A transparently scalable metadata service for the Ursa Minor storage system

Shafeeq Sinnamohideen

Raja Sambasivan, James Hendricks, Likun Liu, Gregory R. Ganger

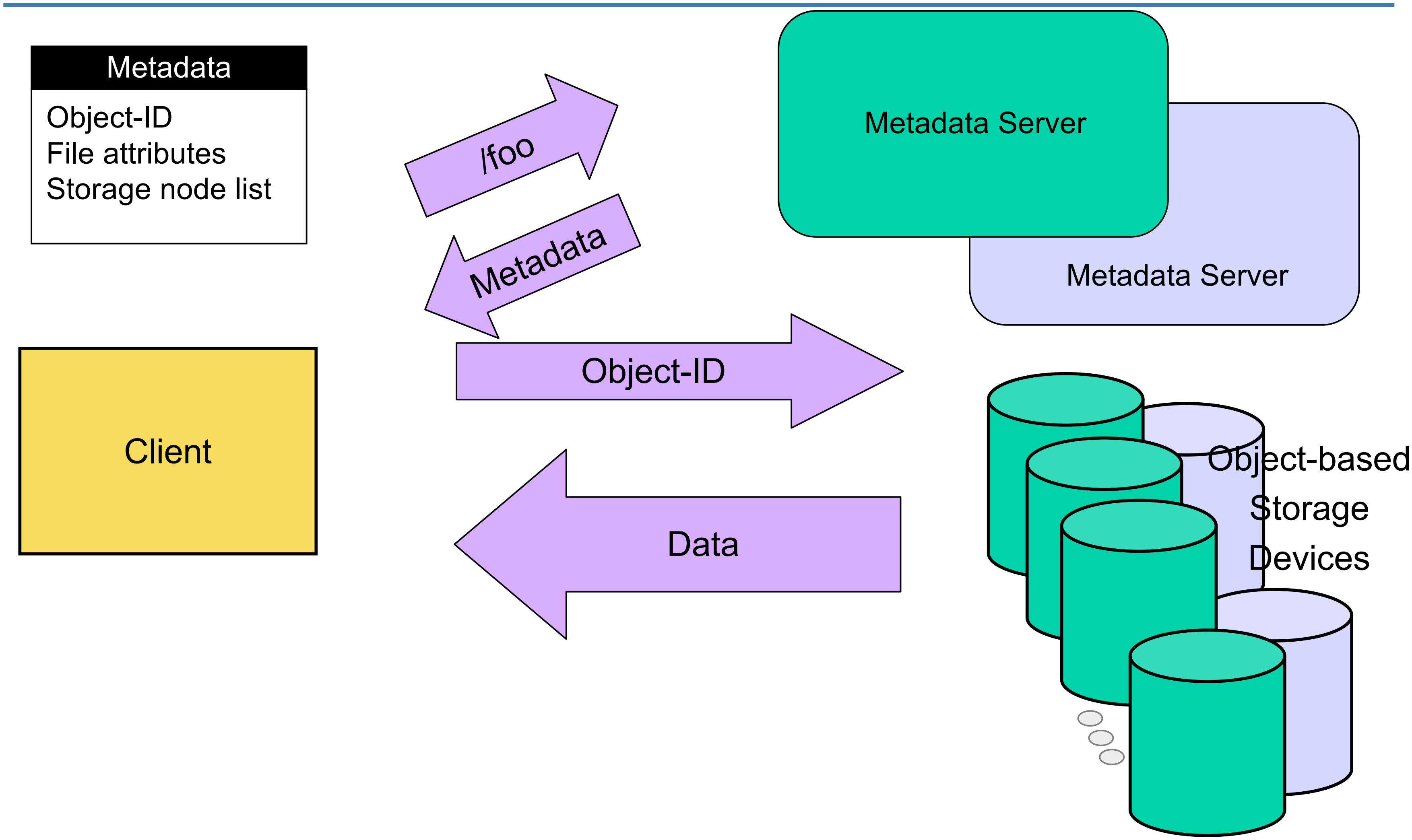
PARALLEL DATA LABORATORY

Carnegie Mellon University

Ursa Minor

- Prototype of a Self-* storage system [FAST05]
- Direct-access system model
  - Data path for bulk data
  - Metadata path for attributes
  - Similar to NASD, Panasas, PVFS, Lustre, etc.
- Research questions
  - How to automate management?
  - How to build a versatile system?
    - *This talk : one hard problem with simple solution*

Ursa Minor Overview



Desired properties

- Scalability
  - Adding servers increases capacity
  - Ideally the increase is proportional
- Transparency
  - Users don’t care which server is used
  - Always provide consistent semantics
    - Atomic operations are a useful building block
    - Standard compliance
    - Difficult for programmers to do without

Maintaining semantics

Easy for the data path:

- Operations affect a single file
- Only one server involved in each op

Some metadata ops can affect two items:

