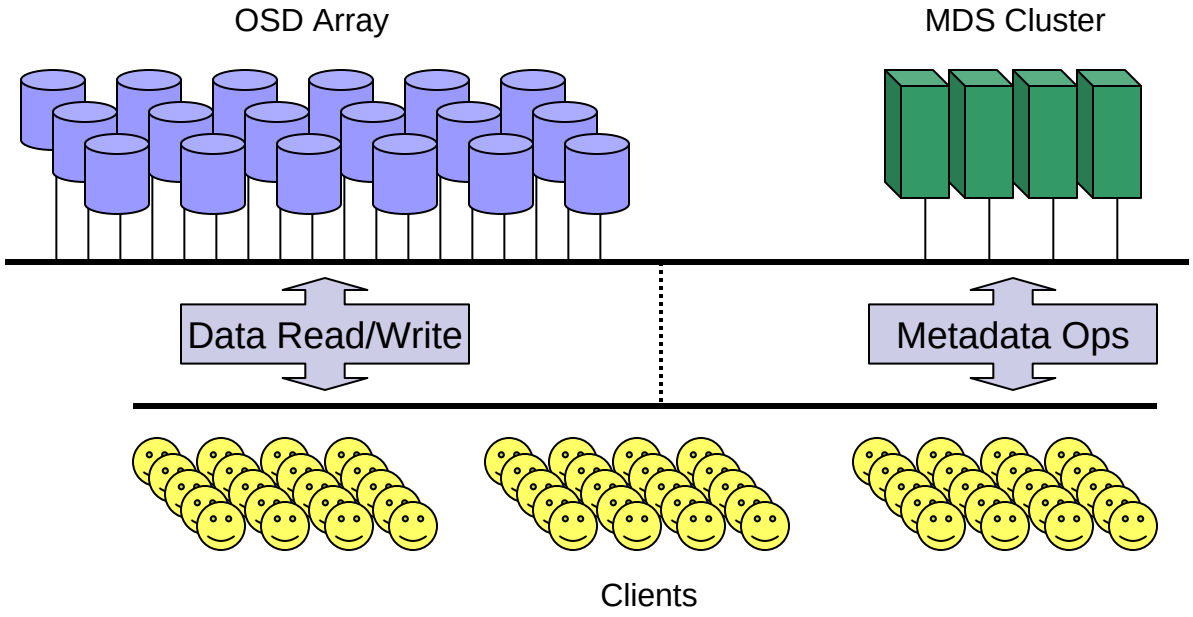
Intelligent Metadata Management for a Petabyte-scale File System

Sage Weil
Scott Brandt, Ethan Miller, Kristal Pollack
UC Santa Cruz

May 19, 2004

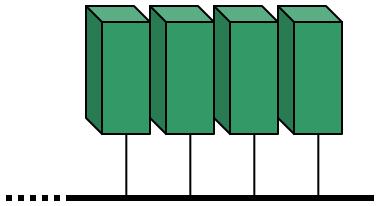
System Overview

- Petabytes of storage (10^{18} bytes, millions of gigabytes)
- Billions of files ranging from bytes to terabytes
- 10,000 to 100,000 clients
- File I/O decoupled from metadata, namespace operations
  - ~1,000 Object-based Storage Devices (OSDs)
  - ~10 Metadata Servers (MDSs)



Metadata Cluster

- The MDS cluster is responsible for
  - Managing the file system namespace (directory structure)
  - Facilitating client access to file data
- Hierarchical directory structure (POSIX semantics)
- Traditionally two types of metadata
  - Inodes
    - Size, mtime, mode
    - Where is the data?
  - Directory entries
    - File name
    - Inode pointer



- Goal #1: Mask I/O to underlying disk storage by leveraging MDS cluster's collective cache

Why is Metadata Hard?

