
IndexFS:

Scaling File System Metadata Performance with Stateless Caching and Bulk Insertion

Kai Ren

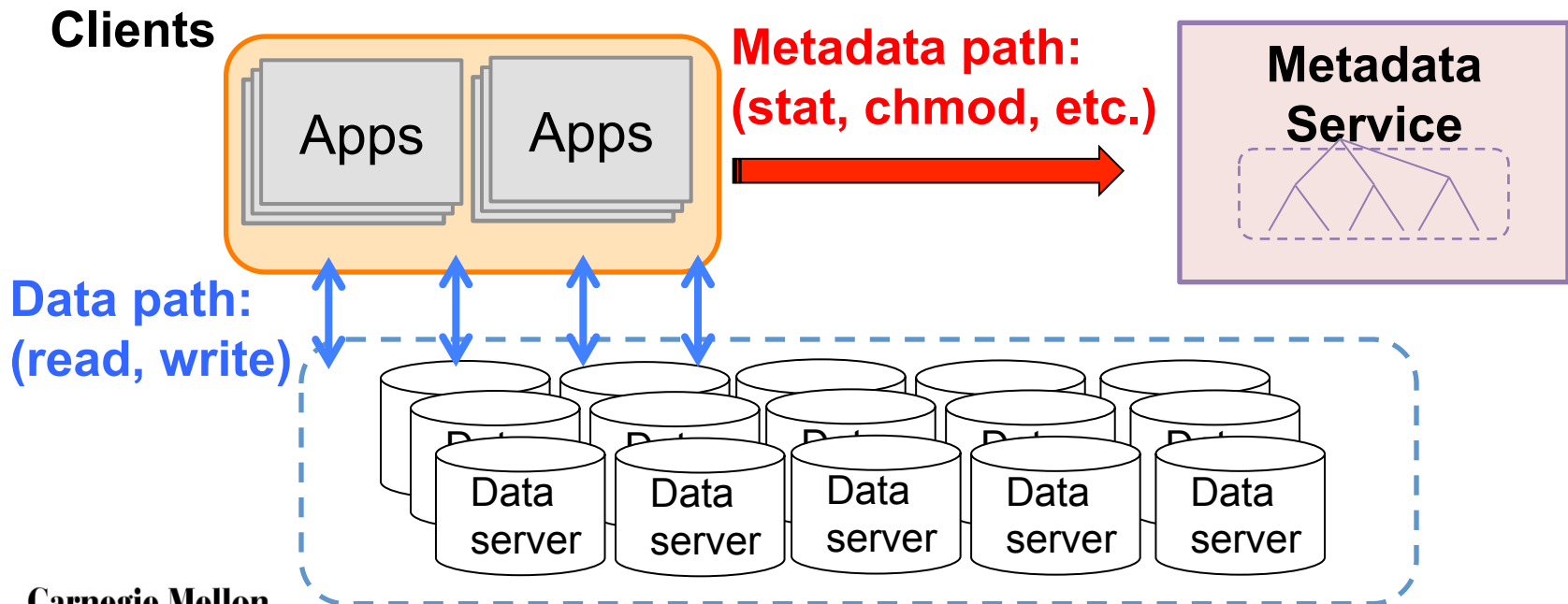
Qing Zheng, Swapnil Patil, Garth Gibson

PARALLEL DATA LABORATORY

Carnegie Mellon University

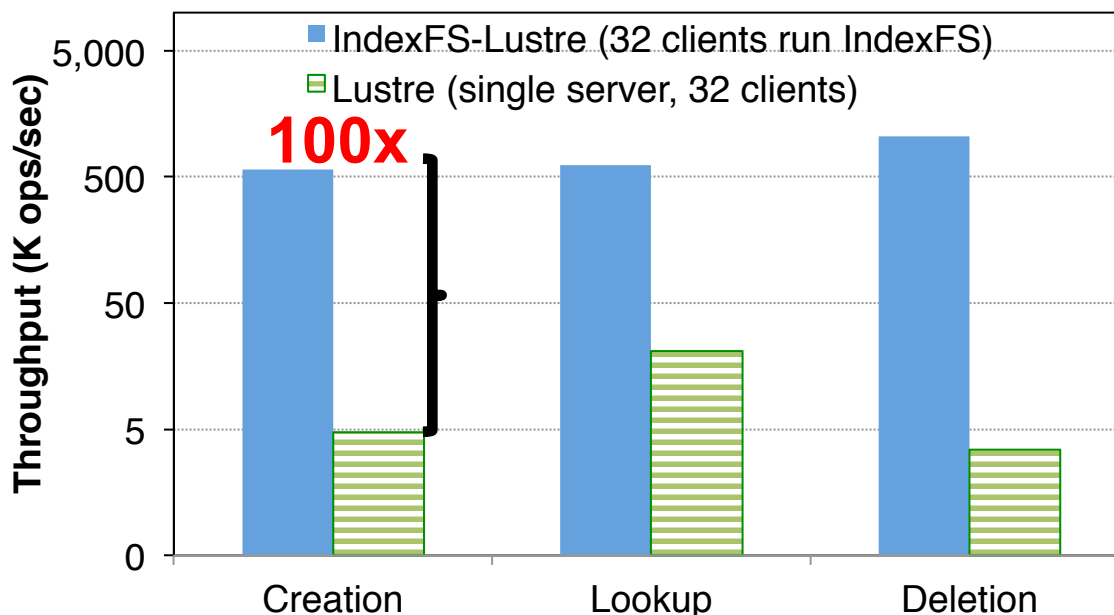
Why Scalable Metadata Service

- Many cluster file systems don't scale MDS well
 - Single metadata server (e.g. Lustre/HDFS)
 - Flat object name space (e.g. Amazon S3)
 - Static partition namespace (e.g. Fed. HDFS/Lustre 2.4)



Why Scalable Metadata Service

- Need a scalable, automatically load balanced, distributed metadata service
- IndexFS:
 - A middleware solution that provides metadata performance scaling for existing file systems

