL: John Cobb, Greg Pike
North Carolina State University: Xiaosong Ma, Zhe Zhang, Chao Wang, Frank Mueller
The University of British Columbia: Matei Ripeanu, Samer Al Kiswany
Virginia Tech: Ali Butt

SC07

OAK RIDGE National Laboratory

Problem space: Petascale storage crisis

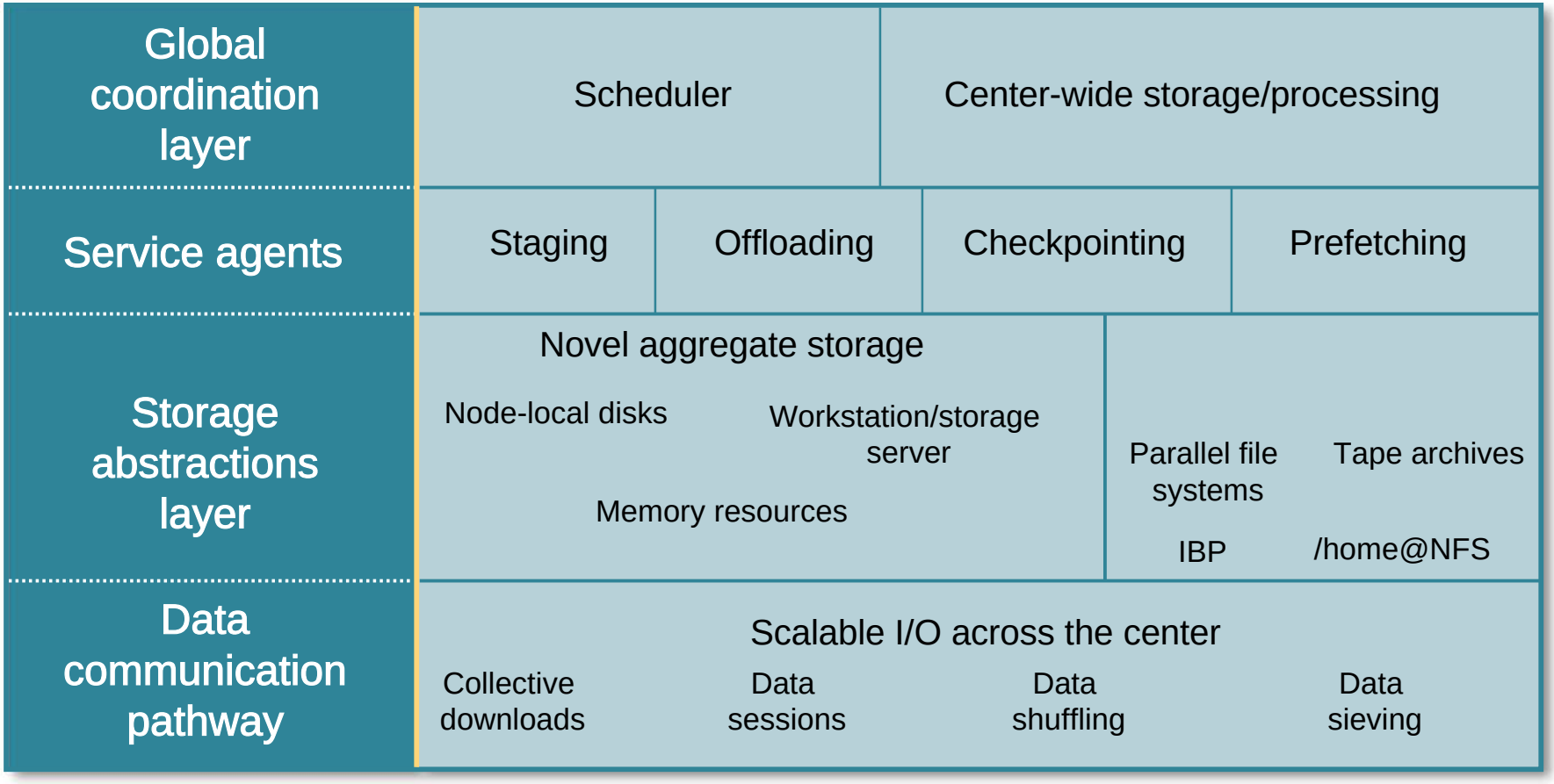
- Data staging, offloading, and checkpointing are all affected by data unavailability and I/O bandwidth bottleneck issues:
  - Compute time wasted on staging at the beginning of the job.
  - Early staging and late offloading waste scratch space.
  - Delayed offloading renders result data vulnerable to purging.
  - Checkpointing terabytes of data to a traditional file system results in an I/O bottleneck.
  - Storage failure:
    - Significant contributor to system downtime and CPU underutilization (during RAID reconstruction).
    - Failures per year: 3–7% disks, 3–16% controllers, and up to 12% SAN switches; 10x the rate expected from vendor specification sheets (J. Gray and C.V. Ingen, "Empirical measurements of disk failure rates and error rates," Technical Report MSR-TR-2005-166, Microsoft, December 2005.)!
  - Upshot:
    - Uptime low:
      - Due to job resubmissions.
      - Since checkpoints and restarts are expensive.
    - Increased job wait times due to staging/offloading and storage errors.
    - Poor end-user data delivery options.