- File data storage is trivially parallelizable
  - File I/O occurs independent of other files/objects
  - Scalability of OSD array limited only by network architecture
- Metadata semantics are more complex
  - Hierarchical directory structure defines object interdependency
    - Metadata location & POSIX permissions depend on parent directories
  - Consistency and state
    - MDS must manage open files, client capabilities for data access, locking
- Heavy workload
  - Metadata is small, but there are lots of objects and lots of transactions
  - Good metadata performance is critical to overall system performance

Metadata Workload

- As many as half of all file system operations access metadata
  - Open, close
  - Readdir, many stats (ls –F or -al)
- Large systems must address a variety of usage patterns
  - General purpose computing
  - Scientific, cluster computing
- Must efficiently handle “hotspot” development
  - Directories near root of hierarchy are necessarily popular
  - Popular files: many opens of same file (e.g. /lib/*)
  - Popular directories: many creates in same directory, (e.g. /tmp)

➢ Goal #2: MDS cluster should efficiently handle varied and changing file systems and usage patterns

MDS Cluster Design Issues

- Consistency
  - Updates must be consistent within MDS cluster; clients should have consistent view of file system
- Leveraging Locality of Reference
  - Hierarchies are our friend
- Workload Partitioning
  - Balance MDS loads, maximize efficiency and throughput
- Traffic Management
  - Hot spots, flash mobs
- Metadata Storage
  - We want fast commits and efficient reads

Simplifying Consistency

- When metadata is replicated, a simple locking and coherence strategy is best

- Single MDS node acts as “authority” for any particular metadata object (“primary copy replication”)
  - Serializes updates
  - Commits updates to stable storage (e.g. disk, NVRAM)
  - Manages cooperative cache

- MDS cluster is collectively a large, consistent and coherent cache

Client Consistency

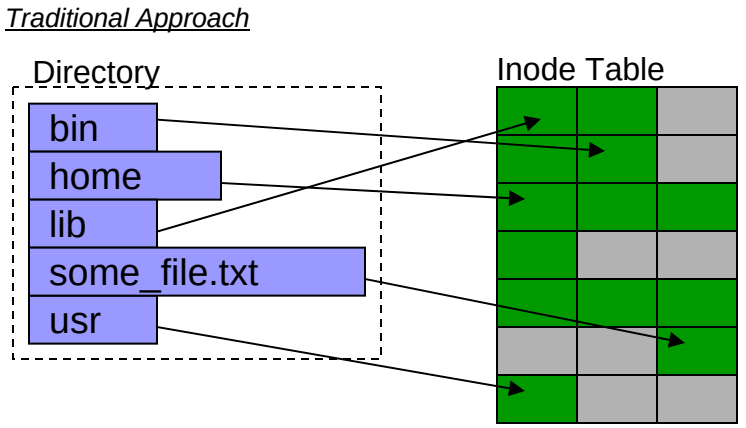
- Two basic strategies
  - Stateless (NFS): cached metadata times out at client after some period
    - Easy, but weak (no client cache coherence)
    - More traffic as clients refresh previously cached metadata
  - Stateful (Sprite, Coda): MDS cluster tracks what clients cache what, and invalidates as necessary
    - Strong consistency, coherence
    - Higher memory demands on MDS cluster
- Two kinds of state
  - Open file coherence (shared read/write)
    - Allows *data* consistency
    - Client callbacks can inform clients of concurrent modifications and potentially stale data
    - Byte-range locking, etc.
  - Complete metadata coherence ('ls –F' results)
    - Managing cached client metadata allows consistent, coherent view of namespace, inode metadata
    - Significant memory demands for tracking client cache contents
      - Can be reduced by tracking at granularity of directories
- Choice of strategy may be site specific

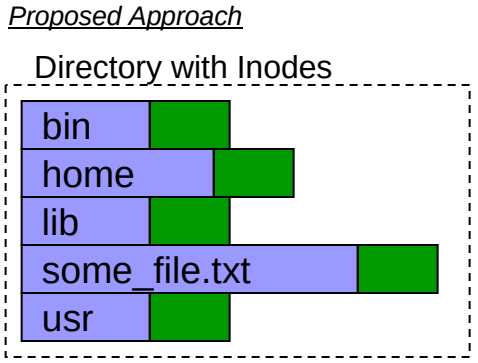
Leveraging Locality of Reference

- A hierarchical storage paradigm leads to high locality of reference within directories and subtrees

- Store inodes with directory entries
  - Streamlines typical usage patterns (lookup + open or readdir + stats)
  - Allows prefetching of directory contents

- Inodes no longer globally addressable (by inode number)
  - Avoids management issues associated with a single large, distributed, and sparsely populated table
  - Hard links are easy to handle with small auxiliary inode table for the rare doubly-linked inode