- Renaming a file to different directory
- Parent & child
- *Could involve two servers*

Handling multi-server ops

1. Only allow single-server ops
   - e.g.: AFS, NFS, OnTAP GX
   - Volume abstraction->limited transparency
2. Use a distributed transaction protocol
   - e.g.: Farsite
   - Complex to implement
3. Use distributed locking & shared state
   - e.g.: GPFS
   - Push complexity into lock manager

Our approach to multi-server ops

- Use the simplest possible solution
- System can already:
  - Perform single server atomic operations
  - Migrate items for load balancing

Reuse features to support multi-server ops

The idea

When a request needs multiple files:

- Migrate file's metadata to one server
- Execute the single-server code path
- Fix any load imbalance
  - Return metadata to original server
  - Move other files

Core tradeoff

- Gain simplicity through reuse
  - Unmodified single server execution
  - Unmodified migration path
- Lose some performance
  - Migration latency added to op latency

- *Expect this to be a worthwhile tradeoff*

What do we expect?

Traces of large file systems show that:

- Multi-object ops are a tiny fraction
- Most multi-object ops are parent-child
  - CREATE, DELETE
  - Parent & child on same server for locality
- Other multi-object ops extremely rare
  - RENAME: 0.005% involve 2 dirs
  - LINK: 0.120% possible *(0.005% actual)*
  - Most of these will be close in directory tree
- *Rare case doesn't have to be fast*

Metadata distribution

- Distributed key-value store for “inodes”
  - Key: Object-ID
  - Value: object metadata (attributes & layout)
- Distribute by Object-ID

Object-IDs

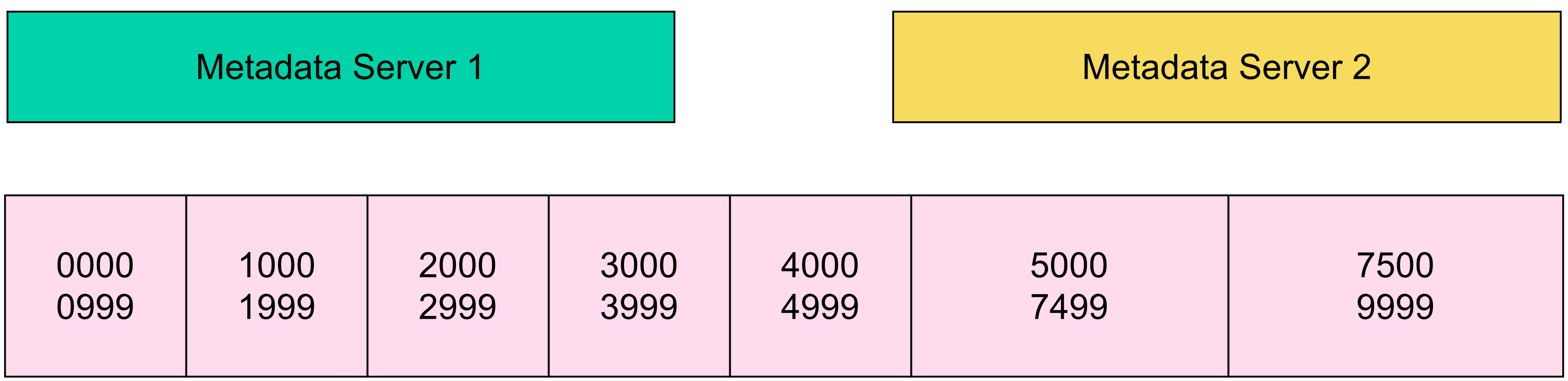
Metadata distribution

- Distributed key-value store for "inodes"
  - Key: Object-ID
  - Value: object metadata (attributes & layout)
- Distribute by Object-ID
  - Partition into ranges