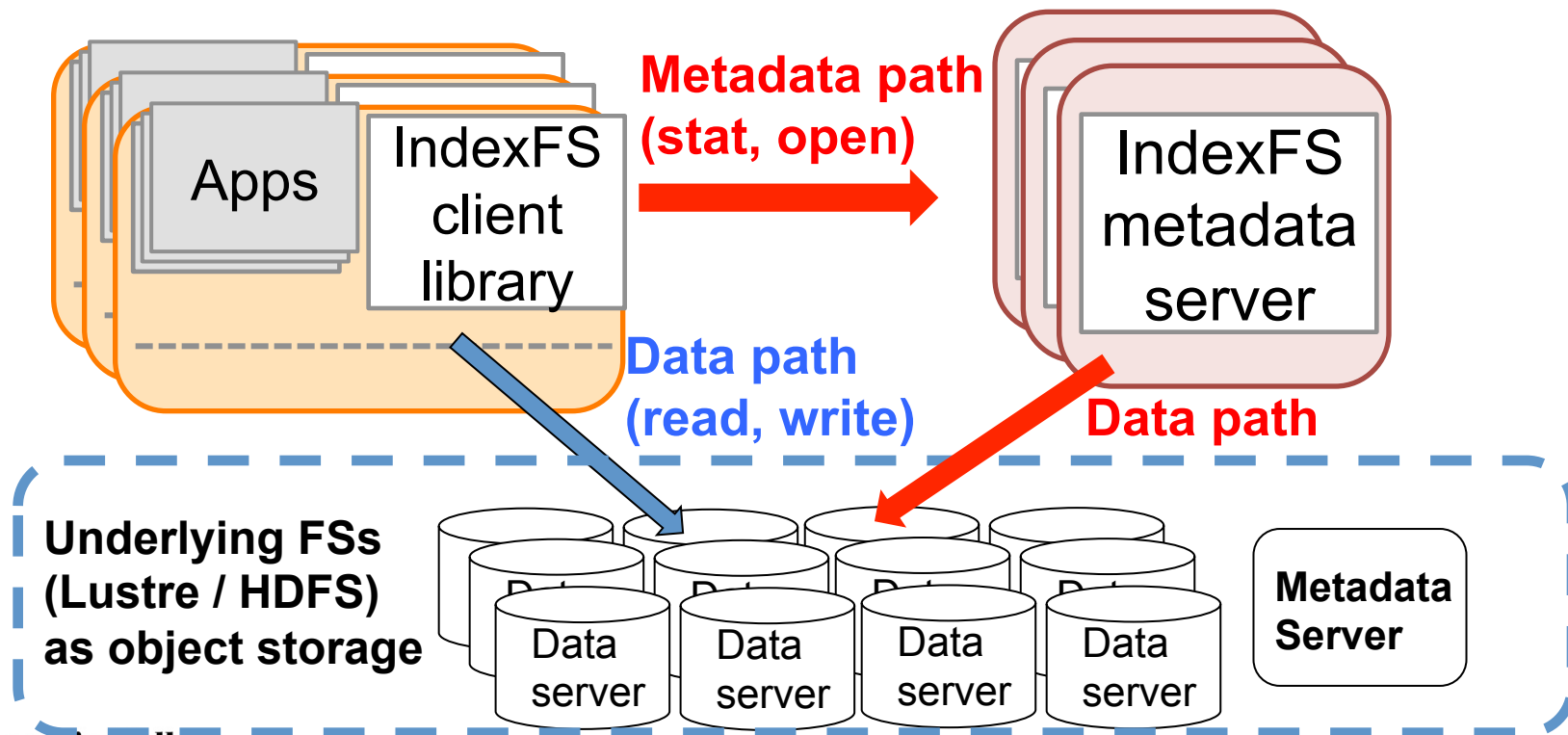


Outline

- Motivation
- ✓ **IndexFS: Distributed Metadata Service**
 - Architecture
 - Namespace Distribution
 - Hotspot Mitigation with Storm-free Caching
 - Column-style Table and Bulk Insertion
- Summary