Approach

- If you cannot afford a balanced system, develop management strategies to compensate.
- Exploit opportunities throughout the HEC I/O stack:
  - Parallel file system.
  - Many unused resources: Memory, cluster node-local storage, desktop idle storage (both in machine room and client-side).
  - Disparate storage entities including archives and remote sources.
- Concerted use of aforementioned:
  - Can be brought to bear upon urgent supercomputing issues, such as *staging, offloading, prefetching, checkpointing, data recovery, I/O bandwidth bottleneck, and end-user data delivery.*

Approach (cont'd.)

- View the entire HPC center as a system.
  - New ways to optimize this system's performance and availability.



Global coordination

- Motivation: Lack of global coordination between the storage hierarchy and system software.
- As a start, need coordination between staging, offloading, and computation:
  - Problems with manual and scripted staging:
    - Human operational cost, wasted compute time/storage, and increased wait time due to resubmissions.
  - How?
    - Explicit specification of I/O activities alongside computation in a job script.
    - Zero-charge data transfer queue.
    - Planning and orchestration.

Coordinating data and computation

- Specification of I/O activities in PBS job script:
  - BEGIN STAGEIN
    - retry=3; interval=20
    - hsi -A keytab -k MyKeytab -l user ‘‘get /scratch/user/Destination: Input’’
  - END STAGEIN
  - BEGIN COMPUTATION
    - #PBS...
  - END COMPUTATION
  - BEGIN STAGEOUT ... END STAGEOUT
- Separate data transfer queue: Zero charge:
  - Queuing up and scheduling data transfers.
  - Treats data transfers as “data jobs.”
  - Data transfers can now be charged, if need be!
- Planning and orchestration
  - Parsing into individual stage-in, compute, and stage-out jobs.
  - Dependency setup and management using resource manager primitives.

