SLASH2

A Filesystem for Widely Distributed Systems

Pittsburgh Supercomputing Center
Advanced Systems Group

Paul Nowoczynski
Jared Yanovich
Zhihui Zhang

http://quipu.psc.teragrid.org/slash2

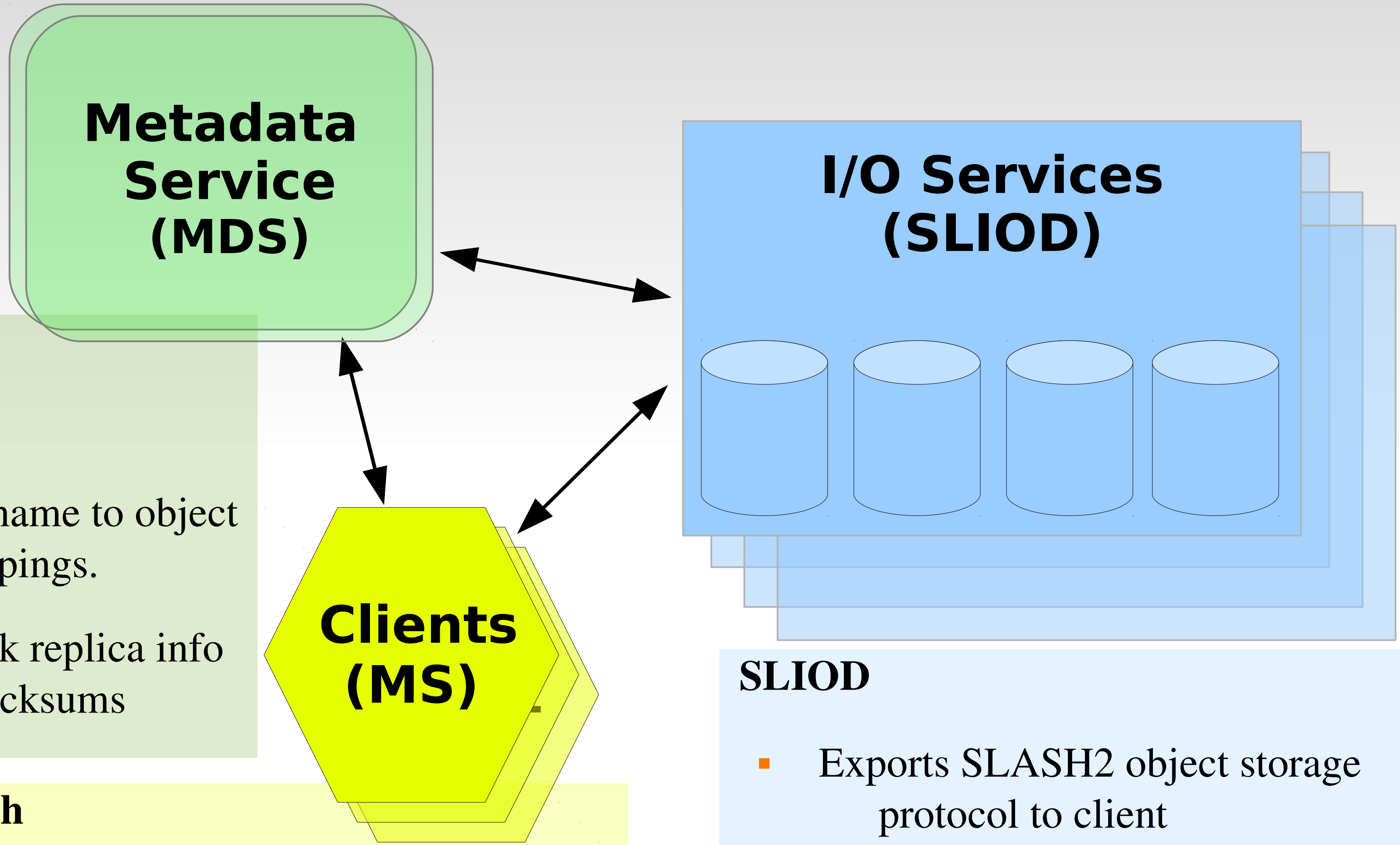
SLASH2 High-Level Architecture

- Distributed / Parallel Filesystem
  - ..with focus on wide-area
- Similar to Lustre in terms of core architecture
  - Object-based
  - Portals RPC / LNet
- Runs in user space
  - I/O services are highly portable
  - Clients utilize FUSE