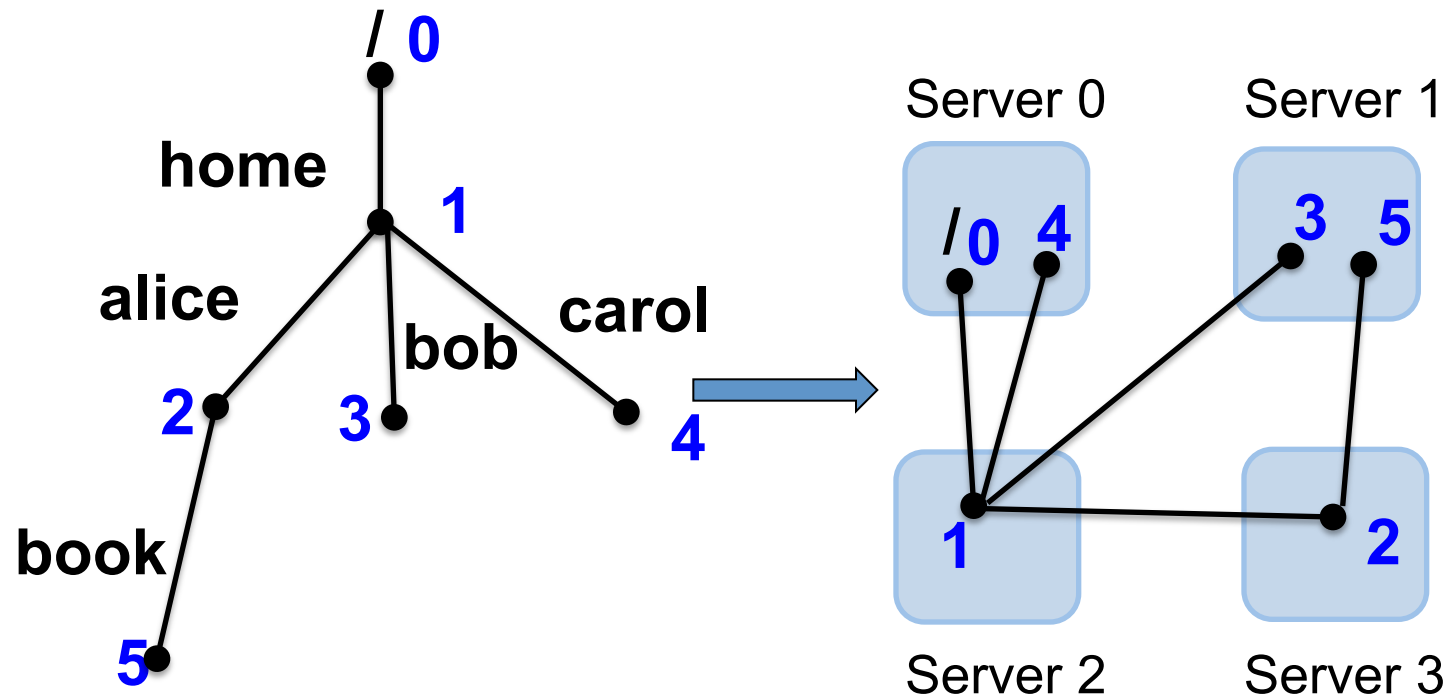
Middleware Design

- IndexFS is layered on top of original DFSs
 - Provide a scalable metadata path for existing file systems such as HDFS, PVFS and Lustre



IndexFS Directory Distribution

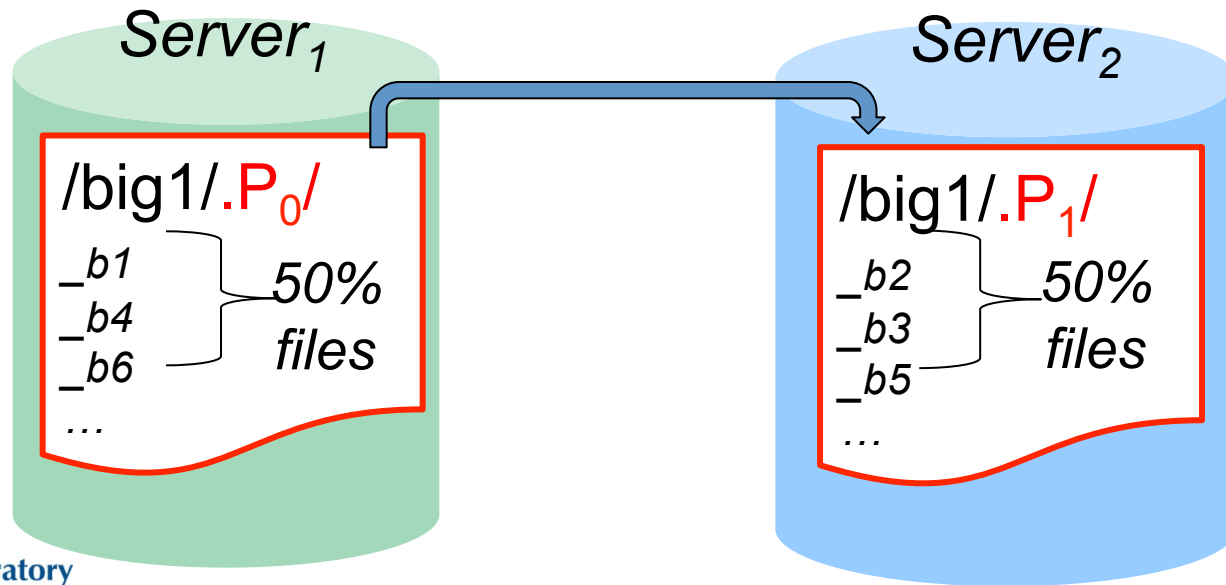
- Randomly assign directory to servers at creation
 - Load balance small directory working set