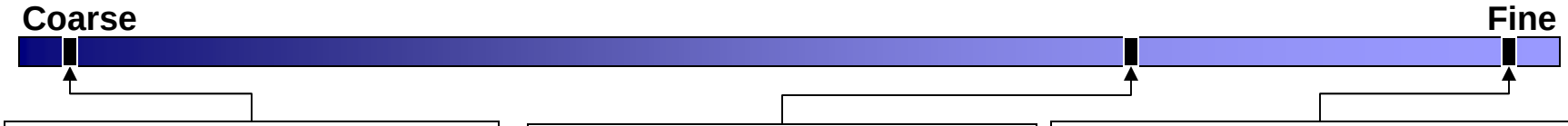




Partitioning Approaches

How to delegate authority?

- Manual subtree partitioning (coarse distribution)
  - preserves locality
  - leads to imbalanced distribution as file system, workload change
- Hash-based partitioning (finer distribution)
  - provides a good probabilistic distribution of metadata for all workloads
  - ignores hierarchical structure
    - less vulnerable to “hot spots”
    - destroys locality

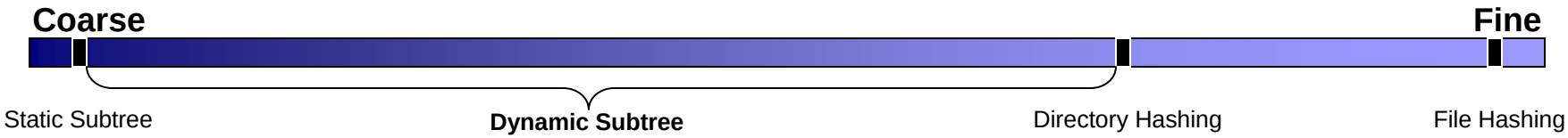


Static Subtree Partitioning
- Portions of file hierarchy are assigned to MDS nodes (ala NFS, AFS, etc.)

Directory Hashing
- Hash on directory portion of path only
- Maintains locality within a directory but not subtree
- Large or popular directories skew distribution

File Hashing
- Metadata distributed based on hash of inode or path
- Good distribution, but eliminates locality of reference by scattering directory contents

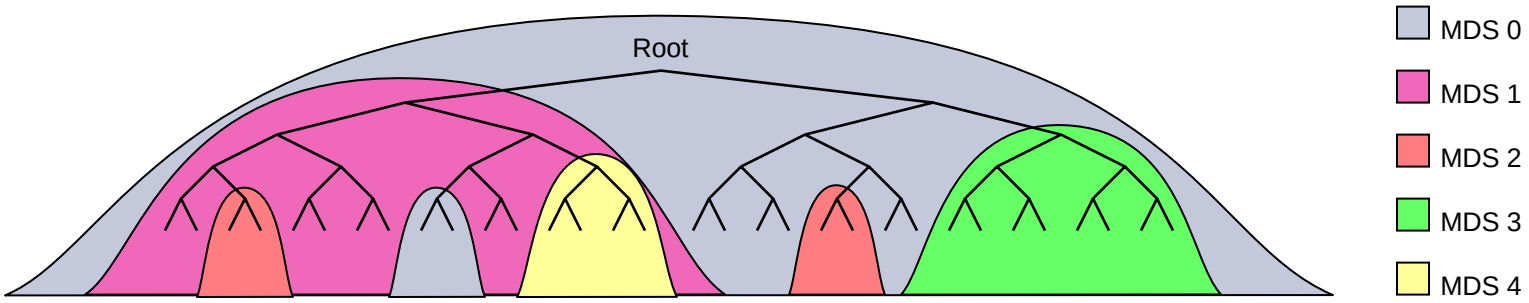
Dynamic Subtree Partitioning



- Distribute subtrees of directory hierarchy
  - Somewhat coarse distribution of variably-sized subtrees
  - Preserve locality within entire branches of the directory hierarchy
- Must intelligently manage distribution based on workload demands
  - Keep MDS cluster load balanced
  - Requires active repartitioning as workload and file system change instead of relying on a (fixed) probabilistic distribution