0000 0999	1000 1999	2000 2999	3000 3999	4000 4999	5000 7499	7500 9999

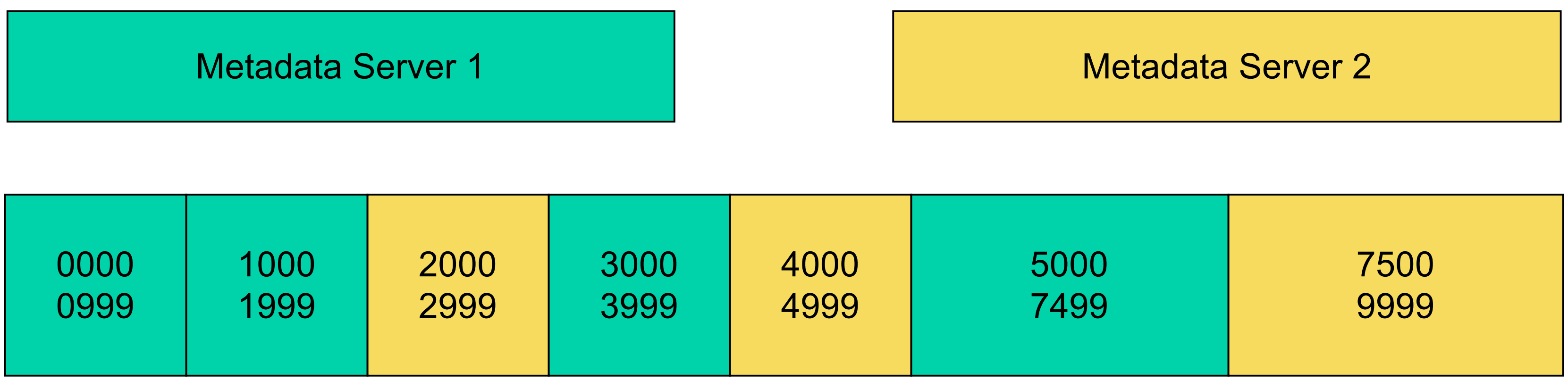
Metadata distribution

- Distributed key-value store for “inodes”
  - Key: Object-ID
  - Value: object metadata (attributes & layout)
- Distribute by Object-ID
  - Partition into ranges
  - Assign each range to a server



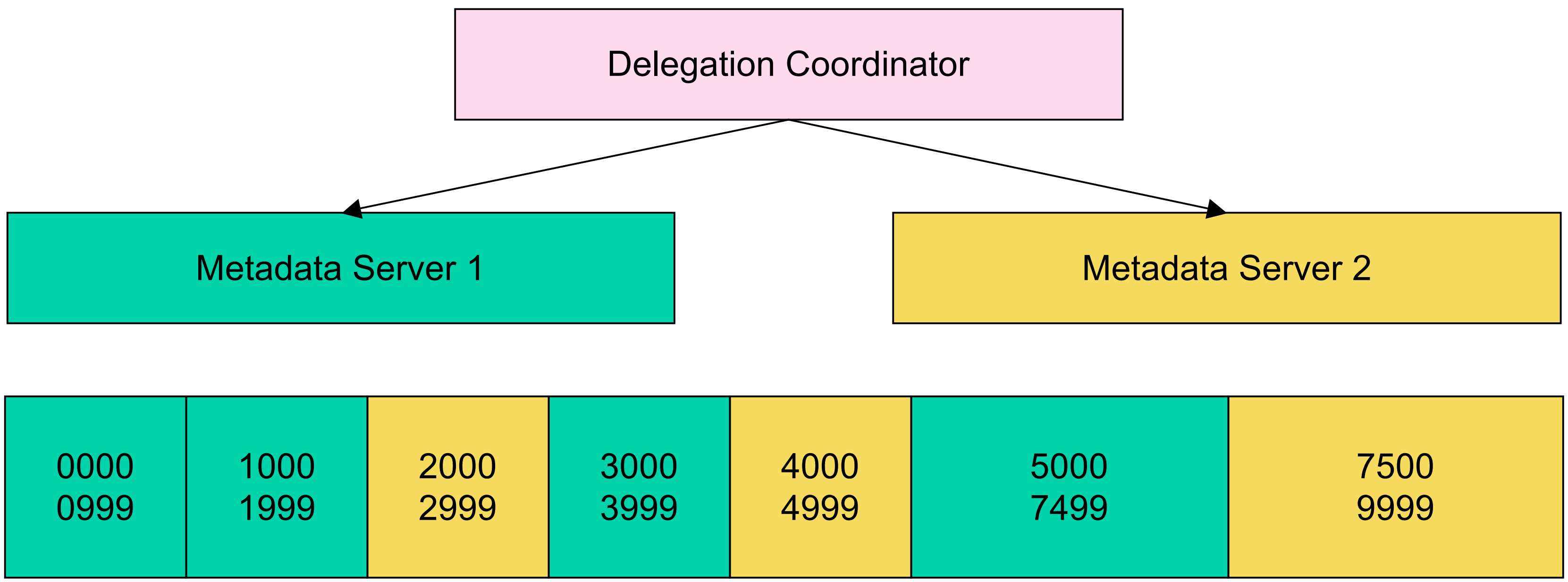
Metadata distribution

- Distributed key-value store for “inodes”
  - Key: Object-ID
  - Value: object metadata (attributes & layout)
- Distribute by Object-ID
  - Partition into ranges
  - Assign each range to a server