Motivation behind this work

- Create filesystem which can logically bind independent storage systems
- Provide a common wide-area filesystem
  - Span institutions
  - Integrate with a variety of existing storage systems
- Alternatives solutions: Lustre-WAN, iRODS, GPFS, Panache
- Issues with existing solutions
  - too intrusive, vendor specific, or don't allow for POSIX I/O, don't support mult-resident data

SLASH2 Components



MDS

- ZFS-based
- Maintains name to object ID mappings.
- Stores block replica info and checksums

Mount_Slash

- FUSE based, multi-threaded
- Advanced features like write-back, read-ahead, and I/O coalescing

SLIOD

- Exports SLASH2 object storage protocol to client
- Checksum calculation and verification
- Highly Portable

Features of Interest

- Multi-resident data
  - System managed replication
  - Supports parallelism - even for a single file
- Block-level checksumming
  - Stored and maintained on the metadata server
- Location aware data retrieval
  - Client may chose source based on proximity or other metric

Features for WAN

Filesystem protocols designed to decrease or minimize communications between clients and servers

- Create-on-write backing objects
- Readdir+
  - Minimizes attribute fetch requests for operations like: 'ls -al', 'find'..
- File size stored on metadata server
- Asynchronous garbage collection
  - Invalidated file replicas
  - Unlinked file objects
  - Fully truncated file objects

Portable I/O Service

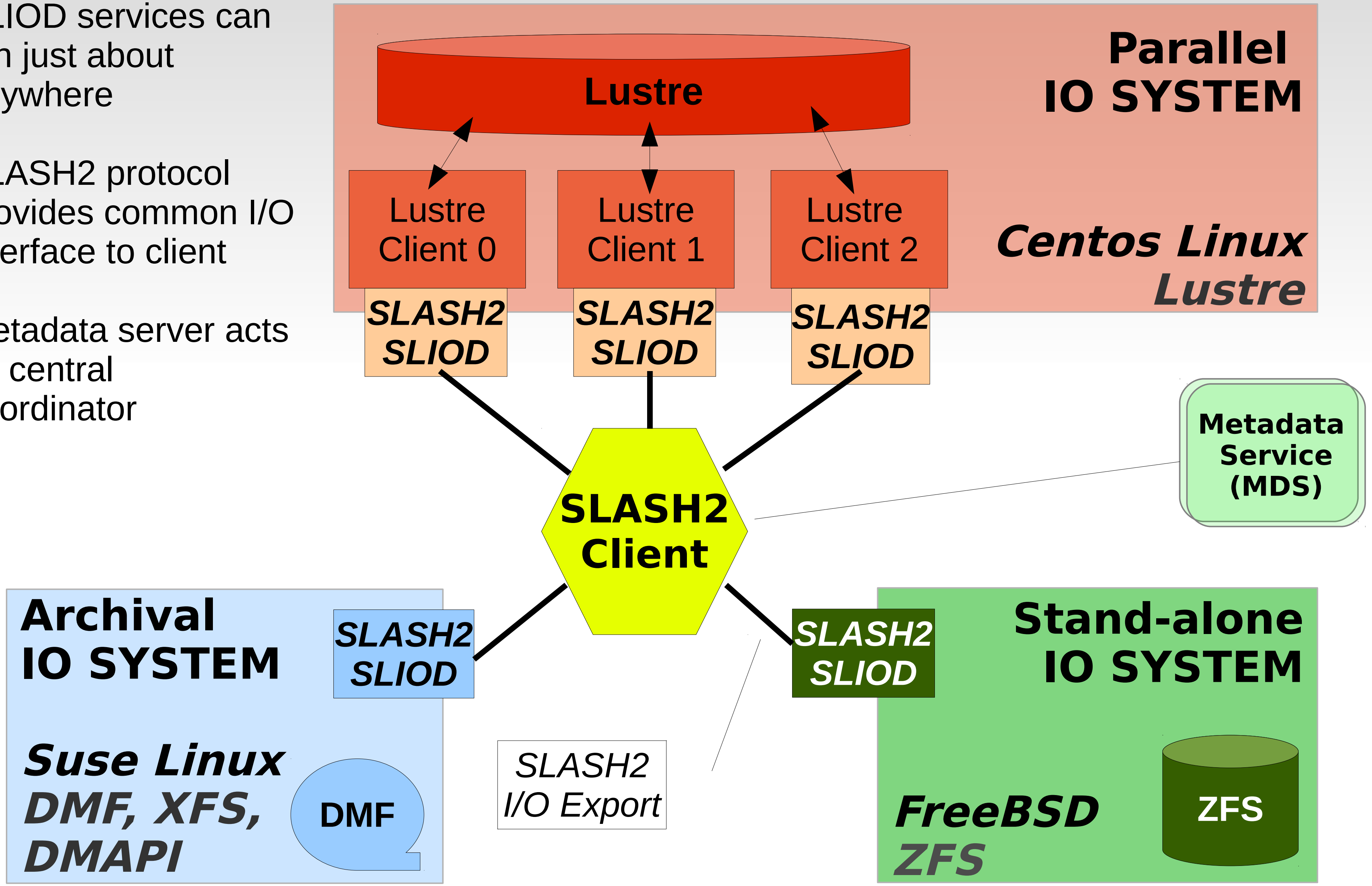
- Designed for use on a wide variety of systems
- Utilizes existing filesystems as storage backends for SLASH2
- Flexible system requirements
  - Presentation of a POSIX filesystem
  - TCP sockets
- No kernel modules required
  - Behaves like an application to the underlying FS
    - Open, close, create, unlink, p[write|read], etc.

