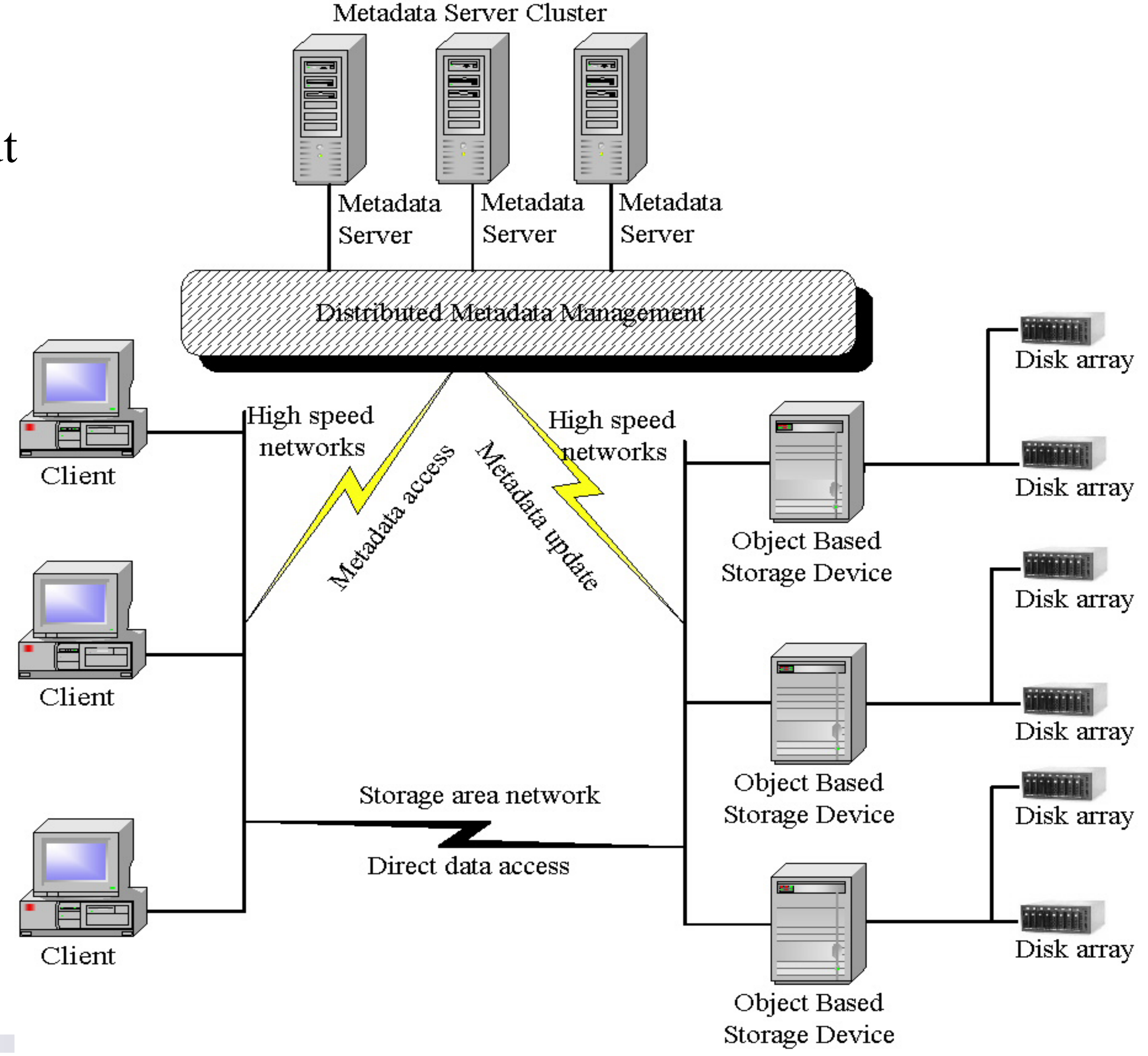
Efficient Metadata Management in Large Distributed Storage Systems

Scott Brandt, Ethan Miller, Darrell Long, and Lan Xue
Storage Systems Research Center
University of California, Santa Cruz
April, 2003

UC Santa Cruz

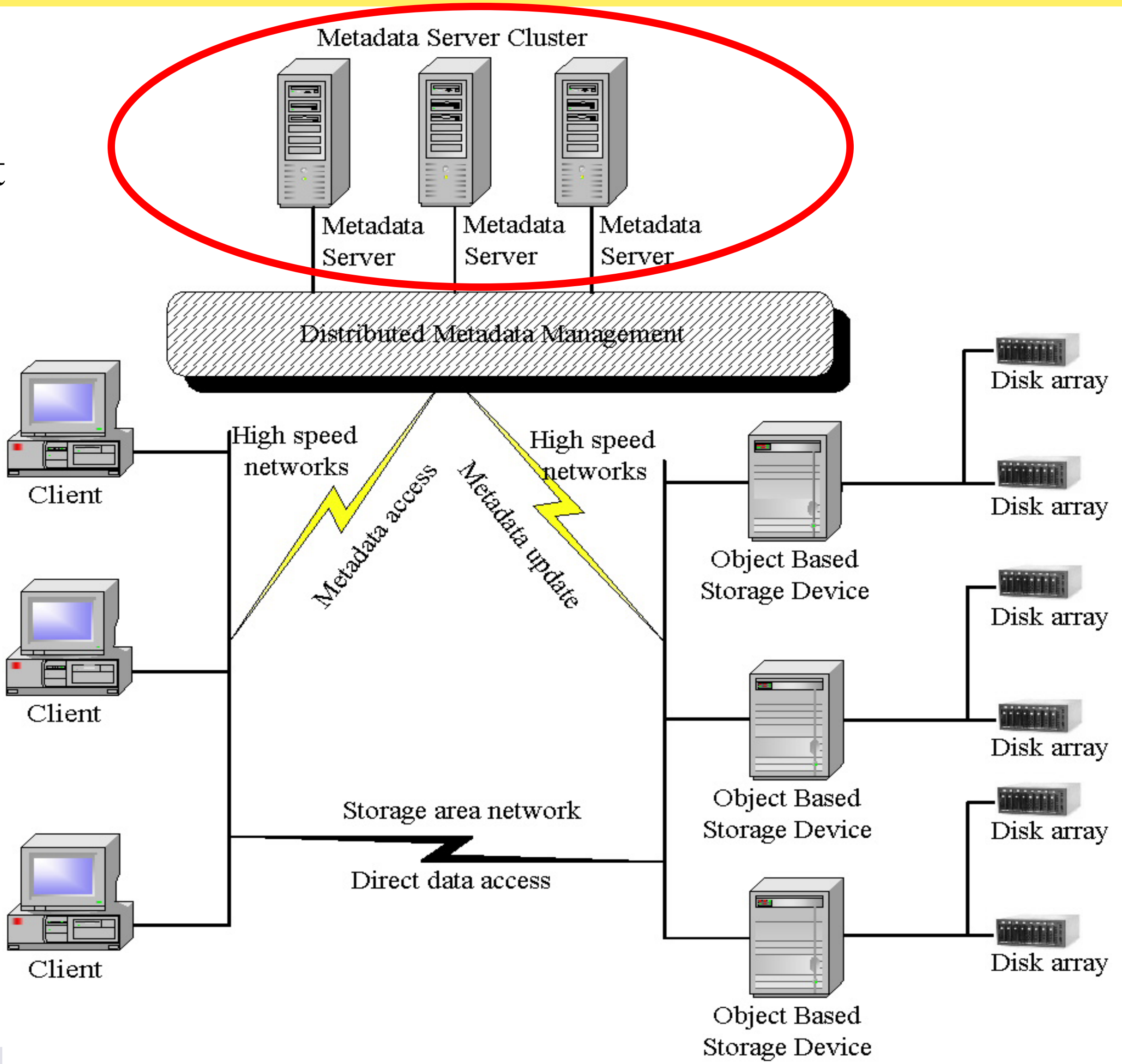
OSD Storage System Overview

- 2PB data (billions of files)
- 100 GB/sec throughput
- 10,000 client nodes active simultaneously
  - To different directories, same directory, or even same file
- Research issues:
  - OSD FS
  - Reliability
  - Data distribution
  - Metadata server internals
  - Metadata server cluster architecture