- ✓ Consistency
- ✓ Locality
- ➢ **Partitioning**
- • Traffic Management
- • Storage

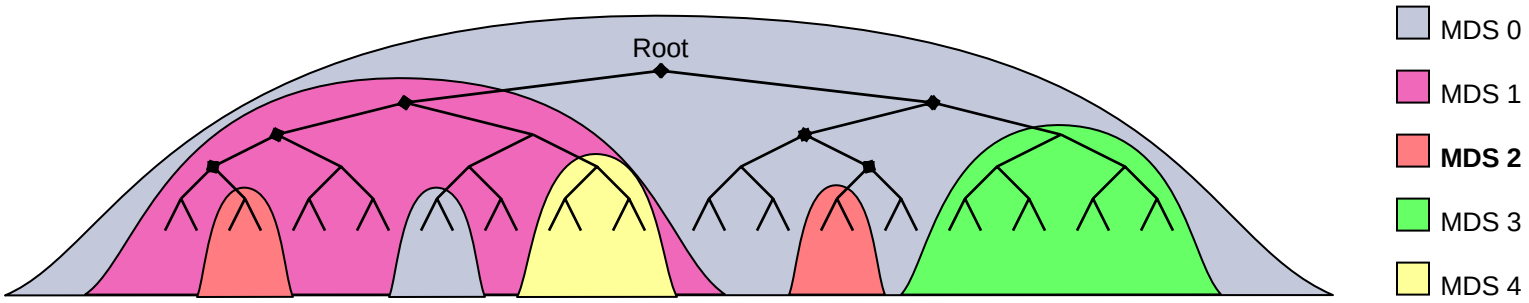
A Sample Partition



- The system dynamically and intelligently redelegates arbitrary subtrees based on usage patterns
- Coarser, subtree-based partition means higher efficiency
  - Fewer prefixes need to be replicated for path traversal
- Granularity of distribution can range from large subtrees to individual directories

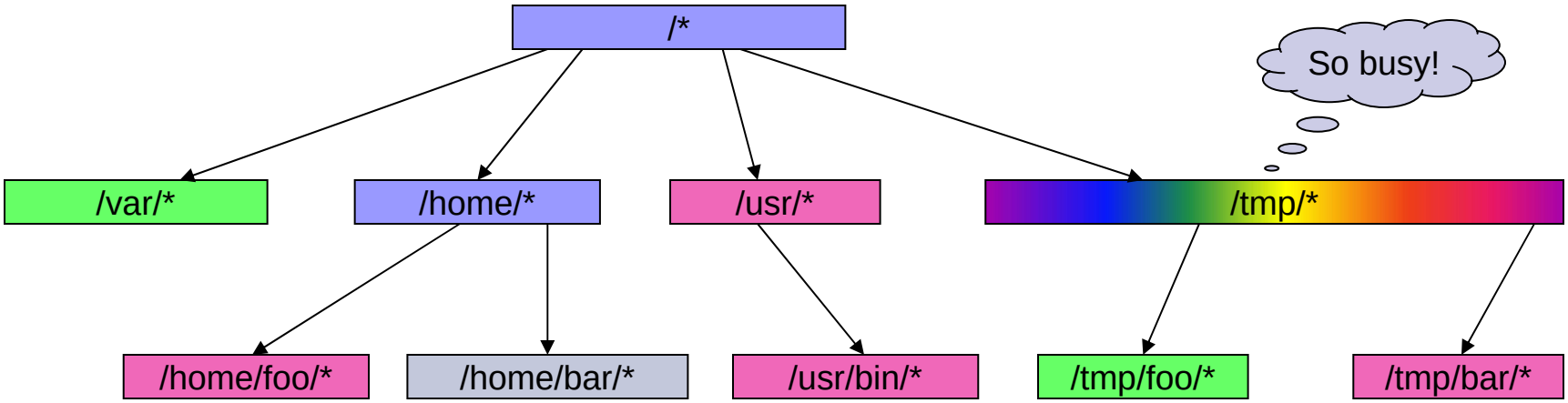
- ✓ Consistency
- ✓ Locality
- ➢ **Partitioning**
- • Traffic Management
- • Storage