Possible I/O System Configuration

SLIOD services can run just about anywhere

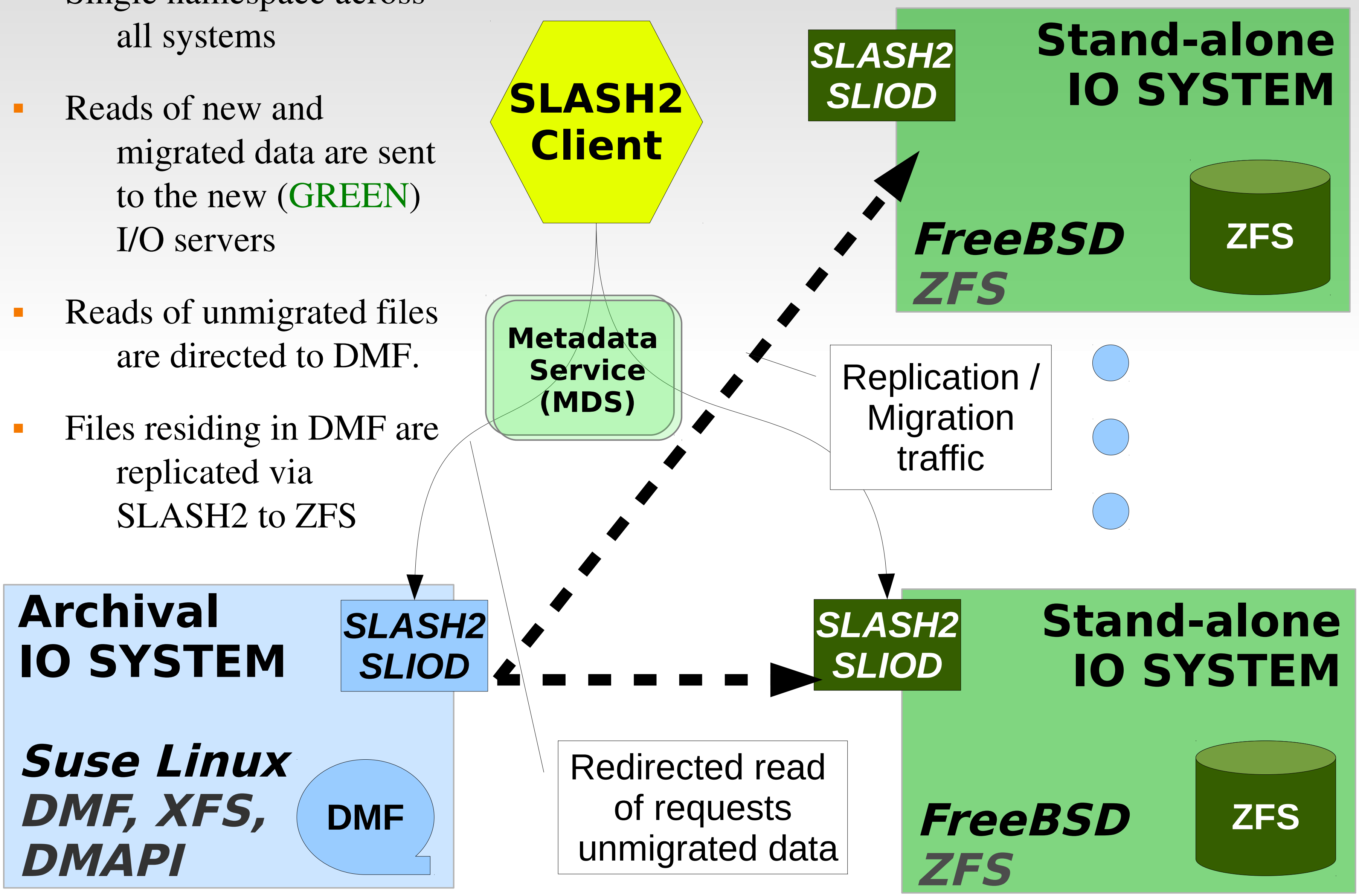
SLASH2 protocol provides common I/O interface to client