OSD Storage System Overview

- 2PB data (billions of files)
- 100 GB/sec throughput
- 10,000 client nodes active simultaneously
  - To different directories, same directory, or even same file
- Research issues:
  - OSD FS
  - Reliability
  - Data distribution
  - Metadata server internals
  - **Metadata server cluster architecture**



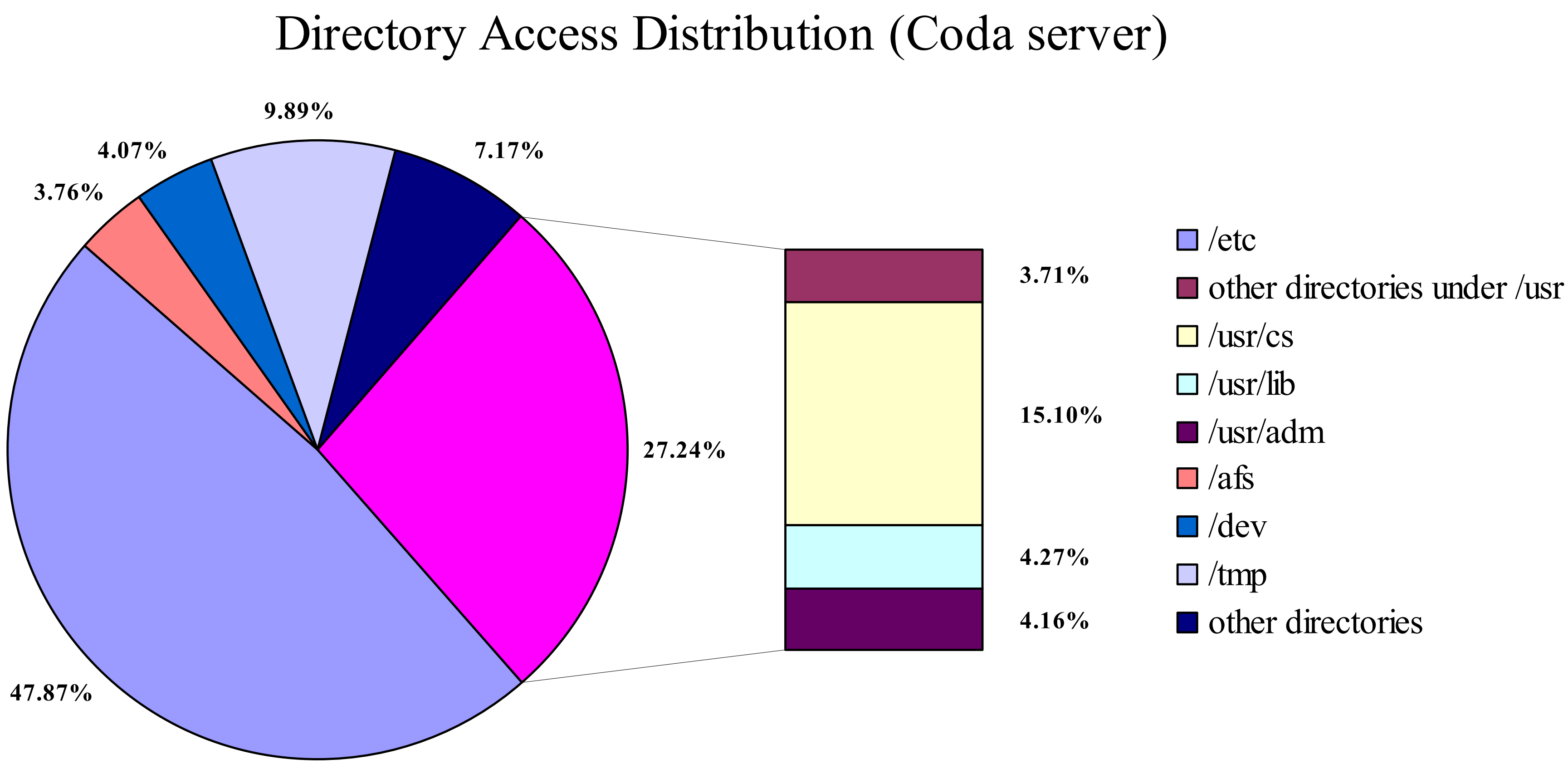
Metadata Server Cluster Goals

- POSIX-compliant API
  - Standard UNIX-style file and directory semantics
- High Performance
  - Efficient metadata access
  - Efficient directory operations
  - Efficient access control
  - High degree of parallelism
- Scalability
  - Performance scales with the number of metadata servers
  - Uniform namespace
  - Load balancing among metadata servers under various conditions
  - Easy addition and removal of metadata servers

Background: Directory Subtree Partitioning

- Hierarchical namespace partitioned by directory subtrees (e.g. NFS)
- Pros:
  - Supports standard directory semantics
  - Efficient access to multiple files in same directory
- Cons:
  - Bottlenecks with high concurrent accesses
  - Coarse granularity of load balancing
  - Adding or removing metadata servers is costly
    - Difficulty to manage
    - May have to move a significant amount of metadata

Sample Workload



- Conclusion: some directories are MUCH more popular than others.

Background: Pure Hashing

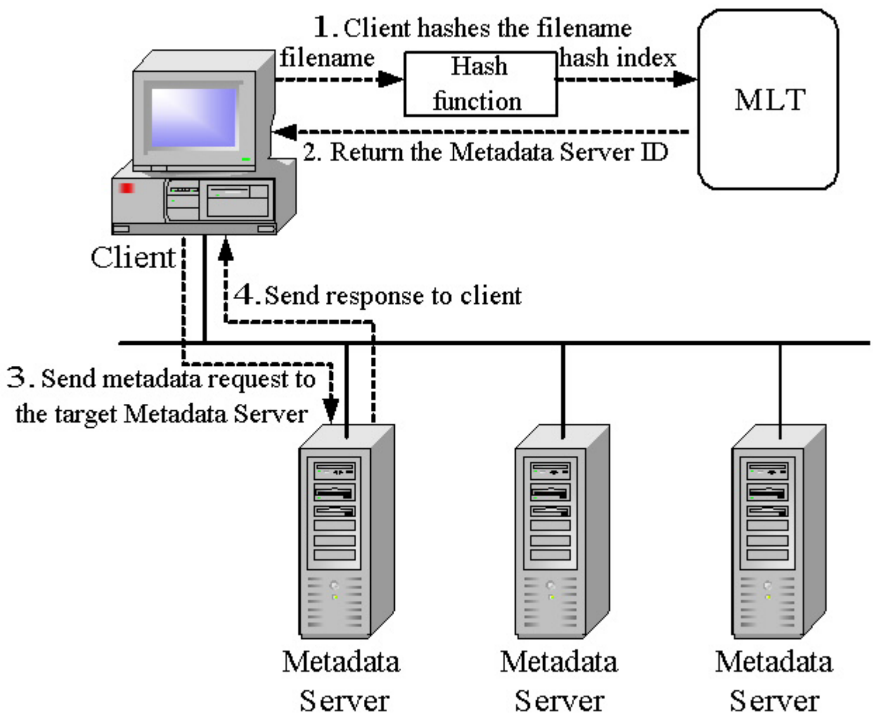
- Namespace widely distributed among the metadata servers based on hash of file or pathname (e.g. Vesta)
- Pros:
  - One-request metadata lookup
  - Bottleneck avoidance
- Cons:
  - Hard to support standard directory semantics
    - ls, directory permissions, etc.
  - Adding or removing metadata servers is costly
    - May have to move most of the metadata