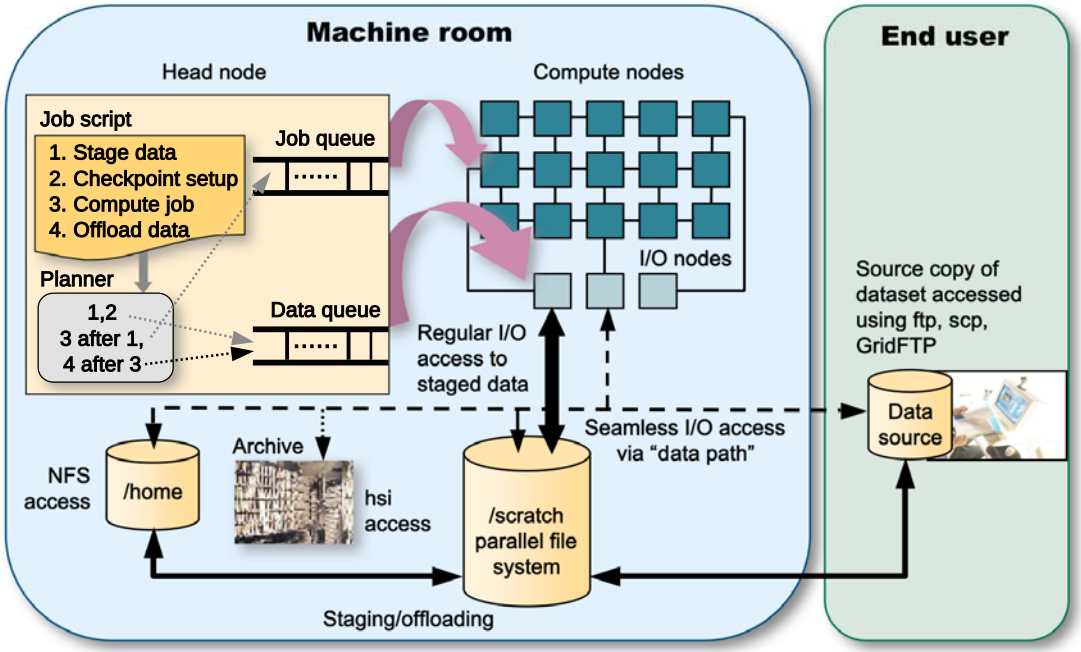
Seamless data path

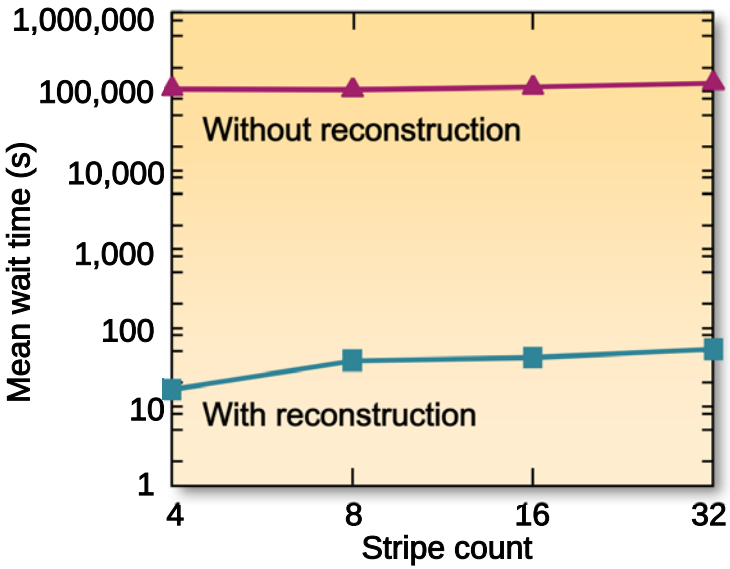
- Motivation: Standard data availability techniques designed with persistent data in mind:
  - RAID techniques can be time consuming; a 160-GB disk takes order of dozens of minutes.
  - Multiple disk failure within a RAID group can be crippling.
  - I/O node failovers are not always possible (thousands of nodes).
  - Need novel mechanisms to address "transient data availability" that complement existing approaches!
- What makes it feasible?
  - Natural data redundancy in the staged job data.
  - Job input data usually immutable.
  - Network costs drastically decreasing each year.
  - Better bulk transfer tools with support for partial data fetches.
- How?
  - Augmenting FS metadata with "recovery hints" from job script.
  - On-the-fly data reconstruction on another object storage target (OST).
  - Patching from data source.

Data recovery

- Embedding recovery metadata about transient job data into the Lustre parallel file system:
  - Extend Lustre metadata to include recovery hints.
  - Metadata extracted from job script.
  - “Source” and “sink” information becomes an integral part of transient data.
- Failure detection to check for unavailable OSTs.
- Reconstruction:
  - Locate substitute OSTs and allocate objects.
  - Mark data dirty in metadata directory service (MDS).
  - Recover URI from MDS.
  - Compute missing data range.
- Remote patching:
  - Reducing multiple authentication costs per dataset.
  - Automated interactive session with “Expect” for single sign-on.
  - Protocols: hsi, GridFTP, NFS.

Results





- Better use of users' compute time allocation and decreased job turnaround time.
- Optimal use of center's scratch space, avoiding too early stage in and delayed offloading.
- Reduces resubmissions due to result-data loss.
- Reduces wait time: Trace-driven simulation of LANL operational data + LCF scratch data.

New storage abstractions

- Checkpointing TB of data cumbersome; need better tools to address the storage bandwidth bottleneck.
- Options:
  - Machines can potentially be provisioned with solid-state memory.
  - ~ 200 TB of aggregate memory in the PF machine; potential "residual memory":
    - Tier 1 Apps (GTC, S3D, POP, CHIMERA) seldom use all memory.
    - Checkpoint size is ~ 10% of the memory footprint.
  - Many applications oversubscribe for processors in an attempt to plan for failure:
    - Use the oversubscribed processors!
- Checkpoint to memory can expedite these operations.
  - Previous solutions are not concerted in their memory usage, creating artificial load imbalance:
    - Examples: J.S. Plank, K. Li, and M.A. Puening, "Diskless Checkpointing," IEEE Transactions on Parallel and Distributed Systems, 1998. 9(10): p. 972-986. L.M. Silva, and J.G. Silva, "Using two-level stable storage for efficient checkpointing," IEE Proceedings - Software, 1998. 145(6): p. 198-202.

