SLASH

Scalable Lightweight Archival Storage Hierarchy

Paul Nowoczynski
Pittsburgh Supercomputing Center
pauln@psc.edu

- Distributed Archival Caching System
  - More flexible than a traditional stand-alone archiver
    - Attach archiver to Lemieux via quadrics
    - Incorporate off-the-shelf hardware into mix
    - Minimize tape reads
- Completely user-mode, all I/O's into the archiver namespace are intercepted by a client-side library.
- Allows parallel I/O into separate systems while maintaining everything in a single namespace.
- Cooperative Cache – nodes may read data from one another
- No database needed

- Caveats
  - Works best with put/get requests
    - Edits or files opened without O_TRUNC are problematic because cache filesystems are independent
  - Not a parallel filesystem
    - Multiple nodes cannot write same file simultaneously
- Features
  - Portability – no kernel modifications needed.
  - Archiver system can be modularly expanded without large investment
  - Metadata is held within the tree - not in a separate database
  - Convenient for monitoring and logging of I/O operations
  - Data pre-staging with coordination of system scheduler
  - Intelligent data migration to minimize thrashing of archiver disk
  - MD5 sum “receipts” for uploaded files
  - Good at automated file migration

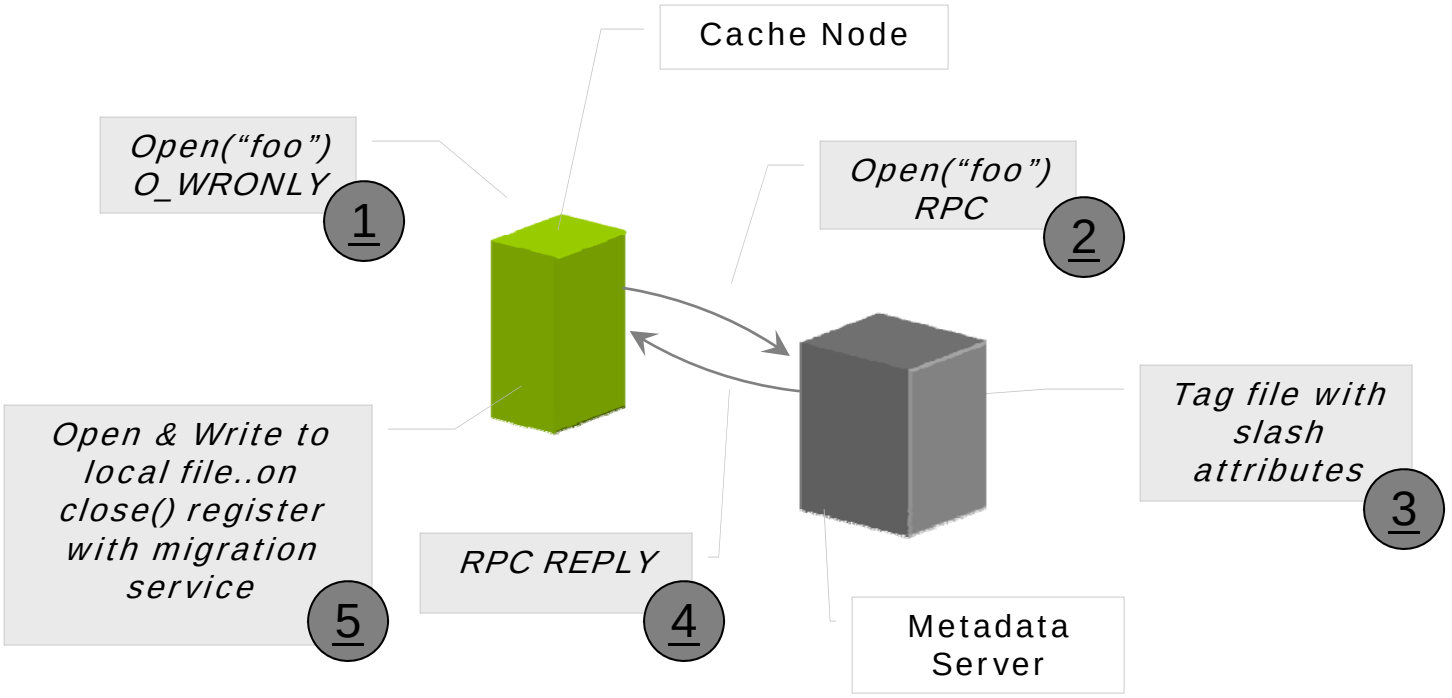
Archiver Task Offloading

- Several Archival Duties may be handled by Slash allowing for greater flexibility and increased performance of the archival system
  - File Caching
    - Data cache management handled by low cost Slash Cache Nodes
  - Data Applications
    - Run on cache nodes instead of archiver cpus.
  - Namespace Exporting
    - Namespace may be owned by an external metadata controller.
  - In the near Future..
    - Will be able to control the millions of archived small files which currently plague DMF
    - Slash will own disk MSPs for various sized files – everything will be treated as an object store (caches, DMF, slash MSPs)

How it works

- Intercepted Read I/Os (i.e. open()) contain RPCs which determine cache status and location of data.
  - When possible, data is read directly from the cooperative instead of from DMF/HPSS.
- Write I/Os are handled locally by cache node and asynchronously migrated to the archiver at a convenient time.