Dynamically Split Large Directories

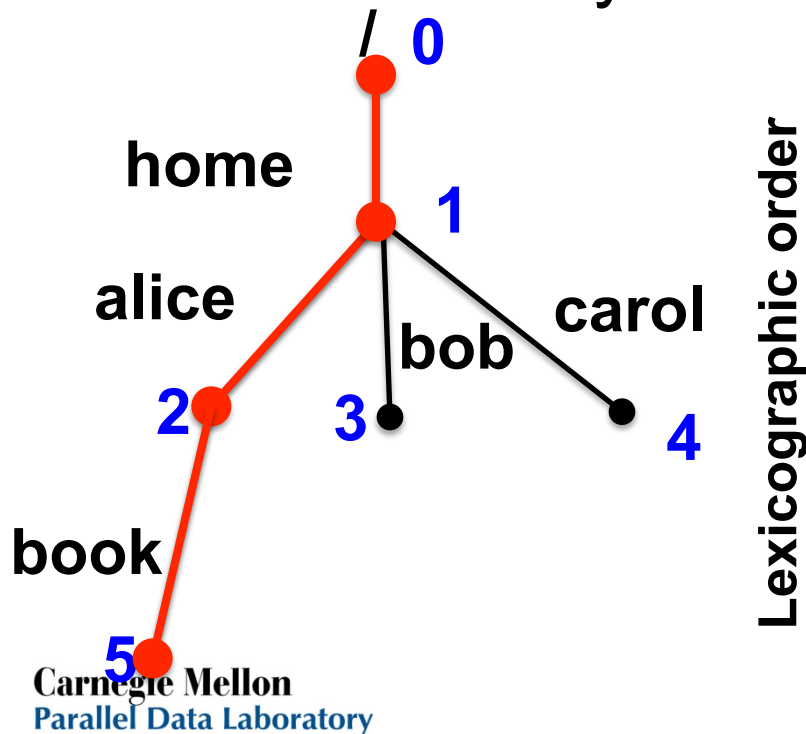
- Use GIGA+ for incremental growth [Patil11]
 - Binary split a partition when its size exceeds a threshold until each server has the same work
 - Load balance giant directories