stdchk: A checkpoint-friendly storage

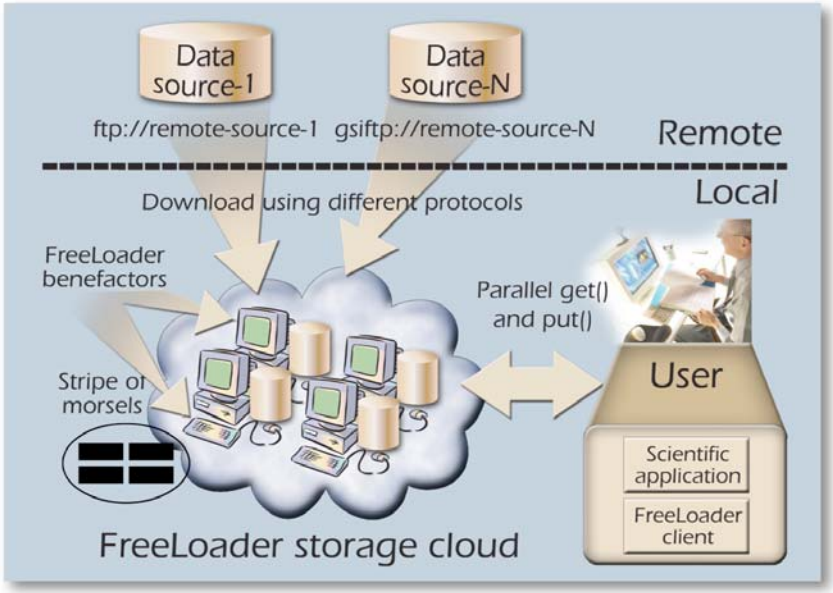
- Aggregate memory-based storage abstraction:
  - Split checkpoint images into chunks and stripe them.
  - Parallel I/O across distributed memory.
  - Redundantly mounted on PEs for FS-like access.
  - Optimized, relaxed POSIX I/O interfaces to the storage.
  - Lazy migration of images to local disk/archives, creating a seamless data pathway.
    - Unused/underutilized processors can perform this operation.
  - Incremental checkpointing and pruning of checkpoint files.
    - Compare chunk hashes from two successive intervals.
    - Initial experiments suggest a 10–25% reduction in size for BLCR checkpoints.
    - Purge images from previous interval once the current image is safely stored.
    - File system is unable to perform such optimizations.
  - Specification of checkpoint preparation in a job script.

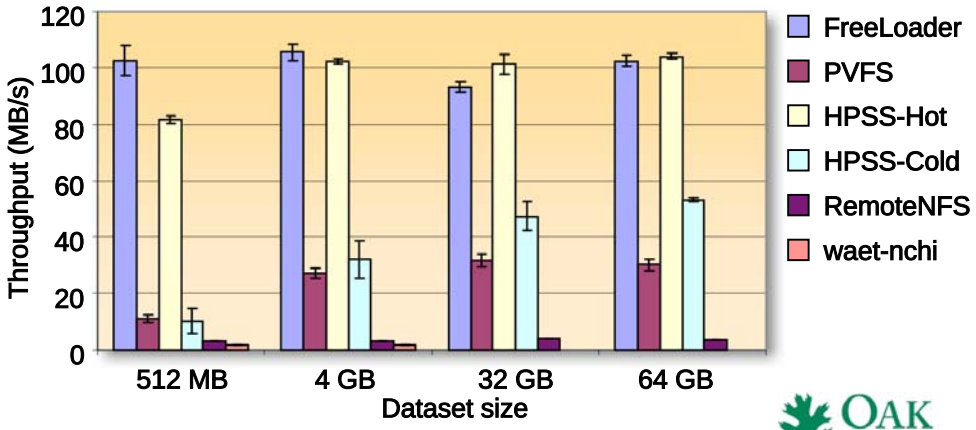
End-user data delivery: An architecture for eager offloading of result data

- Offloading result data equally important for local visualization and interpretation.
- Storage system failure and purging of scratch space can cause loss of result data; end-user resource may not be available for offload.
- Eager offloading:
  - Equivalent to data reconstruction.
  - Transparent data migration using “sink”/destination metadata as part of job submission (done as part of global coordination).
  - Data offloading can be overlapped with computation.
  - Can failover to intermediate storage/archives for planned transfers in the future:
    - Intermediate nodes specified by the user in the job script.
    - A one-to-many distribution followed by a many-to-one download.
    - A combination of bullet + landmarks (p2p + staged).
    - Erasure coded chunks for redundancy and fault tolerance.
    - Monitor offloads for bandwidth degradation and choose alternate paths accordingly.