Slash Write

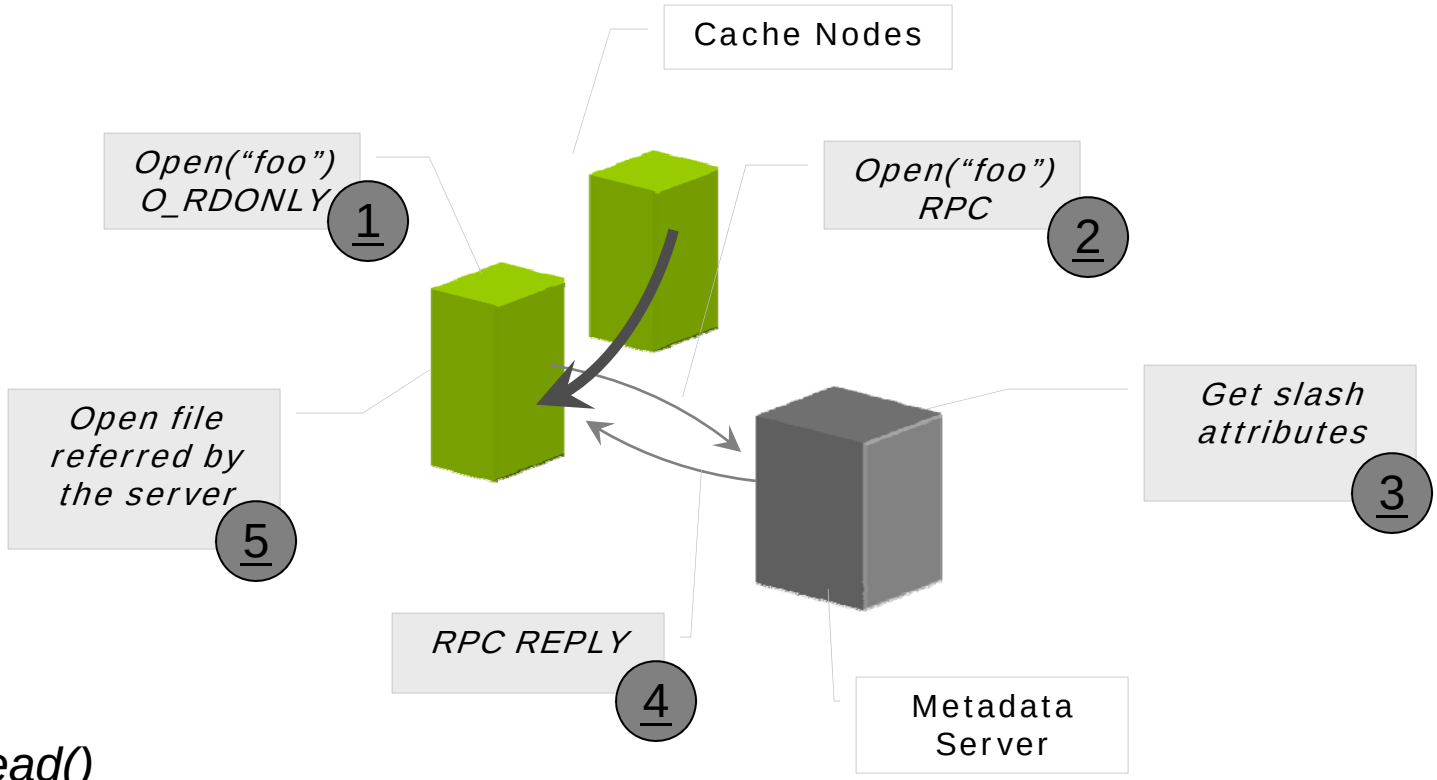


Write()

… New data is written to cache then sent to archiver at some later time (after close())

… Multi-threaded and multi-process opens of the same file are supported

Slash Read



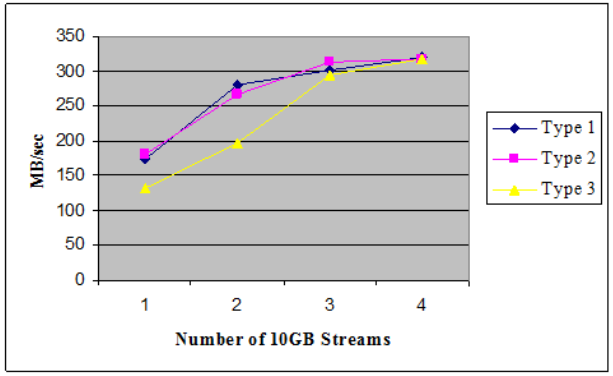
Read()

… Requested data may reside on local cache, remote cache or archiver

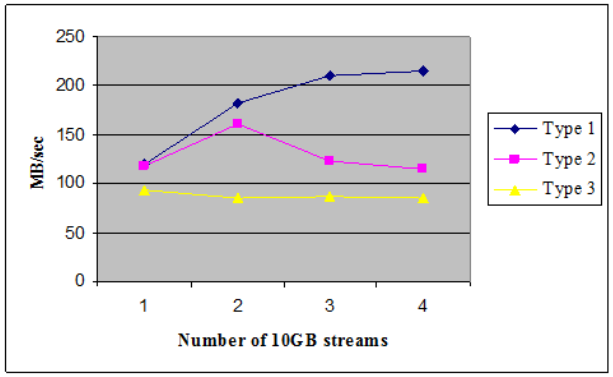
… RPC requests cache status/location info from metadata server

Slash Benchmark

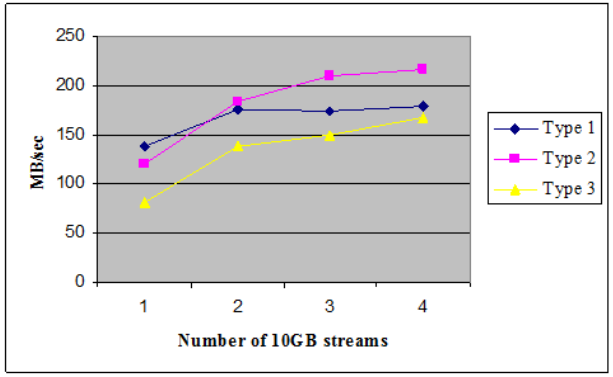
- Read and Write Throughput from Supercomputer to Slash Archiver via TCSIO/Quadrics
- Type[123] are represent different cache node hardware architectures
- This slide provides baseline measurements
- The test reads and writes 10 gigabyte data files from the supercomputer
  - No disk was involved on the supercomputer