A Sample Partition



- The system dynamically and intelligently redelegates arbitrary subtrees based on usage patterns
- Coarser, subtree-based partition means higher efficiency
  - Fewer prefixes need to be replicated for path traversal
- Granularity of distribution can range from large subtrees to individual directories

Distributing Directory Contents



- If a directory is large or busy, it's contents can be selectively hashed across the cluster
  - Directory entry/inode distribution is based on hash of parent directory id and file name

➢ Whether a directory is hashed is *dynamically* determined

Flash Mobs and Hot Spots

- There is a tradeoff between fast access and load balancing
  - If clients know where to locate any metadata, they can access it directly but at any time they may decide to simultaneously access any single item without warning
  - If clients always contact a random MDS or proxy, load can always be balanced but all (or most) transactions require extra network hops
- Flash mobs are common in scientific computing and even general purpose workloads
  - Many opens of the same file
  - Many creates in one directory
- A MDS with a popular item will not operate as efficiently under load, or may crash
- Fine-grained distribution can help, but is an incomplete solution

- ✓ Consistency
- ✓ Locality
- ✓ Partitioning
- ➢ **Traffic Management**
- • Storage

Traffic Control

- ❖ Ideally
  - □ Unpopular items: clients directly contact the authoritative MDS
  - □ Popular items: replicated across the cluster
- ➢ Dynamic distribution can exploit client ignorance!
- ■ MDS tracks popularity of directories and files
  - □ Normally, the MDS tells clients where items it requests live (who the authority is)
  - □ If an item becomes popular, future clients are told the data is replicated
  - ➢ Number of clients thinking that any particular item is in any single place is bounded at all times
- ■ Clients choose MDS to contact based on their best guess
  - □ Direct queries based on deepest known prefix
  - □ Coarse subtree partitioning makes this a good guess

/been/here/before/but/not/here/yet

Traffic Control In Action

- Suppose clients 1 through 1000 are booted in sequence and immediately begin using portions of the file system. After some time, their caches might look like this:

Client1 Cache

Item	Location
/	0
/bin	1
/home	3
/home/foo	2
/home/foo/a	2

Client100 Cache

Item	Location
/	0,1,2,3,4
/bin	0,1,2,3
/home	0,3
/home/bar	1
/home/bar/b	1

Client1000 Cache

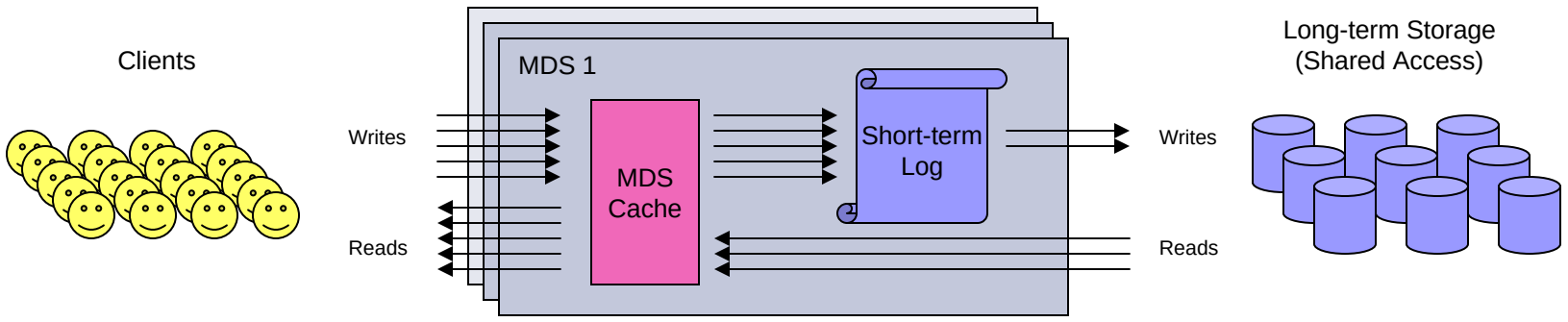
Item	Location
/	0,1,2,3,4
/bin	0,1,2,3,4
/home	0,1,2,3,4
/home/baz	3
/home/baz/c	3

- Access to /*, /bin/*, /home/* is distributed across the cluster
- Access to /home/$user/* goes directly to authoritative MDS