Metadata server acts as central coordinator



PSC Archival Configuration

- Single namespace across all systems
- Reads of new and migrated data are sent to the new (GREEN) I/O servers
- Reads of unmigrated files are directed to DMF.
- Files residing in DMF are replicated via SLASH2 to ZFS



PSC Archival Store

- Upcoming system utilizing SLASH2
- Deployment will feature core SLASH2 capabilities
- System-managed data transfer
  - Handle migration from old system to new
- Encapsulation of otherwise disparate systems
  - Logically binds a DMF / Tape-based system with a cluster of FreeBSD / ZFS servers.

SLASH2 Metadata Structures

- Unix file attributes are stored within ZFS object for performance
- SLASH2 specific data is stored as ZFS contents on the MDS
- Per file support for replication schemes and policies

SLASH2 Metadata Structures

```c
/*
 * The inode structure lives at the beginning of the metafile.
 * @ino_version: compatibility.
 * @ino_flags: slash2 specific file attributes.
 * @ino_bsz: size of this objects bmap.
 * @ino_nrepls: number of replicas, if > SL_DEF_REPLICAS use inode_extras.
 * @ino_replpol: file replication policy.
 * @ino_repls: replica storage.
 * @ino_repl_nblks: support st_blocks in multi-res filesystem.
 */
struct slash_inode_od {
	uint16_t		ino_version;
	uint16_t		ino_flags;
	uint32_t		ino_bsz;
	uint32_t		ino_nrepls;
	uint32_t		ino_replpol;
	sl_replica_t		ino_repls[SL_DEF_REPLICAS];
	uint64_t		ino_repl_nblks[SL_DEF_REPLICAS];
};

struct slash_inode_extras_od {
	sl_replica_t		inox_repls[SL_INOX_NREPLICAS];
	uint64_t		inox_repl_nblks[SL_INOX_NREPLICAS];
};
```

Metadata & Multi-Residency

- SLASH2 maintains per chunk metadata
- 'Chunks' (32MiB-256MiB) are bigger than 'blocks' (4KiB-4MiB)
- The structure which describes chunks is called a ***bmap***
- Residency and policy and managed per ***bmap***

```
/*
 * @bmod_crcstates: bits describing the state of each sliver.
 * @bmod_repls: bitmap used for tracking the replication status of this bmap.
 * @bmod_crcs: the CRC table, one 8-byte CRC per sliver.
 * @bmod_gen: current generation number.
 * @bmod_replpol: replication policy.
 */
struct bmap_ondisk {
	uint8_t		bmod_crcstates[SLASH_CRCS_PER_BMAP];
	uint8_t		bmod_repls[SL_REPLICA_NBYTES];
	uint64_t	bmod_crcs[SLASH_CRCS_PER_BMAP];
	uint32_t	bmod_gen;
	uint32_t	bmod_replpol;
}
```

Replication Example

- **Create a file**

```
(pauln@peel0:~)$ dd if=/dev/zero of=/p0_archive/pauln/big_file count=2k bs=1M
2048+0 records in
2048+0 records out
2147483648 bytes (2.1 GB) copied, 4.81828 seconds, 446 MB/s
```

- **Note current residency status**
  - **File lives on resource "archsliod@PSCARCH"**

```
(pauln@peel0:msctl)$ ./msctl -r /p0_archive/pauln/big_file
file-replication-status                                    #valid  #bmap  %prog
================================================================================
/p0_archive/pauln/big_file
new-bmap-repl-policy: one-time
  archsliod@PSCARCH                                            16     16   100%
    ++++++++++++++++
```