Binary split a large directory



Efficient Per-Node Metadata Table

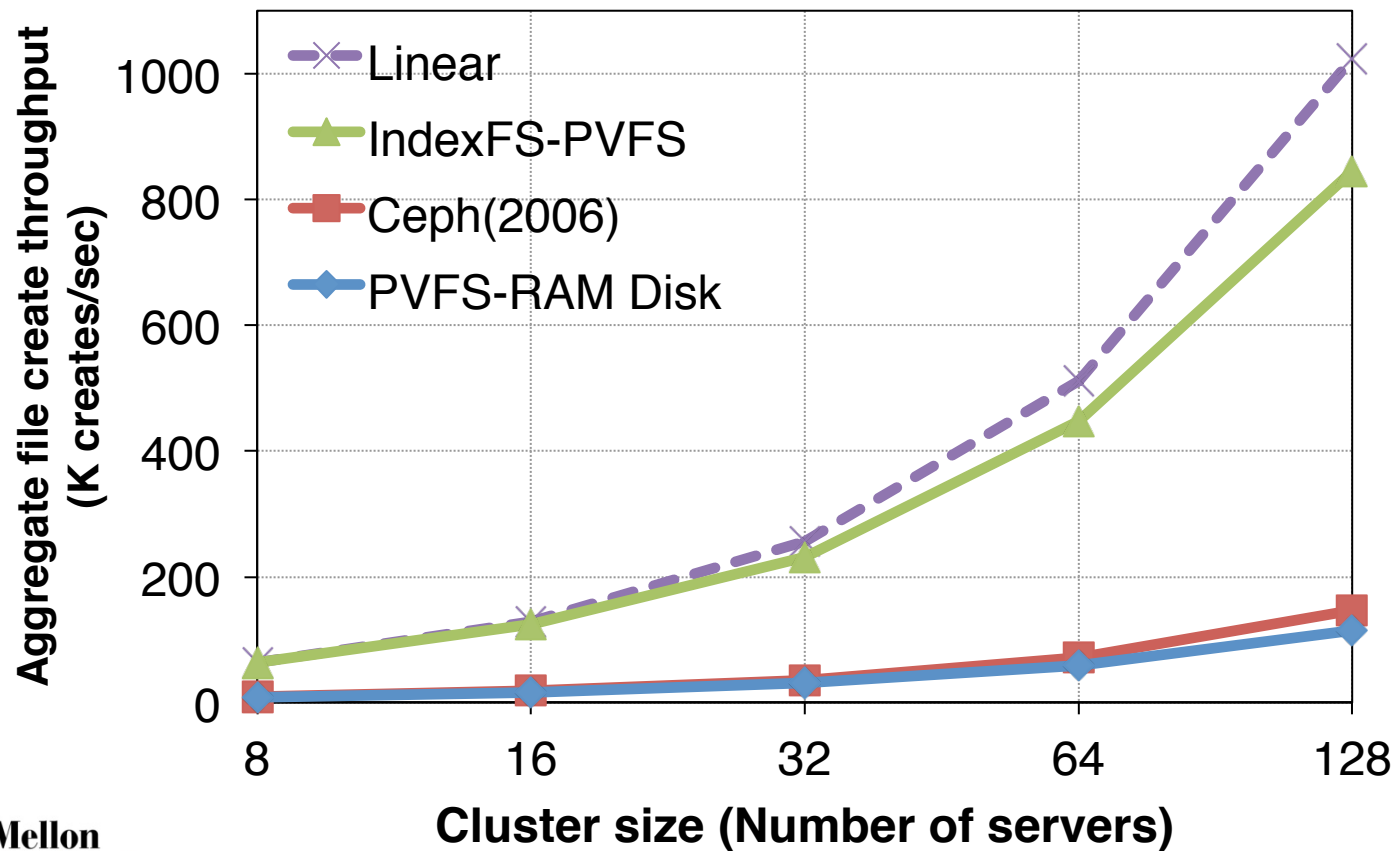
- Represent directory / file as a key-value pair
- Key: <parent directory's inode number, filename>
- Value: attributes, file data or file pointer
- Stored in a key value store : LevelDB [Dean11]



Lexicographic order	Key	Value
	<0, <i>home</i> >	1, attributes
	<1, <i>alice</i> >	2, attributes
	<1, <i>bob</i> >	3, attributes
	<1, <i>carol</i> >	4, attributes
	<2, <i>book</i> >	5, attributes, small file data