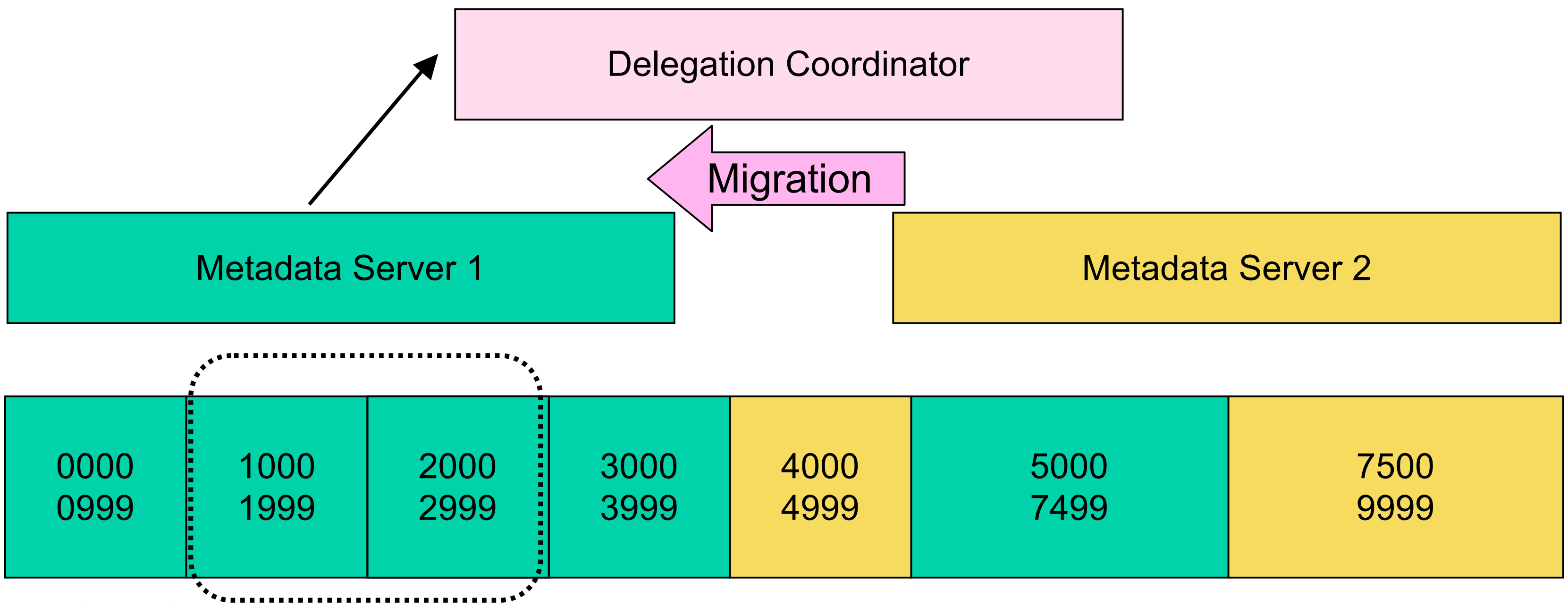
Metadata distribution

- Delegation coordinator assigns ranges
- *Range is unit of migration*
- Metadata persistently stored in data path



Multi-server operations

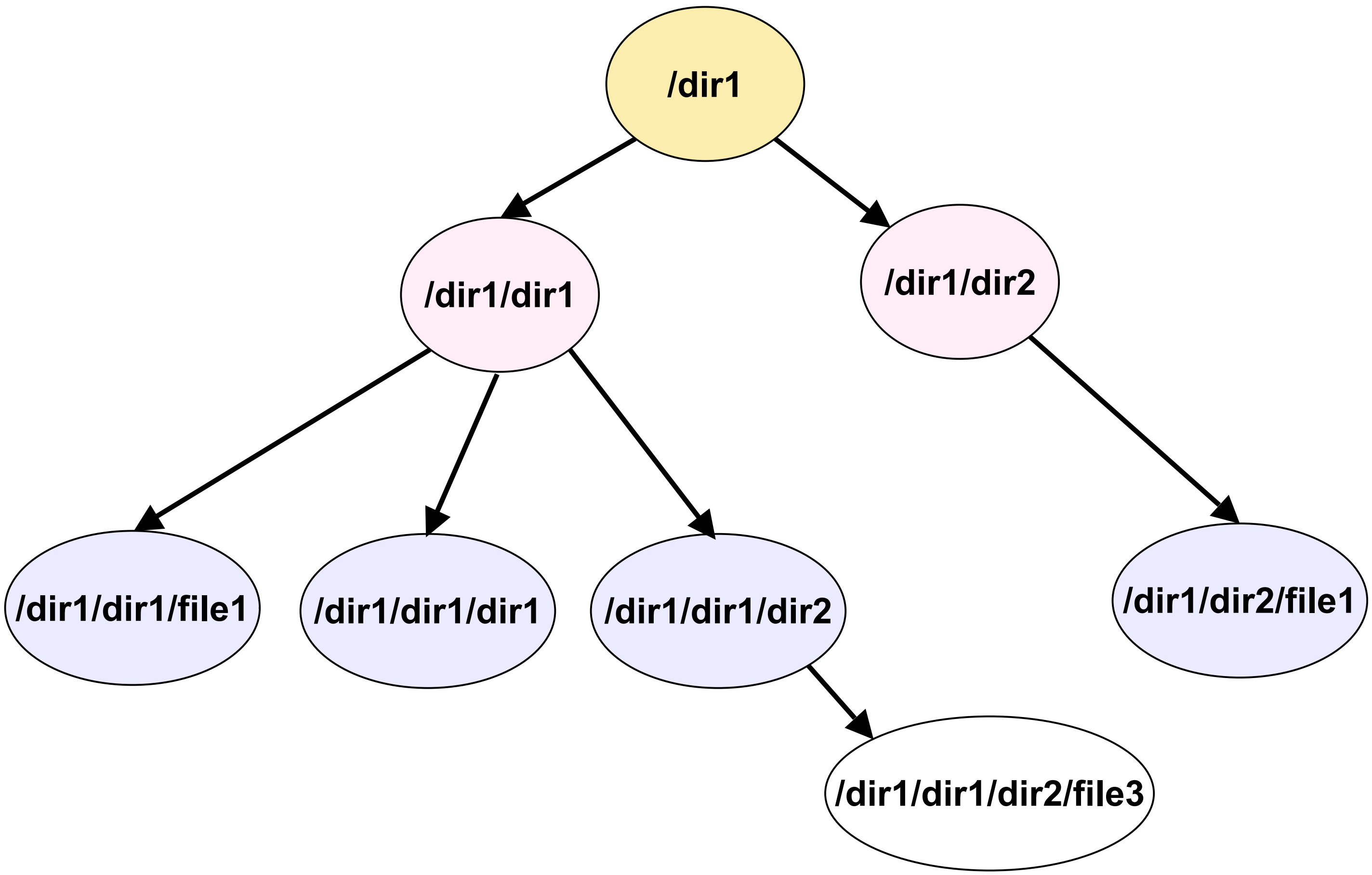
- When a metadata server needs a range :
  1. Borrow it from the server that has it
  2. Perform the operation
  3. Return it to the original server