Raw Tcsio / QSW Performance (no disk I/O)



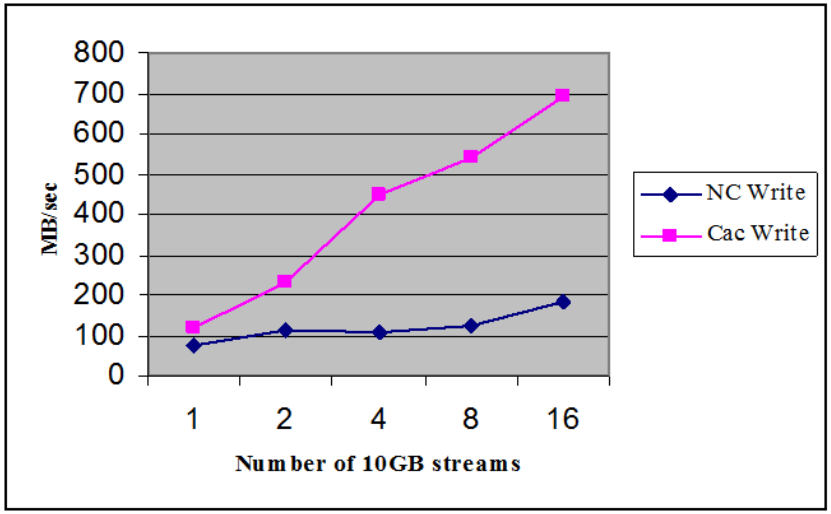
Tcsio/Slash Write Bandwidth (all I/O is written to local cache node disk)



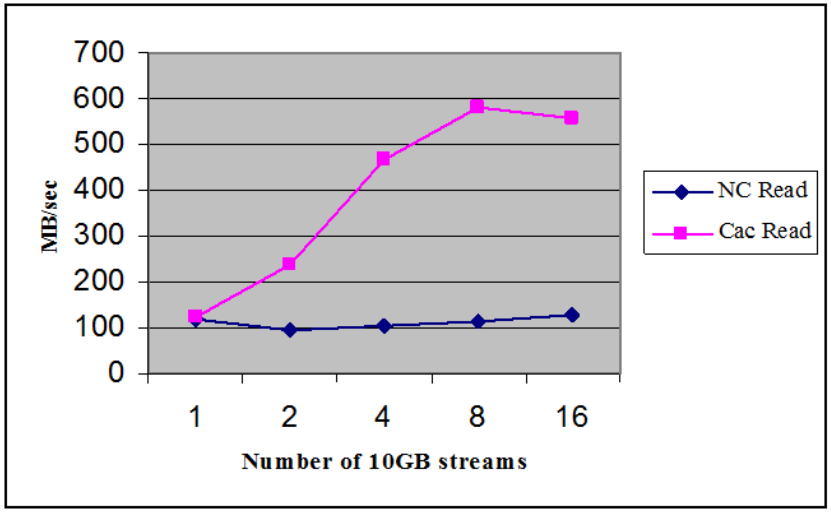
Tcsio/Slash Read Bandwidth

Cached vs. Non-cached I/O

- Upload and download between archiver namespace and supercomputer
  - Up to six cache nodes were used
    - 4 SLCNS for 4 streams
    - 6 SLCNS for 8 and 16 streams
- Blue line represents I/O directly into the archiver's disk
  - Scalability of blue is poor due to concatenation of disk volume group.
- Uploads favored faster Cache nodes
  - Type 1 Cache nodes handled more streams
  - Explains performance drop for 16 readers since the uploaded data was used for the read test.
- Data shows that system scales as nodes are added



Cached vs. Non-Cached Writes



Cached vs. Non-Cached Reads