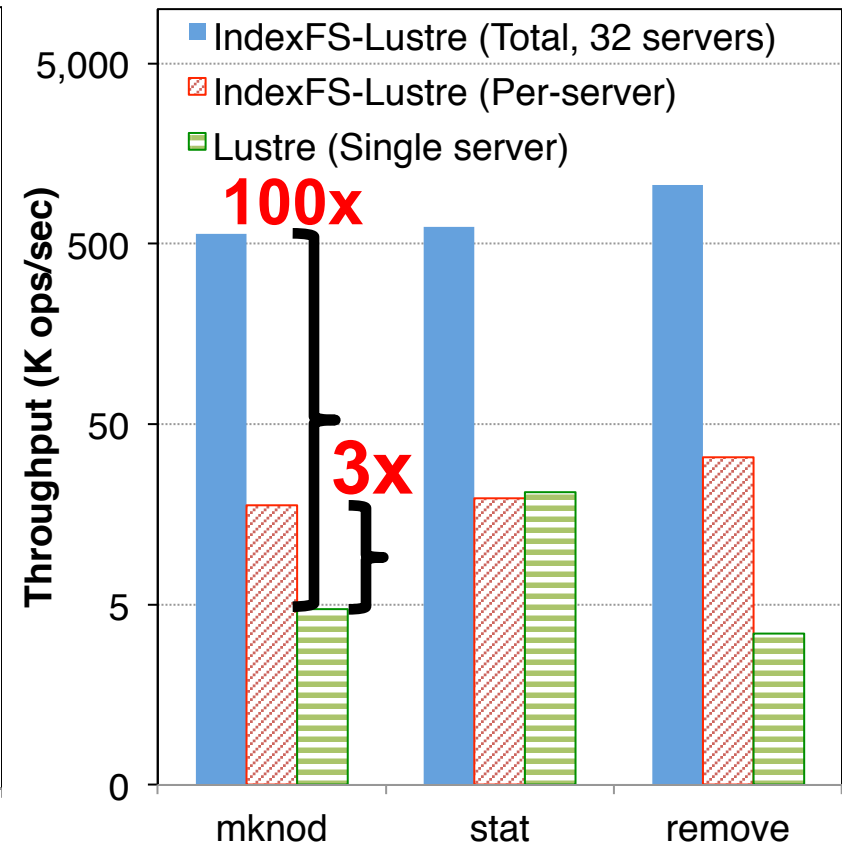
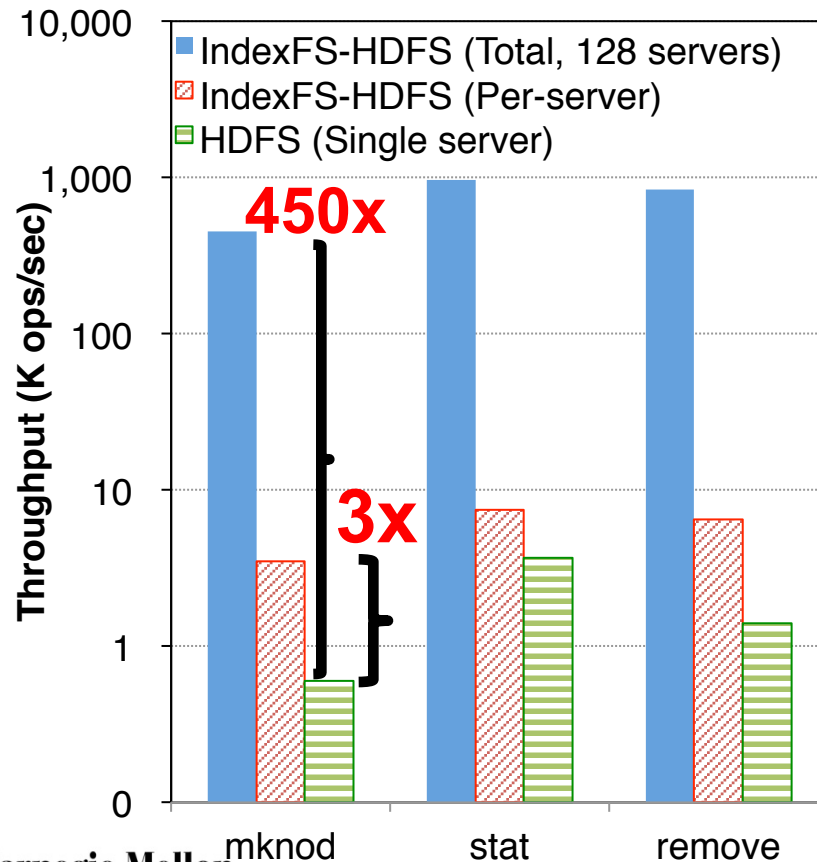
Basic MDS Scalability: File Creation

- Workloads: clients create files in one directory
- Use NFS PRObE Kodiak (8-yr old LANL hardware)