Lazy Hybrid Metadata Management

1. Indirect hash-based metadata location
2. Hierarchical directories
3. Lazy metadata relocation
4. Dual-ACL access control
5. Metadata update logging

Indirect Hash-based Metadata Location

- Hash of pathname is used as an index into the Metadata Lookup Table (MLT)
  - The MLT is a global data structure – cached everywhere, updated infrequently
- MLT location specifies which metadata server contains the metadata
  - Provides for efficient addition and removal of metadata servers from the cluster
  - Updated only when metadata servers are added or removed
  - Only affected metadata is moved
- Result
  - One-request lookup
  - Fine-grained load balancing



Hierarchical Directory Structure

- Directories contain locations of file metadata
- Each metadata object is accessible both by hashing the pathname and by traversing the directory tree
- Directories are updated synchronously
  - Directory lookup always locates metadata
- Result
  - Standard directory semantics are supported

Lazy Metadata Relocation

- Several operations change location of metadata
  - Renaming a file or directory, adding or removing a metadata server
- Moving everything immediately can take a long time (but if it's not moved a metadata lookup may fail)
- Solution: Move directory and file metadata lazily, as it is accessed
  - The metadata server looks in the parent directory to determine the location, the metadata is moved to the new location, and the request is processed
  - Can proceed recursively if the parent directory also needs to move
- Result:
  - Metadata can always be located
  - Metadata is always correctly moved to new location
  - Movement overhead is distributed
  - Can also be accomplished in the background

Dual-ACL Access Control

- Hierarchical directory semantics expect path traversal to determine permissions
  - Disallows direct hash-based metadata lookup
- Solution: Dual ACLs allow for direct lookup
- File permissions
  - Ordinary file permissions
- Path Permissions
  - "Intersection" of file permissions and parent directory permissions
  - Computed at file creation and updated as appropriate

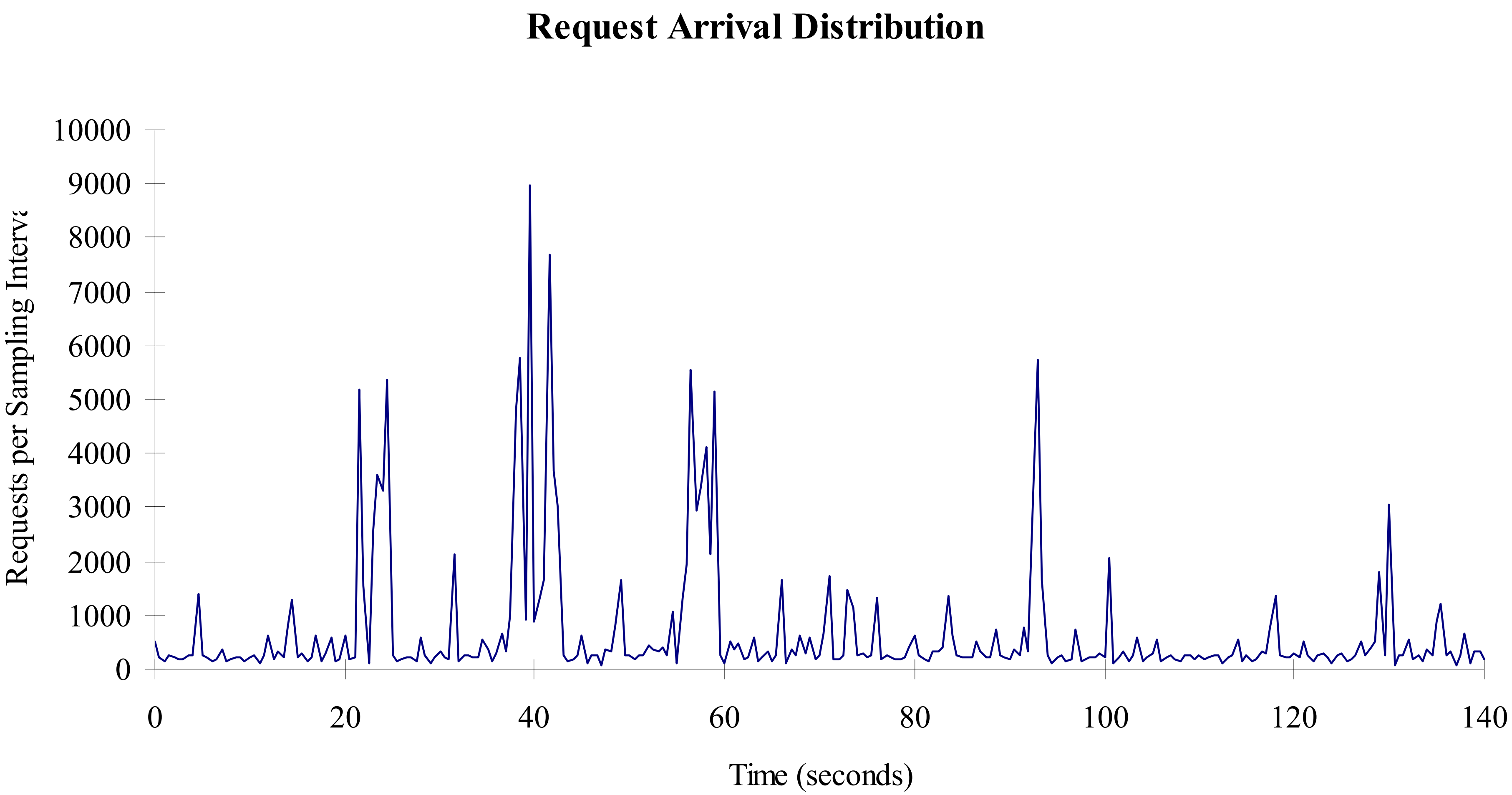
Metadata Update Logging

- Directory rename and permission change require updates to a lot of metadata
- Solution: large metadata updates are synchronously broadcast and recorded in a log on each server
- Metadata is compared to the log and appropriate updates are applied before each request is processed
- Update timestamp allows for efficient log search
  - Metadata stores timestamp of last update compared
- Result:
  - Large metadata updates can be accomplished quickly
  - Updates can be accomplished lazily or in the background