FreeLoader: Improving end-user data delivery with client-side collaborative caching

- Enabling trends:
  - Unused storage: More than 50% desktop storage unused.
  - Immutable data: Data are usually write-once, read-many with remote source copies.
  - Connectivity: Well-connected, secure LAN settings.
- FreeLoader aggregate storage cache:
  - Scavenges O(GB) of contributions from desktops.
  - Parallel I/O environment across loosely connected workstations, aggregating I/O as well as network BW.
  - NOT a file system, but a low-cost, local storage solution enabling client-side caching and locality.

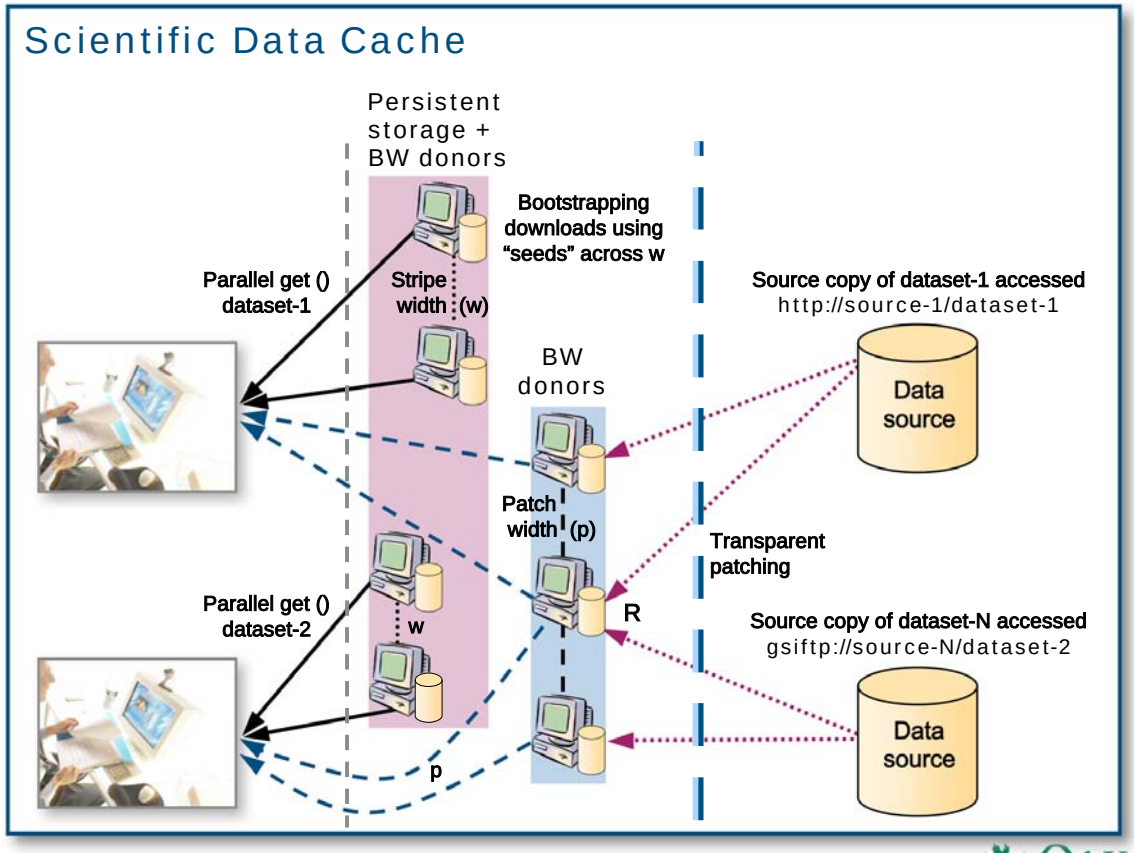




OAK RIDGE National Laboratory

Virtual cache: Impedance matching on steroids!

- Can we host partial copies of datasets and yet serve client accesses to the entire dataset?
  - ~ FileSystem-BufferCache:Disk :: FreeLoader:RemoteDataSource.
- Benefits:
  - Bootstrapping the download process.
  - Store more datasets.
  - Allows for efficient cache management.
  - Persistent storage and BW-only donors.



Contact

Sudharshan Vazhkudai

Computer Science Research Group
Computer Science and Mathematics Division
(865) 576-5547
vazhkudaiss@ornl.gov

http://www.csm.ornl.gov/~vazhkuda/Storage.html

OAK RIDGE National Laboratory