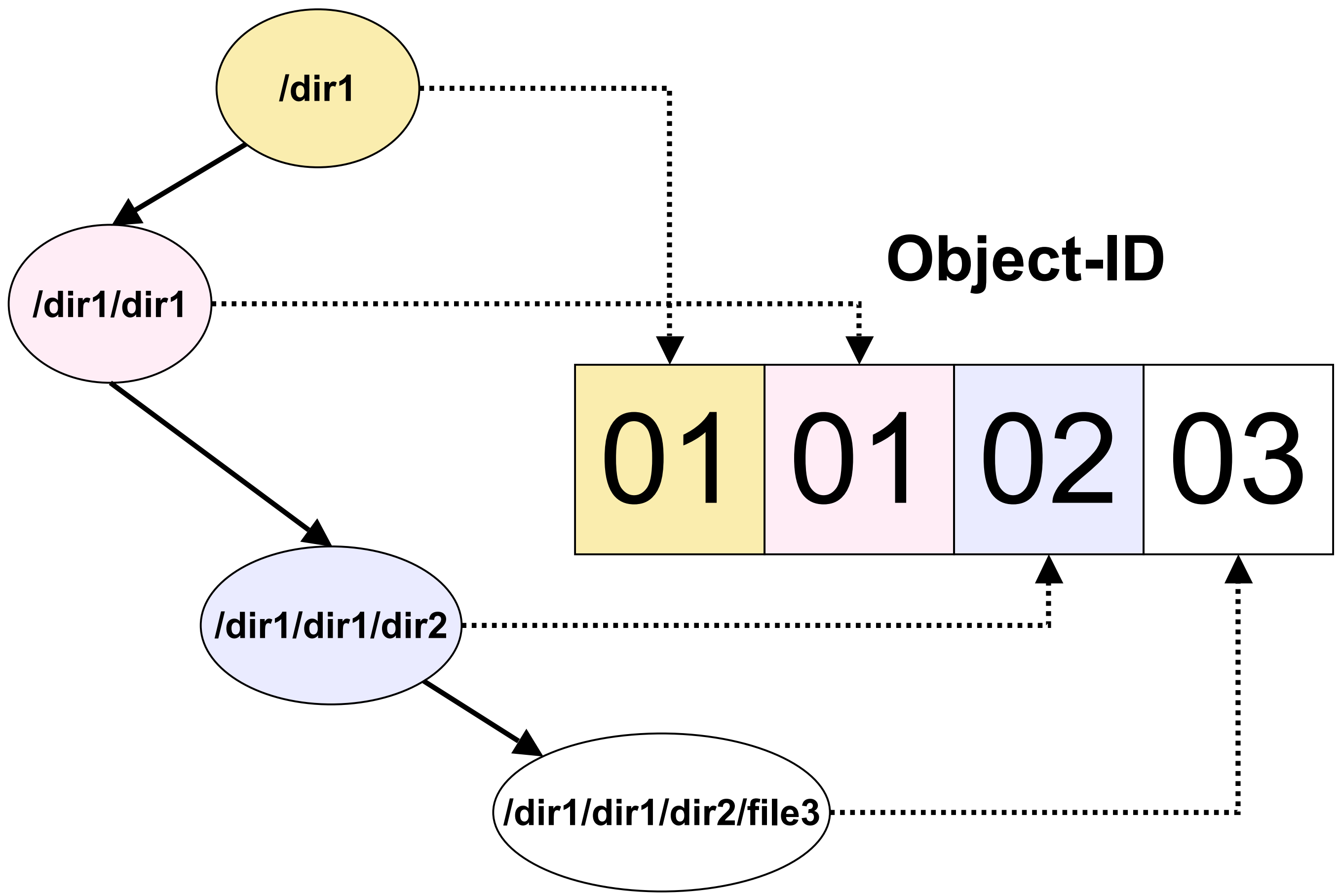
Object-IDs

- Object-ID determines which server to use
- *Assign Object-IDs to minimize multi-server ops*
  - Directory tree determines operation locality
  - Multi-file ops involve nearby directories
  - *Nearby files should get similar Object-IDs*
    - *Fall into same range*