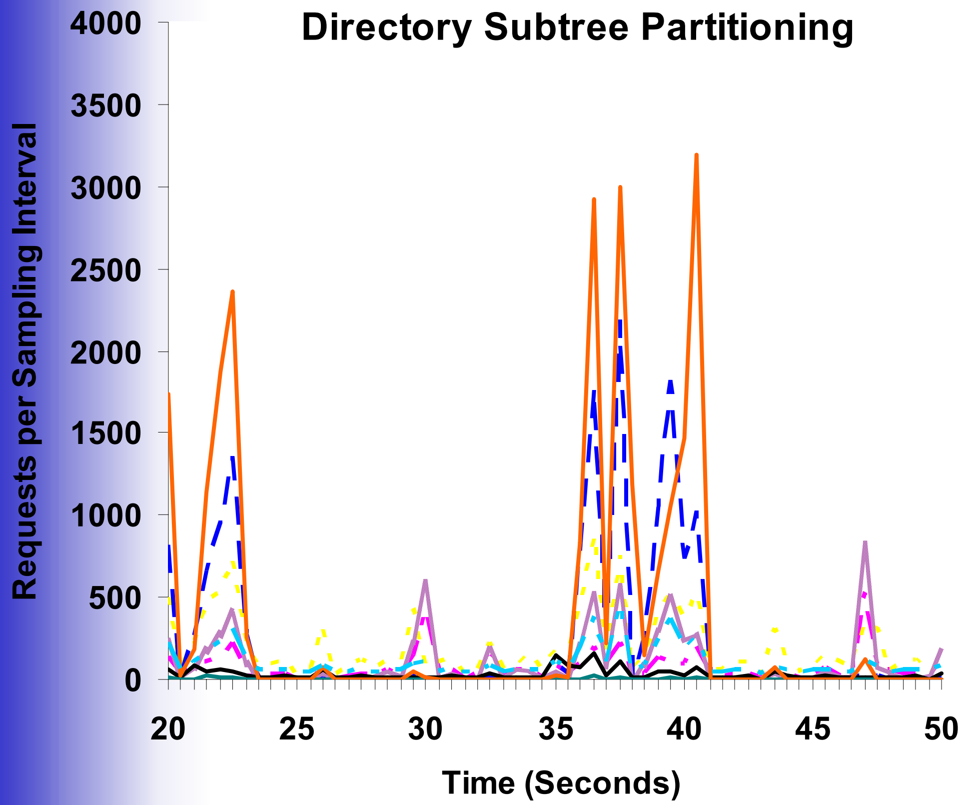
Simulation Environment

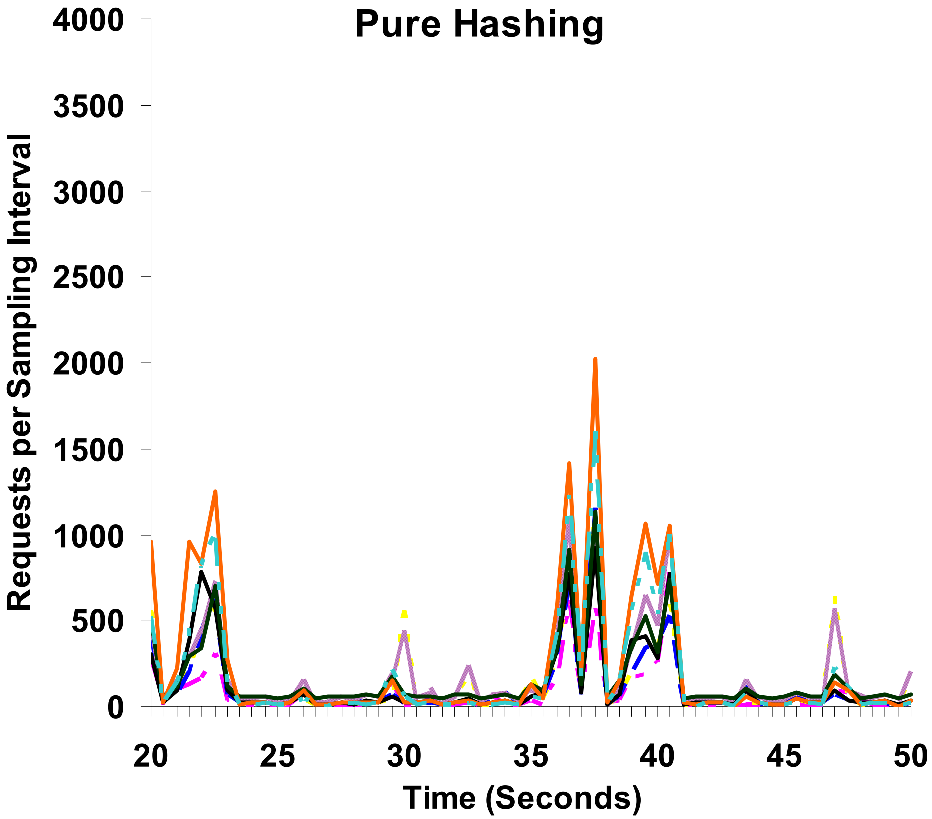
- Simulated Directory Subtree Partitioning, Pure Hashing, and Lazy Hybrid
- Server cache hit rate: 99%
- Client cache hit rate: 100%
- Disk I/O cost (1KB): 15msec
- Memory access cost (1KB): 15µsec
- Network transfer cost (1KB): 100µsec
- Asynchronous write every 30 sec
- Simulation traces: an 8-day file server trace scaled by a factor of 5,000
- Sampling Interval: 0.5 sec
- Number of metadata servers: 8