16 bmaps resident at I/O resource ***archsliod@PSCARCH***

- **Issue replication request to resource "archlime@PSCARCH"**

```
(pauln@peel0:msctl)$ date && ./msctl  -Q archlime@PSCARCH:*:/p0_archive/pauln/big_file
Wed Jul 20 02:51:56 EDT 2011
```

Replication Example (2)

- **Check status**

1 second elapsed
1 bmap en route,
15 enqueued

```
Wed Jul 20 02:51:57 EDT 2011
file-replication-status                                  #valid  #bmap   %prog
==============================================================================
/p0_archive/pauln/big_file
new-bmap-repl-policy: one-time
  archsliod@PSCARCH                                          16     16    100%
    ++++++++++++++++
  archlime@PSCARCH                                            0     16      0%
    sqqqqqqqqqqqqqqq
```

8 seconds elapsed
10 bmaps done
6 enqueued

```
Wed Jul 20 02:52:05 EDT 2011
file-replication-status                                  #valid  #bmap   %prog
==============================================================================
/p0_archive/pauln/big_file
new-bmap-repl-policy: one-time
  archsliod@PSCARCH                                          16     16    100%
    ++++++++++++++++
  archlime@PSCARCH                                           10     16  62.50%
    ++++++++++qqqqqq
```

24 seconds elapsed
all bmaps done !

```
Wed Jul 20 02:52:20 EDT 2011
file-replication-status                                  #valid  #bmap   %prog
==============================================================================
/p0_archive/pauln/big_file
new-bmap-repl-policy: one-time
  archsliod@PSCARCH                                          16     16    100%
    ++++++++++++++++
  archlime@PSCARCH                                           16     16    100%
    ++++++++++++++++
```

More on Replication..

What just happened? System Managed Data Xfer

1. The 'msctl' command created a replication request

```
(pauln@peel0:msctl)$ date && ./msctl  -Q archlime@PSCARCH:*:/p0_archive/pauln/big_file
Wed Jul 20 02:51:56 EDT 2011
```

2. This caused the MDS to statefully create a work request for 'big_file'. Once complete.. it returned 'OK' to the client

3. MDS scans the metadata for bmaps which do not meet the new replication criteria, making a work item for each.

4. Work items are scheduled to be sent from

   'archsliod@PSCARCH' → 'archlime@PSCARCH'

   Upon receiving the work request, the SLIOD retrieves the checksum table from the MDS to verify incoming contents.

5. MDS provides some flow control and rudimentary scheduling – as work is completed, more is allocated until done

Replication – What wasn't shown..

- When using I/O systems which have multiple SLIODs, the MDS can distribute replication work across all destination SLIODs.
- Robustness
  - MDS resumes all unfinished replication work on restart
  - MDS can cope with missing or slow SLIOD endpoints
- Load balancing
  - SLIODs are given work piecemeal.
  - Parallel replications are not disproportionately affected by a single slow or oversubscribed node.

Metadata Replication

- Provide near-uniform metadata performance throughout the wide-area
  - Lots of metadata operations rely on small, serial RPCs
  - multi-millisecond lookup()'s are performance killers!
- Eventual consistency is the only way..
  - Maintaining read and write locks on directories and metadata structure will not work
  - Wide-area latency will crush performance
  - Network partitioning and spanning of administrative domains present challenges

Metadata Replication (2)

- Current plan will rely on ZFS snapshotting
- Lots of advantages
  - zfs [send|recv] does all of the heavy lifting
  - Easily administered (via standard zfs tools)
  - New metadata servers may be easily incorporated

Metadata Replication (3)

How it works..

- Each MDS has a ZFS filesystem for himself and each of his peers
- Local MDS may only modify his designated ZFS filesystem
  - Publishes modifications to his peers
  - Receives updates from his peers on behalf
  - Asymmetric performance for some updates ..BUT..
  - Can make progress on local resources in the event of network partitions or remote server failures

Metadata Replication (4)

- Eventually consistent namespace - system aims to for a 60 second update interval
  - i.e. if the system quiesced for >60 seconds, all MDSs should reach synchronization
  - This is goal.. may not be realistic
- More details will be made available on the web..

Questions?

http://quipu.psc.teragrid.org/slash2