    - *Served by same server - locality benefits*
  - *Encode hierarchy into Object-ID*
  - *Analogous to IP address subnetting*

Example tree



Object-ID assignment



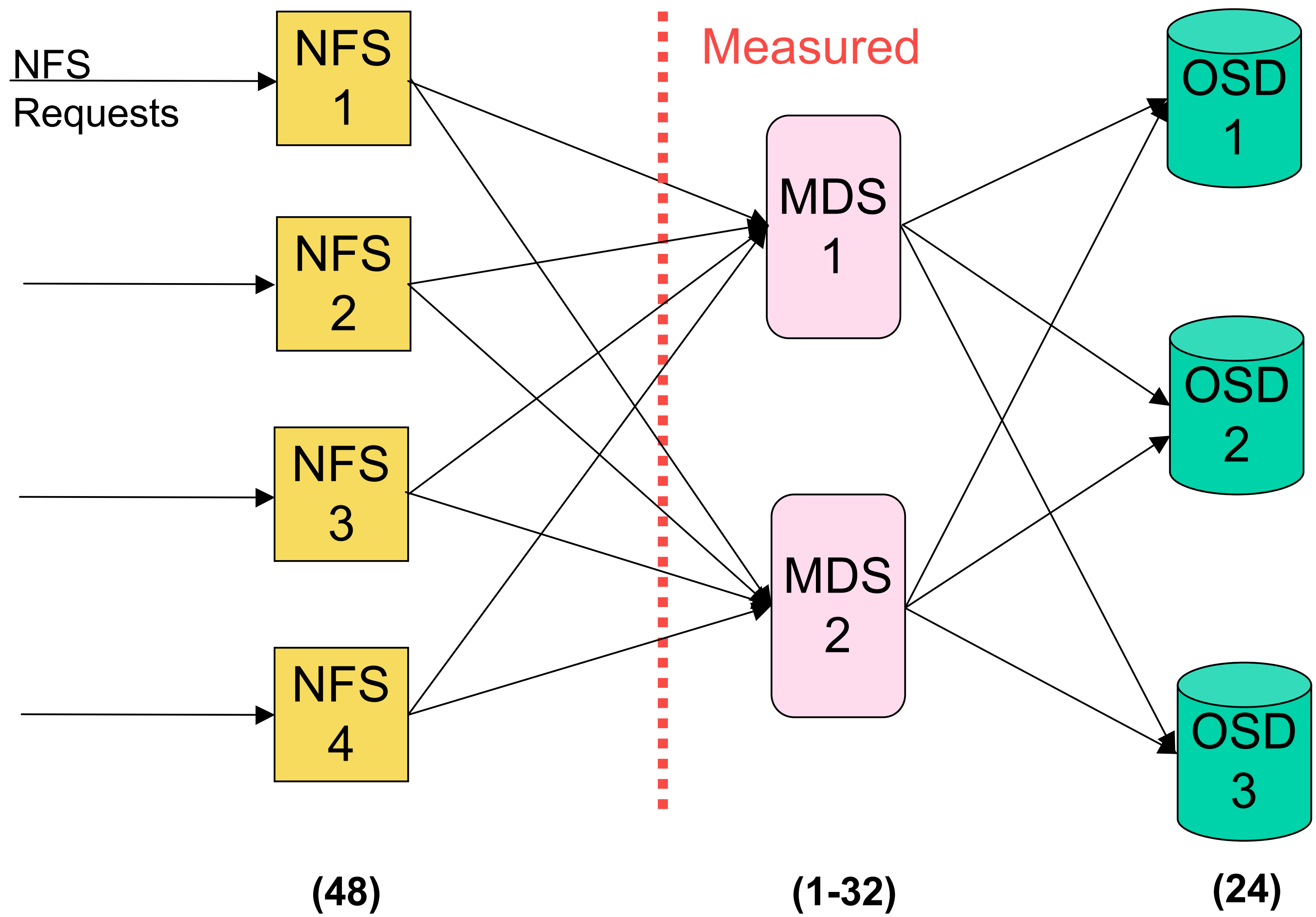
Evaluation

1. Is Metadata Service scalable?
2. Sensitivity to workload characteristics
3. Sensitivity to system parameters
4. Headroom for future workloads