Request Arrival Distribution



Throughput: Directory Subtree Partitioning and Pure Hashing

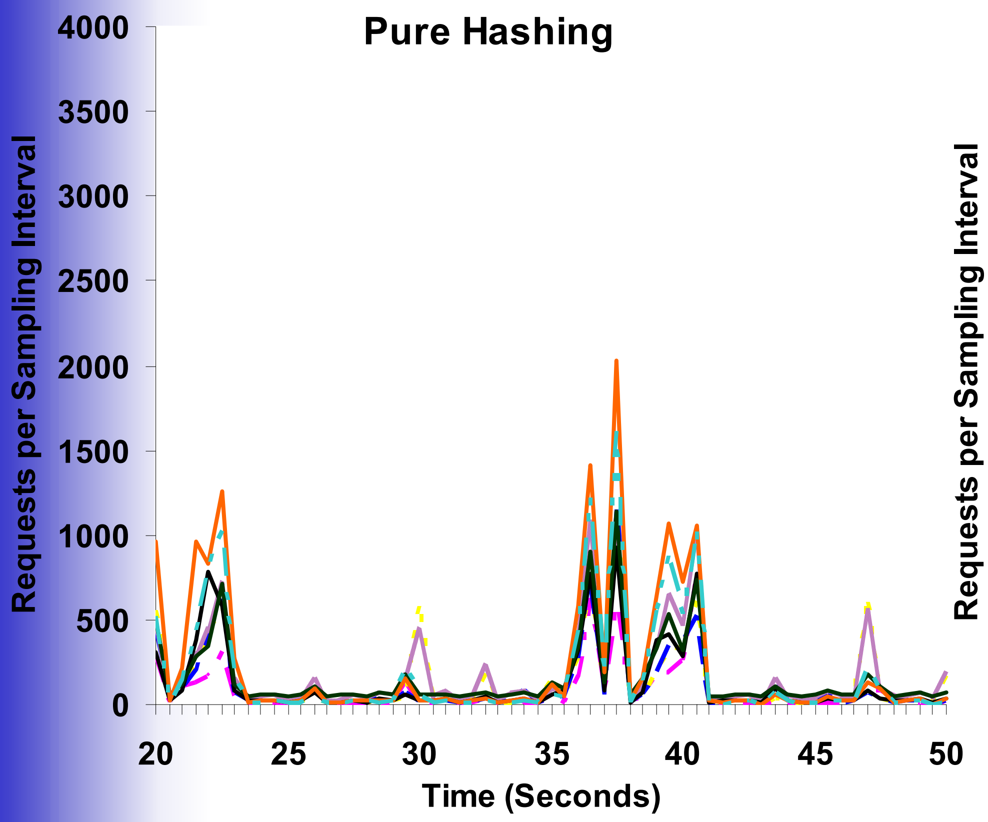


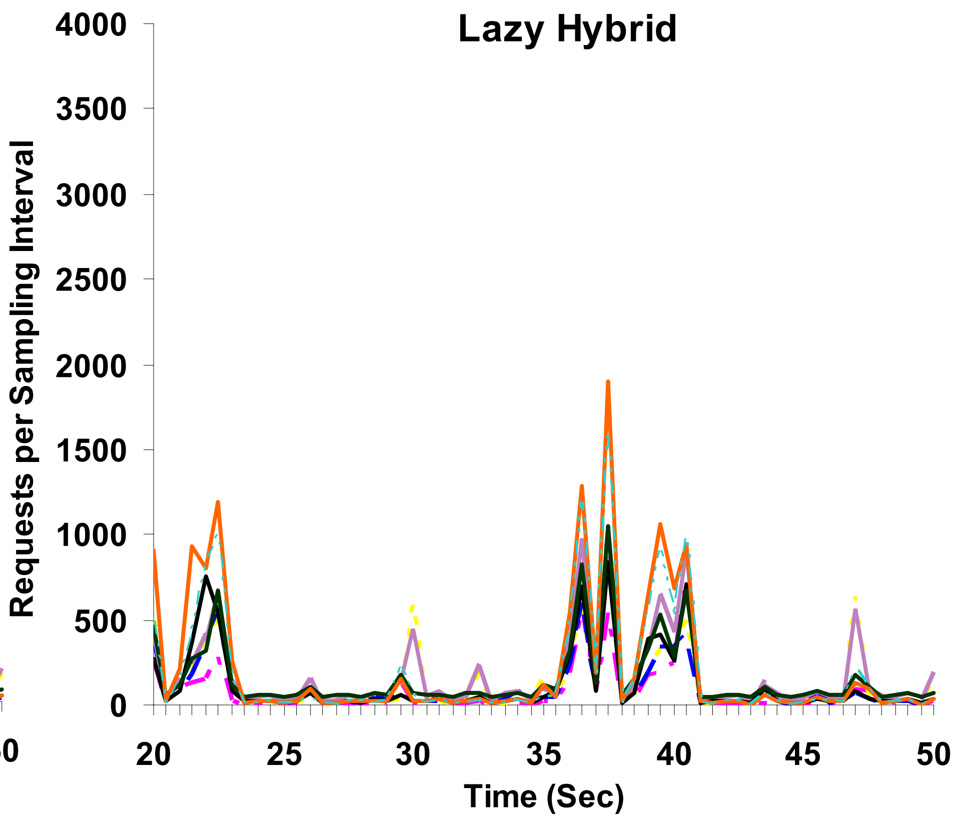


Maximum throughput difference between metadata servers at a given time point:

max/min = 42.05 max/min = 3.44

Throughput: Pure Hashing and Lazy Hybrid

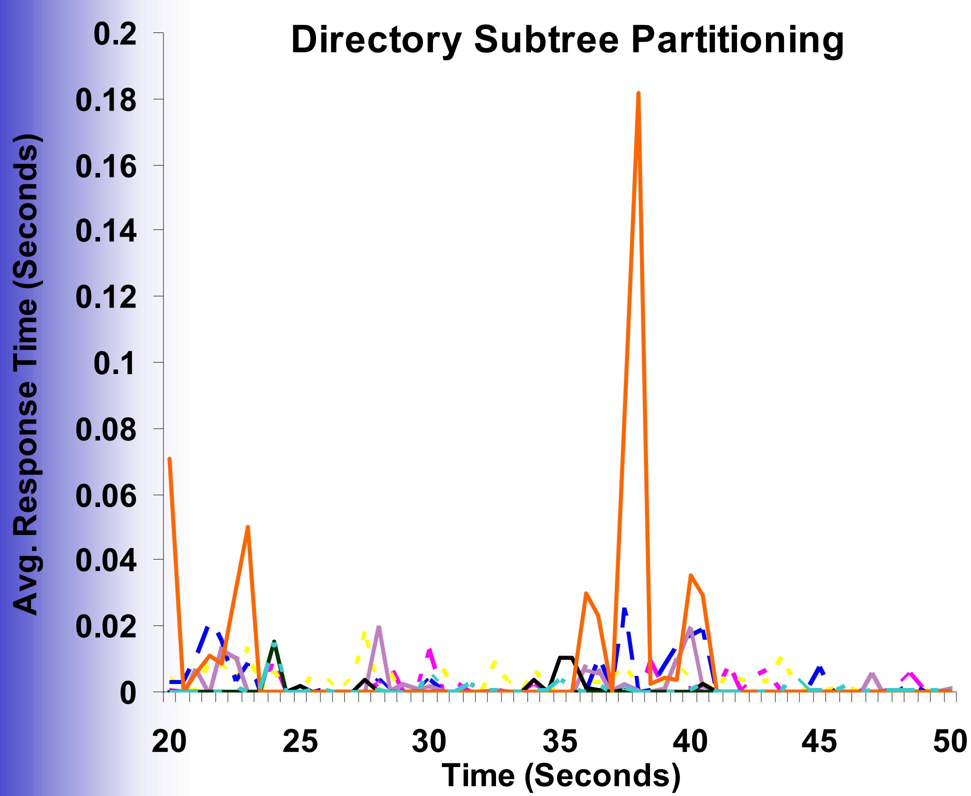


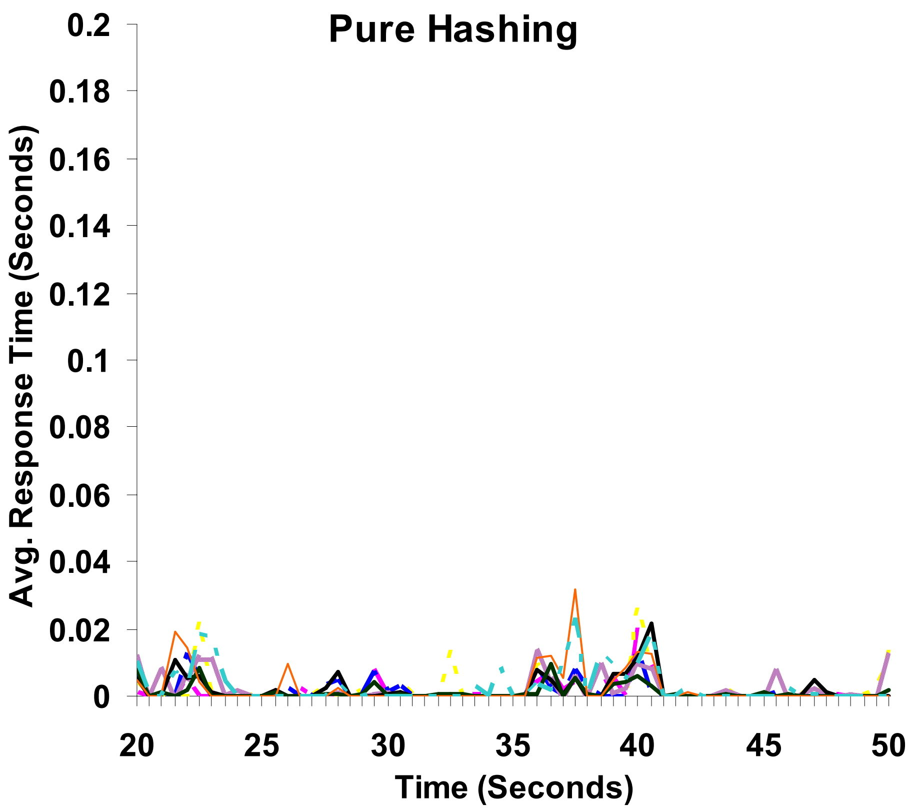


Maximum throughput difference between metadata servers at a given time point:

max/min = 3.44 max/min = 3.49

Response Time: Directory Subtree Partitioning and Pure Hashing

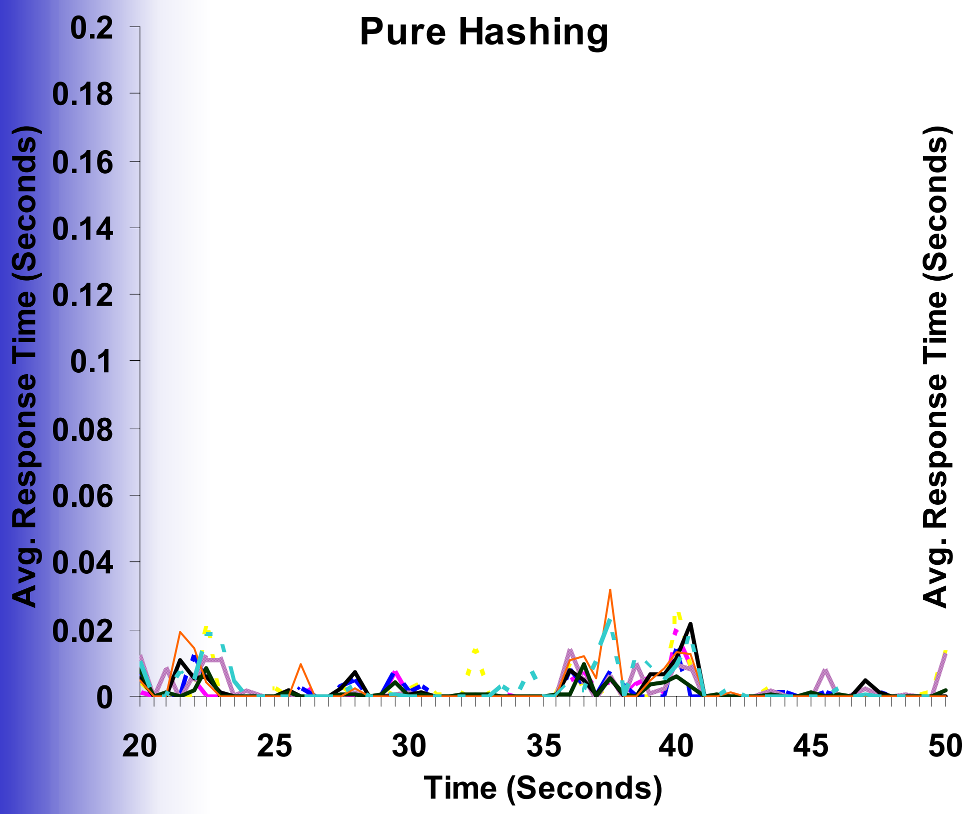


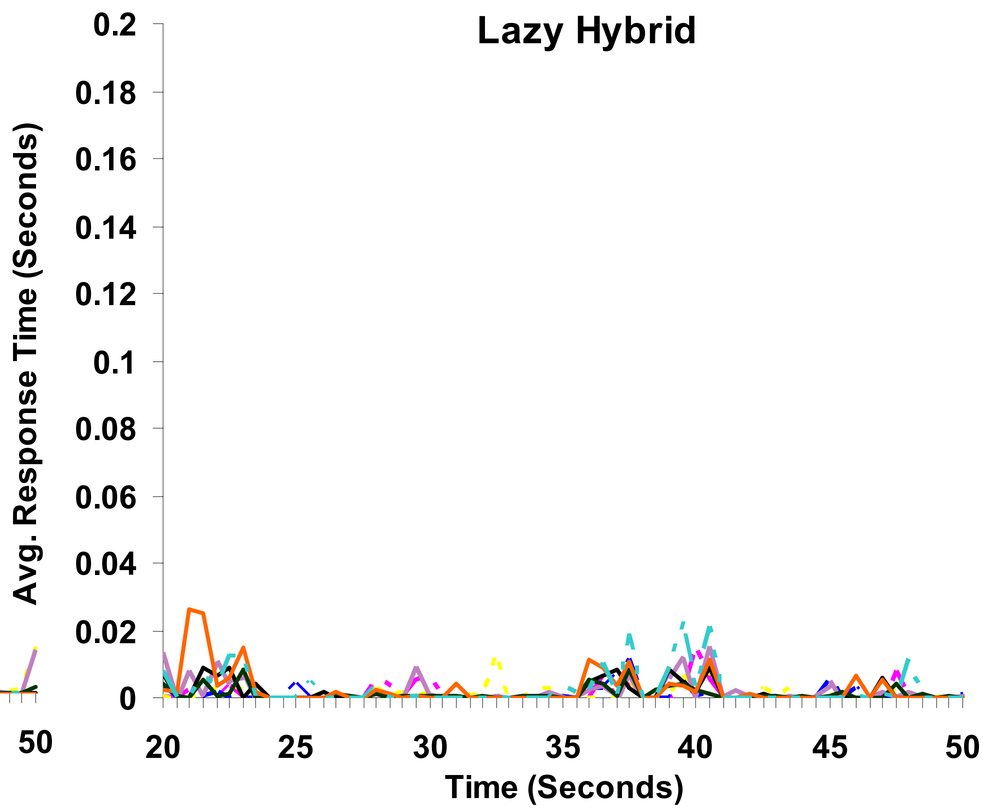


Maximum response time difference between metadata servers at a given time point:

max/min = 5680.25 max/min = 762.72

Response Time: Pure Hashing and Lazy Hybrid

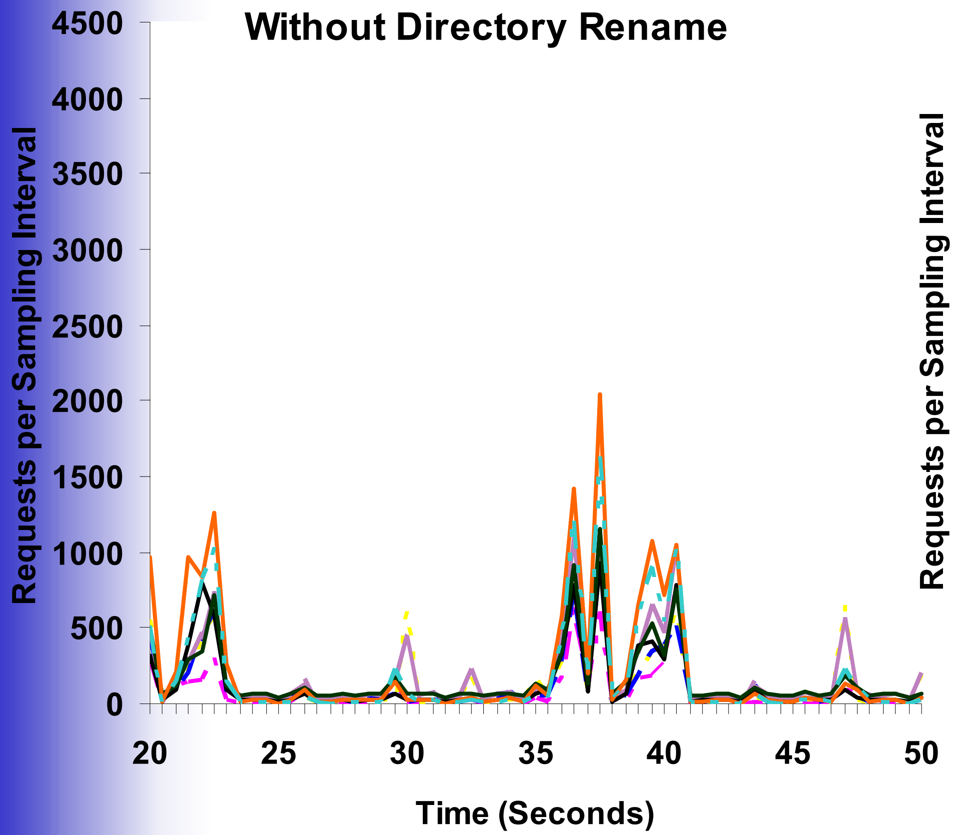


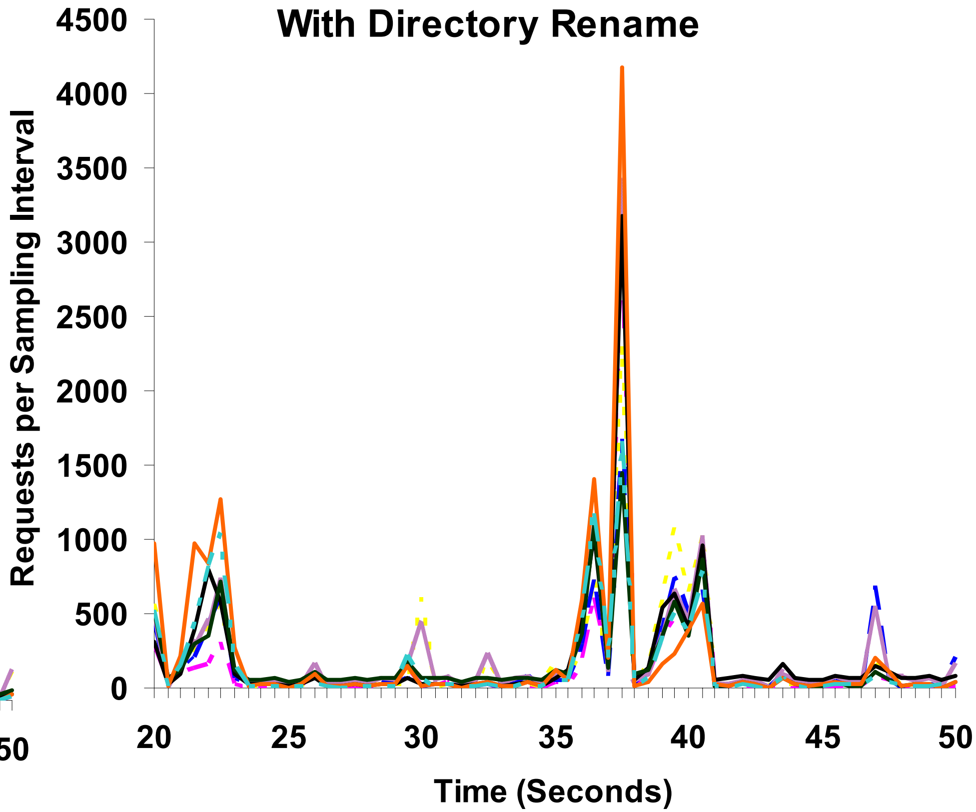


Maximum response time difference between metadata servers at a given time point:

max/min = 762.72 max/min = 802.63

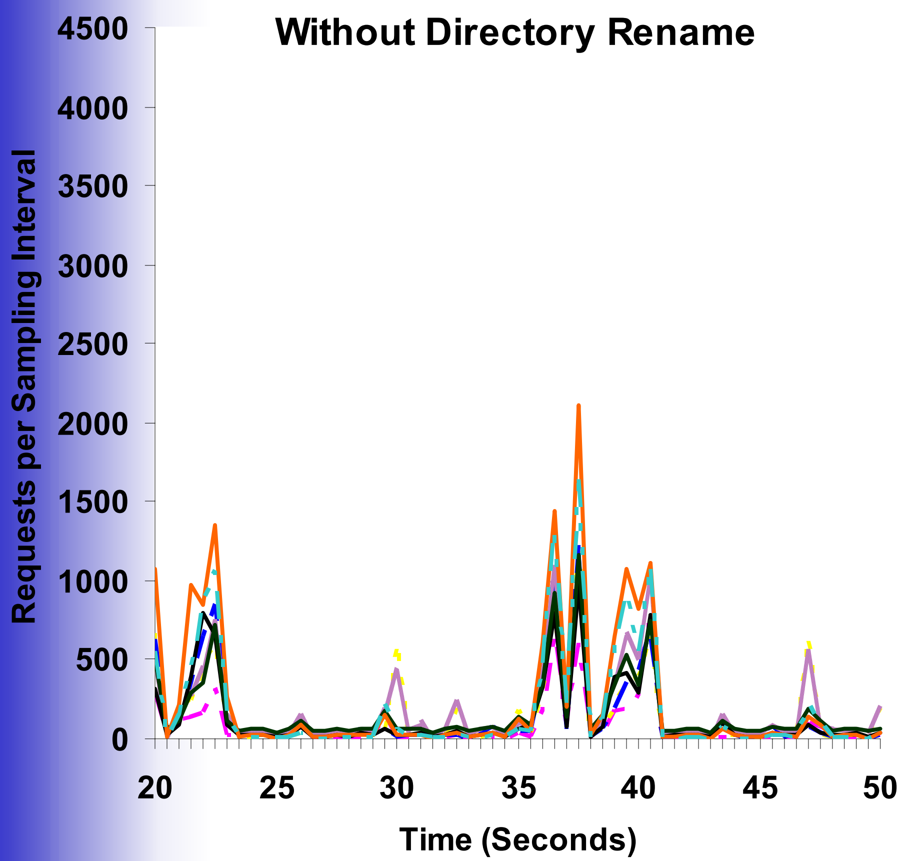
Workload Variation with Directory Rename: Pure Hashing