- Suppose all clients suddenly try to open /bin/ls:
  - Client1 contacts MDS 1, everybody else contacts a random MDS
- All clients open /home/bar/foo.txt:
  - Client100 knows where /home/bar is, but
  - Other clients are only familiar with /home, and thus direct their query at an (effectively) random MDS

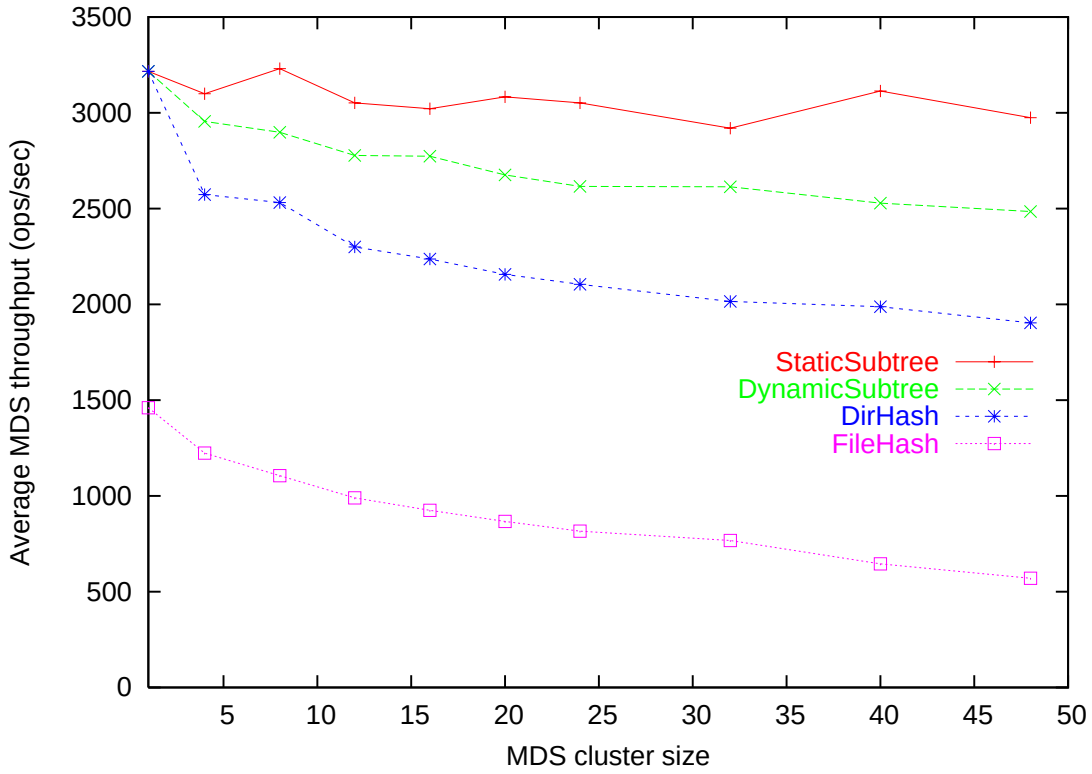
Tiered Metadata Storage

- Short-term storage
  - Immediate commits require high write bandwidth
  - Log structure for each MDS, with size similar to the MDS's cache
  - Items expired from cache or falling off end of log are written to long-term storage
  - Absorbs short-lived metadata
- Long-term storage
  - Reads should dominate
  - Directory contents grouped together (directory entries and inodes)
  - Likely a collection of objects, or log structure
- OSDs as underlying storage mechanism
  - Random, shared access facilitates MDS failover, workload redistribution
  - Objects are suitable for storing logs, directory contents, but OSDs may need to be tuned for a small object workload



Preliminary Results

- Simulated to evaluate partitioning and load balancing strategies
- Compares relative ability of system to scale
  - Identical MDS nodes (memory), scaled # disks, clients, and file system size.