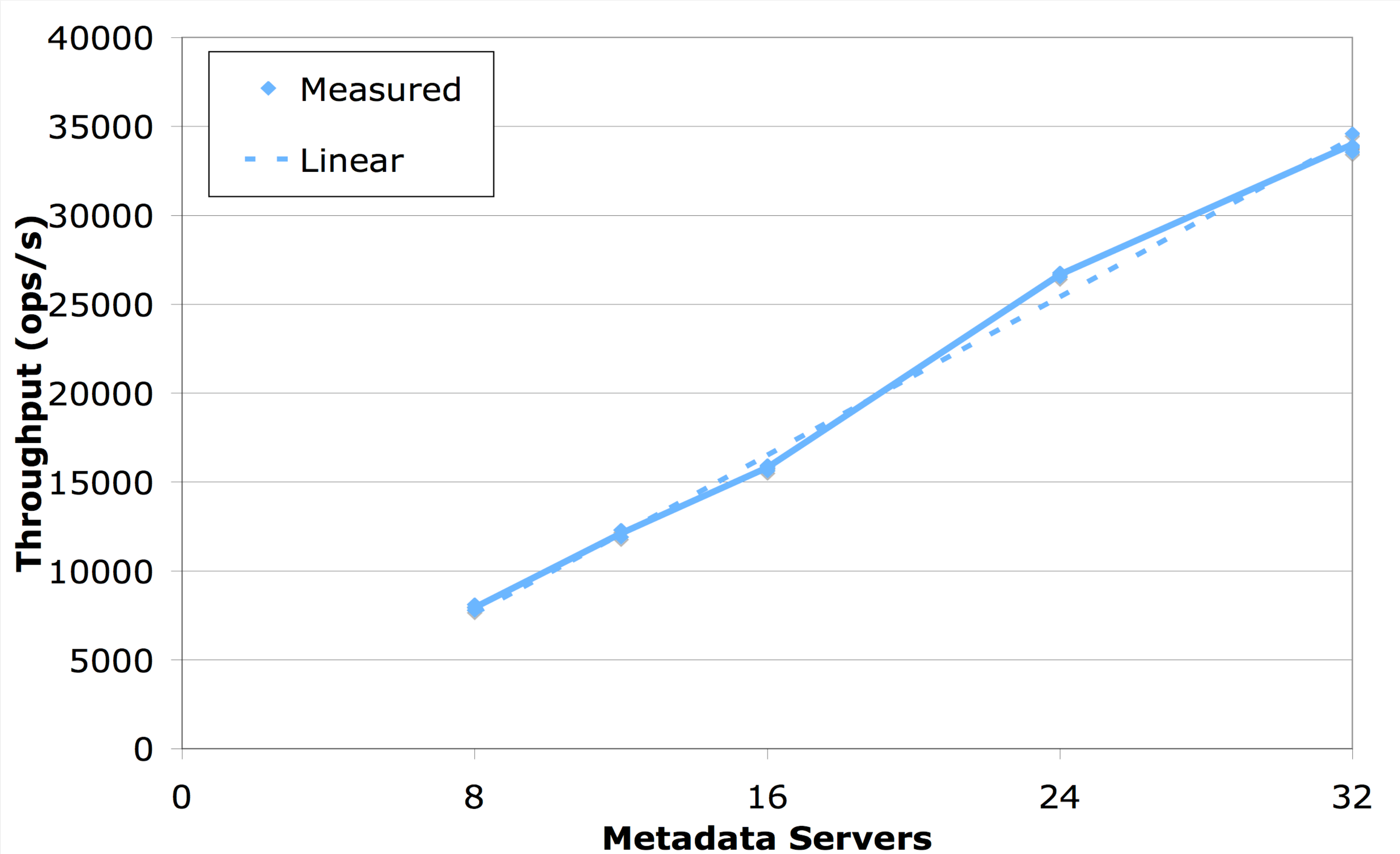
Experimental setup

- Modified SpecSFS97 NFS benchmark
  - Applied to Ursa Minor NFS head-ends
  - NFS head-end translates to Ursa Minor
  - Configured to maximize MDS load
    - 8.3 million files & directories
    - 26GB of metadata (158GB of file data)
- Vary number of metadata servers
- Rest of system is constant