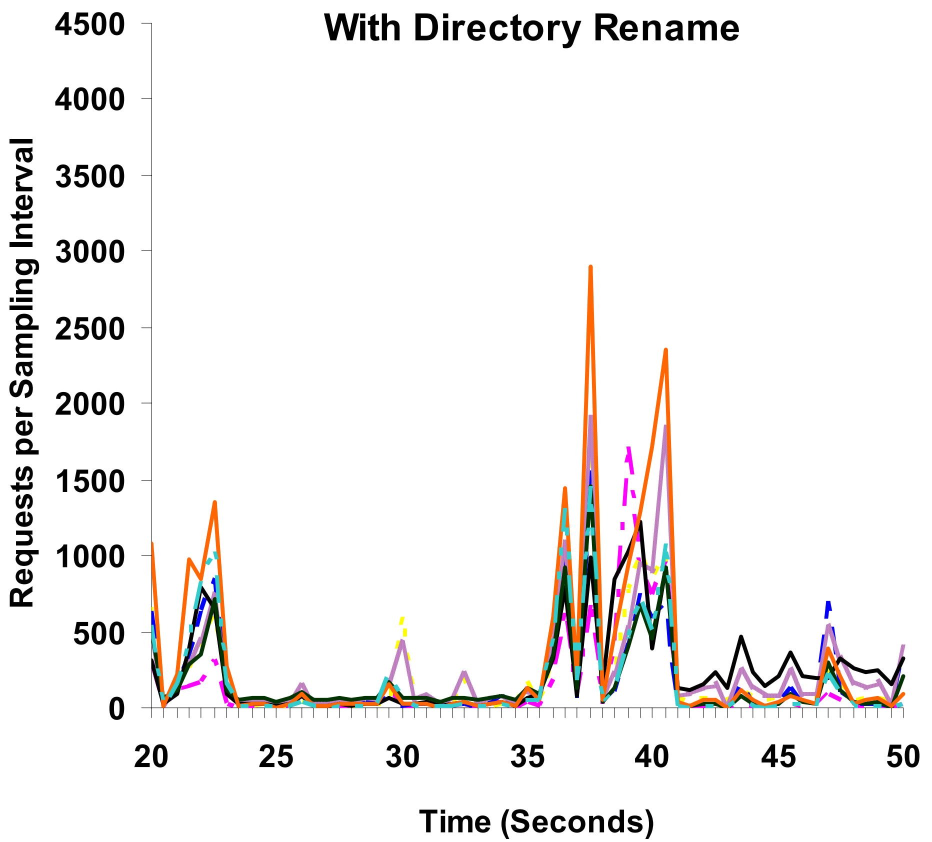




Directory rename at time 37.5 at depth 6 with 768 children

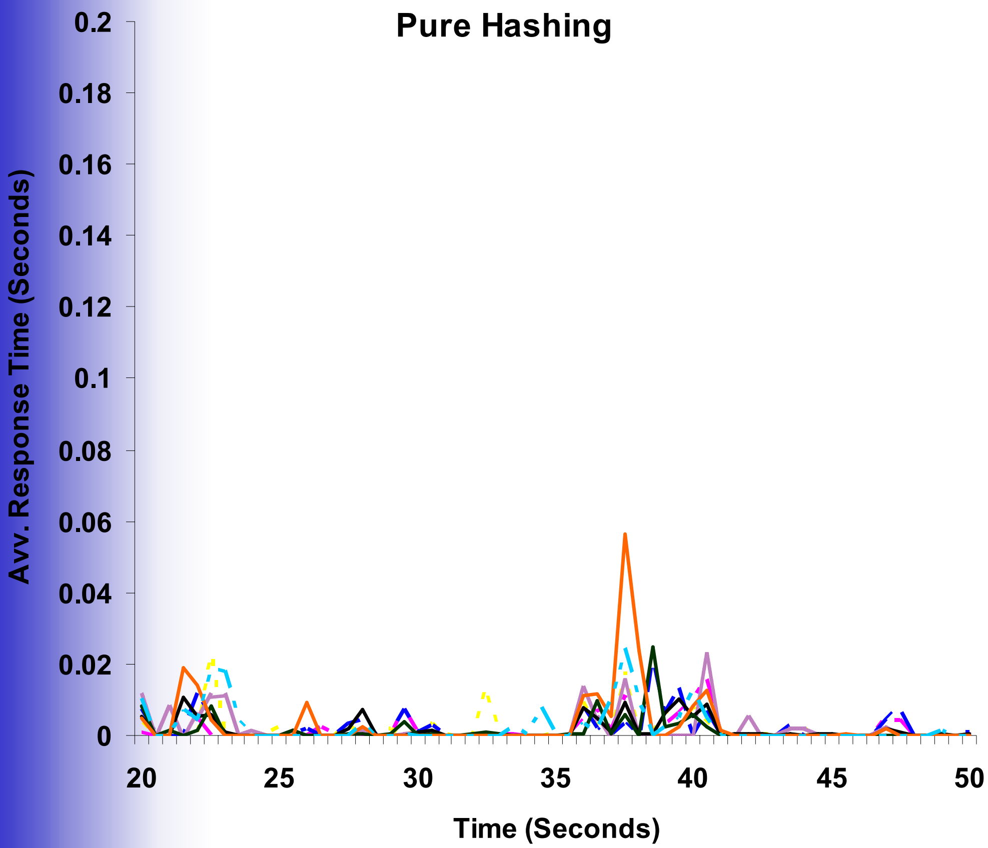
Workload Variation with Directory Rename: Lazy Hybrid

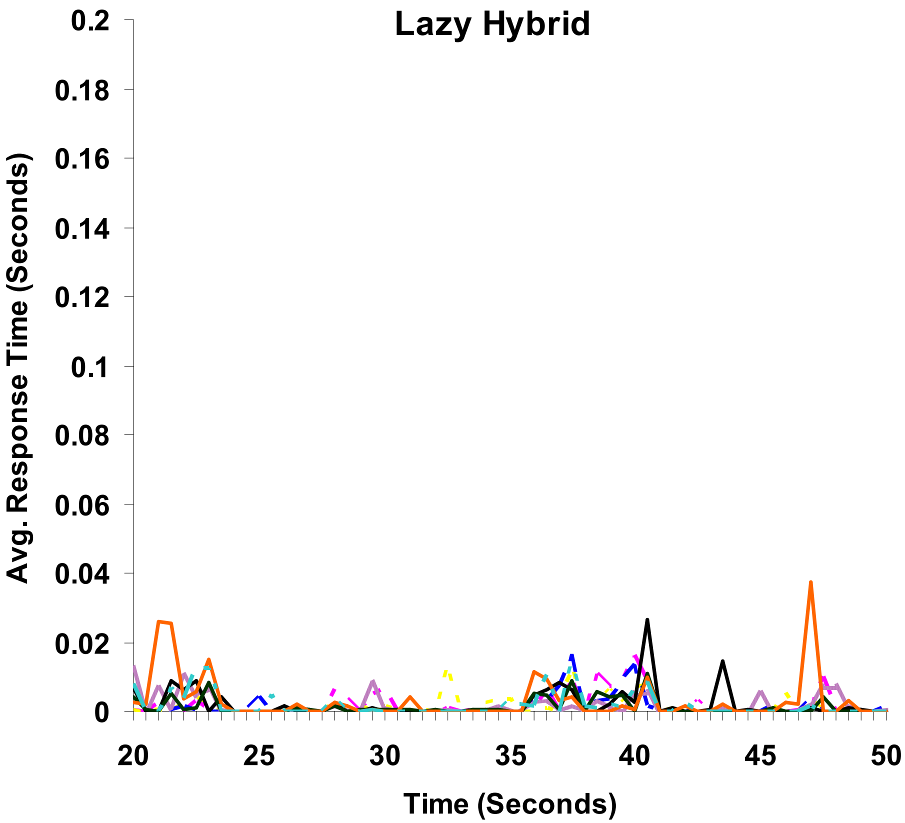




Directory rename at time 37.5 at depth 6 with 768 children

Response Time with Directory Rename: Pure Hashing and Lazy Hybrid





Response time increases sharply immediately

Delayed and distributed increase in response time

Conclusion

- Directory Subtree Partitioning supports standard file/directory semantics
  - But has scalability and bottleneck problems
- Pure Hashing efficiently balances the workload among servers
  - But has difficulty supporting standard directory semantics and incurs high overhead during some operations
- Lazy Hybrid metadata management combines the best of these two approaches, and sometimes does better than both
  - Provides both standard directory semantics and efficient metadata access