Simulation Parameters

Each MDS has fixed memory (20,000 records)

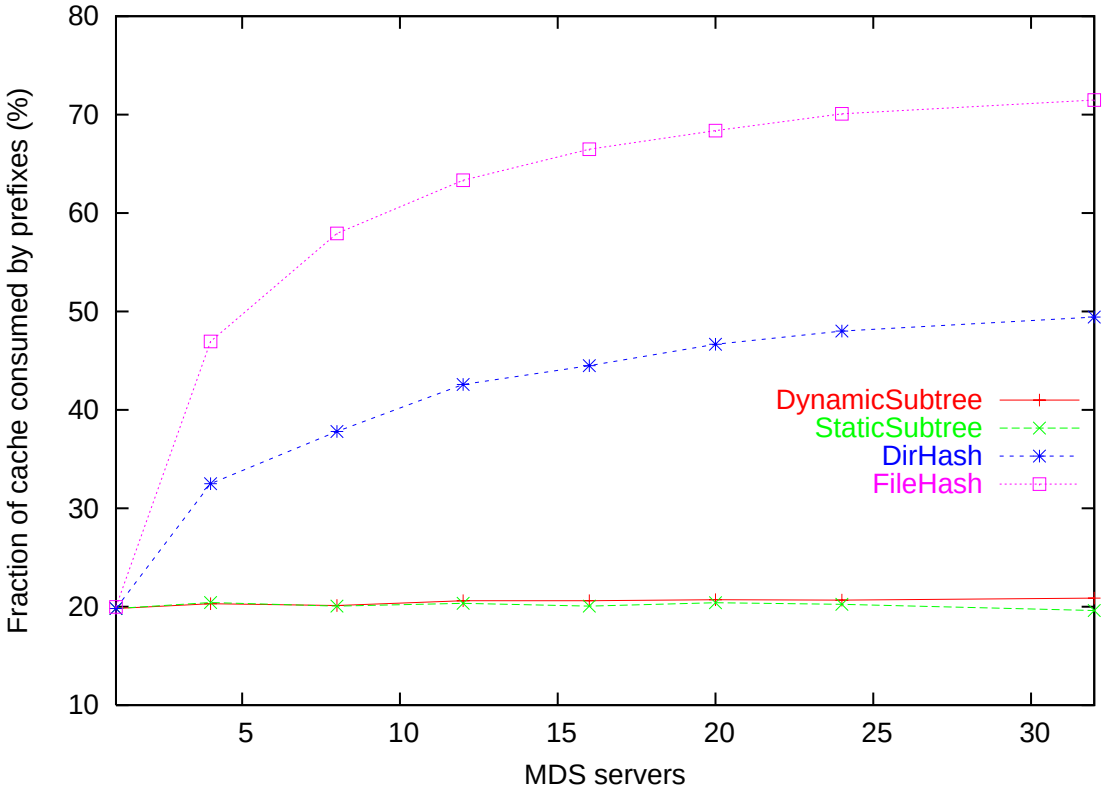
File system size and workload scaled with cluster size:
- 1000 clients/MDS
- 80,000 files/MDS
- 1 disk/MDS

Collective cache always 20% of file system metadata

Workload:
- File system: collection of user home directories
- Clients: issue semi-localized requests

Subtree Partitioning is Efficient

- Subtree partitioning incurs a lower overhead from replicated prefix metadata (implicitly involved in all metadata operations)
- More memory available for caching other metadata