Metadata traffic



Scalability w/o multi-server ops



About multi-server ops

SpecSFS97 doesn't produce any

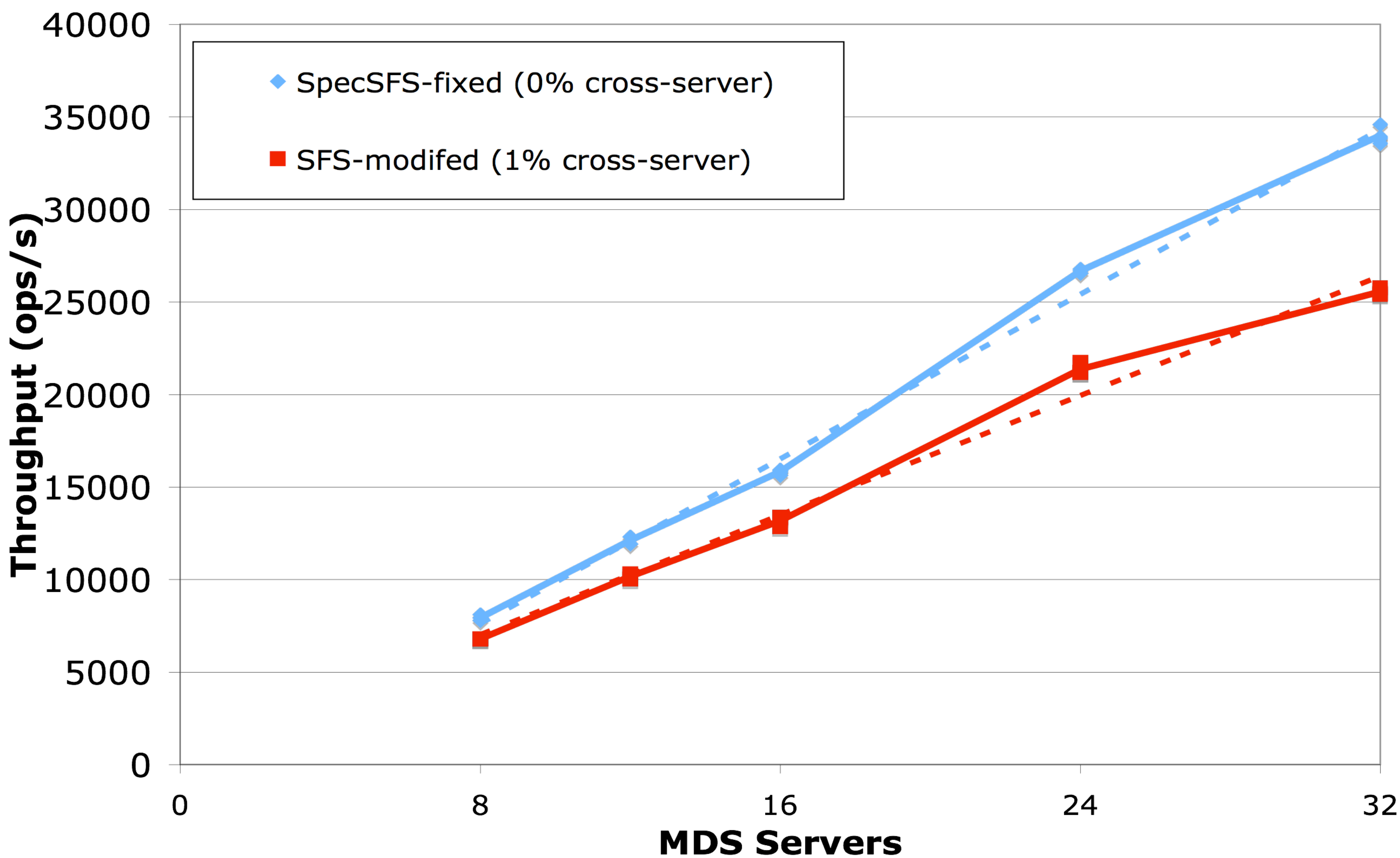
- Simple directory structure
- No multi-directory ops in workload
- OID-assignment policy does perfectly

Adding multi-server ops

Artificially introduce them

- Replace CREATEs with cross-dir LINKs
  - Same work for each operation
- Use “bad” OID-assignment policy
- 1% multi-server ops
- 100X rate from traces!

Scalability with multi-server ops



Carnegie Mellon
Parallel Data Laboratory

Causes of slowdown

- Latency of migration
- Side-effects on other operations
  - Migration makes a table unavailable
  - Servers flush cache on migration
- Granularity of migration is significant
  - The smaller, the better
  - Extreme case is single-object
- *Encountered very rarely in practice*

Implementation

	Lines of C
Base metadata server	47000
Multi-server operations	820
Multi-server using 2PC	2587

- *Half of our implementation is a simple lock manager*
- *Our 2PC implementation is not robust*

Conclusion

- Feasible to reuse migration to support multi-server operations
- Almost no overhead w/ shared storage
  - Harvard, NetApp, SpecSFS97 workloads
  - Even higher multi-server operation rates
- Good choice for system designers
- Transparent scalability made easy