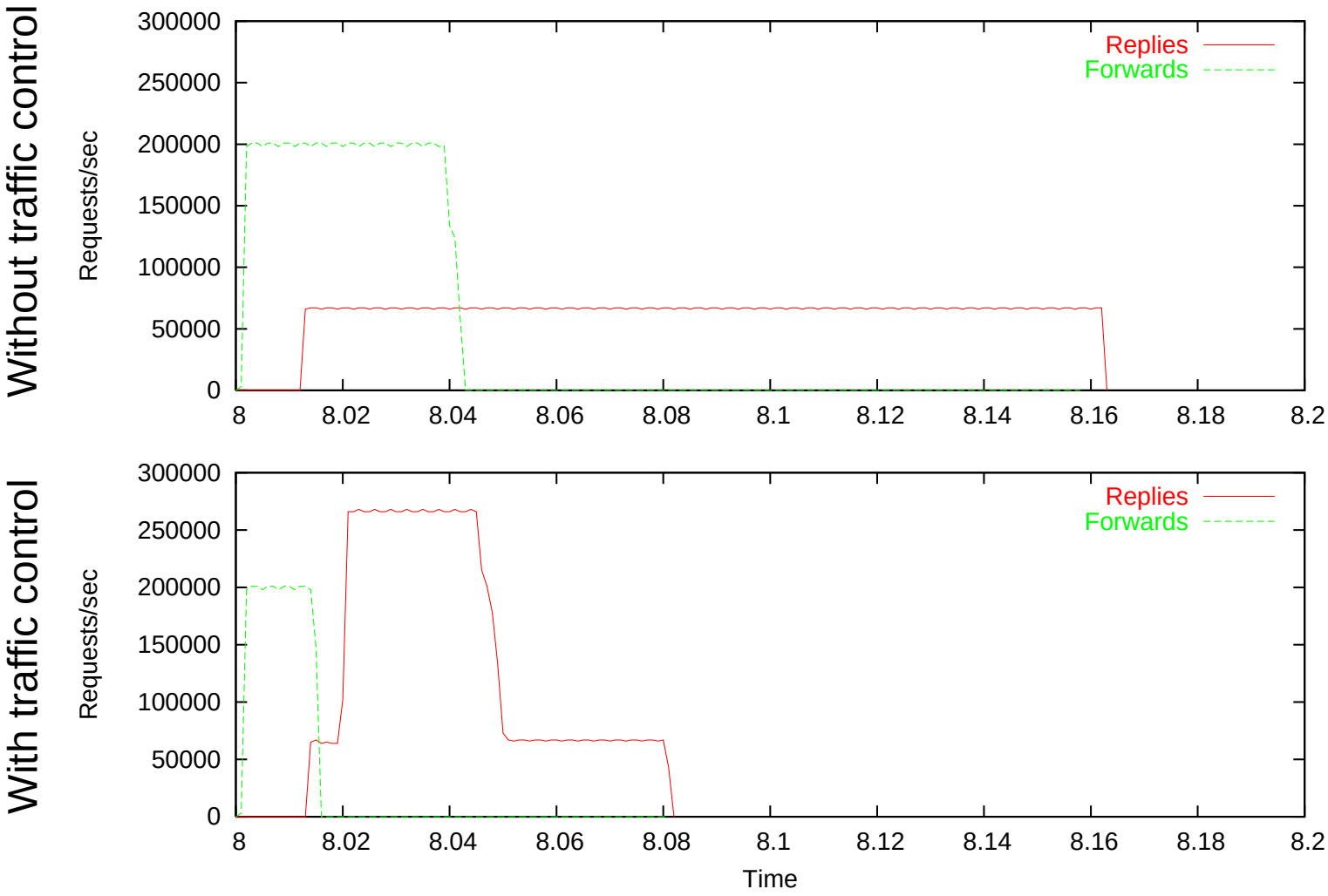


Practical Load Balancing Issues

- Workload partitioning management is difficult
- Defining “load” to balance is critical
  - Different resource constraints: CPU, network, memory
  - Maximize efficiency? throughput? fairness?
  - Making distribution “fair” based on cache utilization or efficiency often results in (equally) poor cache performance on all MDS nodes and lower overall throughput

- Dynamic partitioning allows load to be distributed based on any metric or policy
  - Fair
  - Prioritize portions of the directory hierarchy (e.g. live directories over archival data)
  - Prioritize clients (e.g. visualization vs. background traffic)?

Traffic Control Works



Conclusions

- Subtree-based distributions maximize MDS efficiency in a static environment
- Dynamic partitioning allows the MDS cluster to adapt to changing workloads and file systems
- Hot spots can be preempted by managing client ignorance
- Dynamic partitioning allows workload distribution policies beyond load balancing

Future Work

- Real implementation
  - Simulator does not scale. We need a real cluster to evaluate performance at scale
- More realistic workloads
  - Difficult to simulate a workload (general purpose or next generation scientific) with realistic levels of locality, overlap, etc.
  - Need file system snapshots for traces to be useful
- Long-term storage strategy
  - What will the long-term workload look like? How should we lay out data on disk or in a database?
  - Archival metadata: snapshots?
- Dynamic partitioning and throughput strategies
  - Distributed algorithms for efficiently (re)distributing metadata
  - Strategies for maximizing total throughput, average/median response time, etc.

Questions?

sage@cs.ucsc.edu

Acknowledgements
Lawrence Livermore National Laboratory
Sandia National Laboratory
Los Alamos National Laboratory