100-450X Faster For HDFS & Lustre

- Run mdtest on IndexFS layered on top of HDFS in PRObE Kodiak and Lustre in LANL Smog clusters

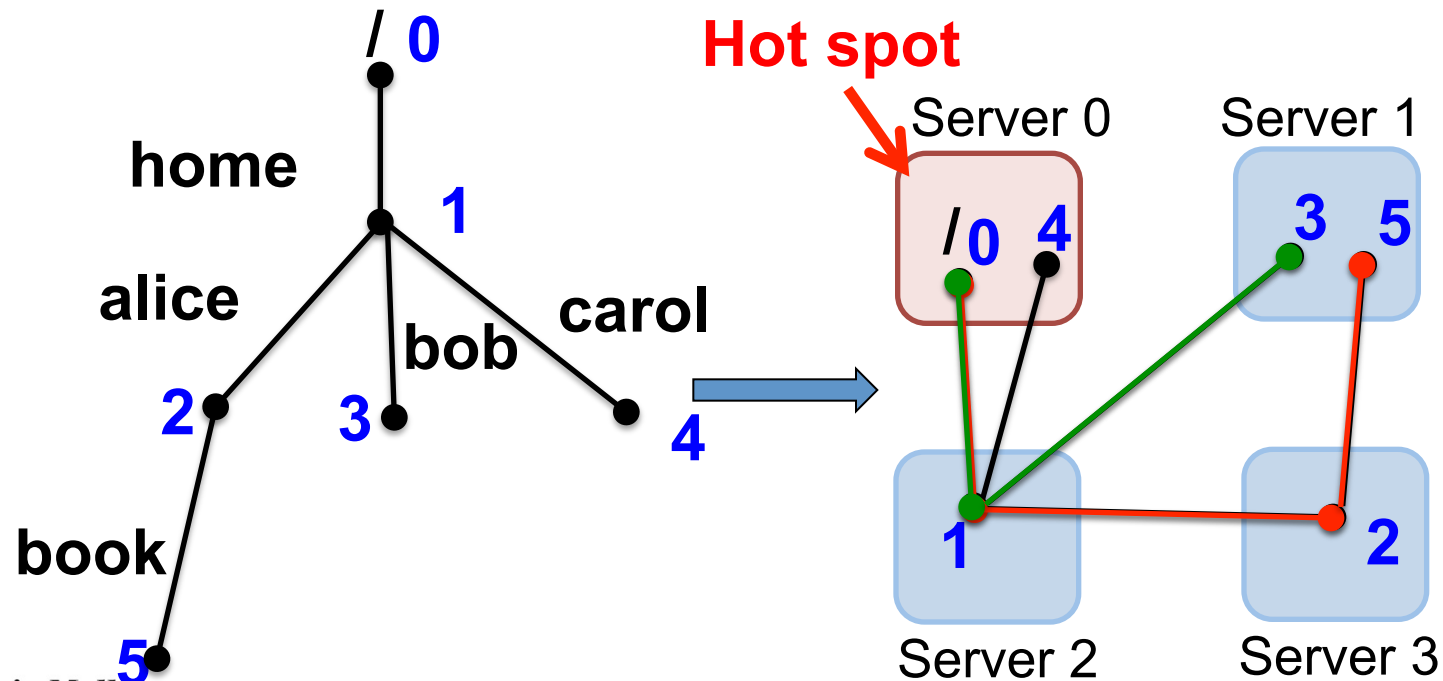


Outline

- Motivation
- ✓ IndexFS: Distributed Metadata Service
 - Architecture
 - Namespace Distribution
 - ✓ Hotspot Mitigation with Storm-free Caching
 - Column-style Table and Bulk Insertion
- Summary

Hot Spot: Server with Root Directories

- Every lookup starts with the root directory
 - Path traversal needs to visit each ancestor
 - Retrieve inode number and access permissions
 - Early IndexFS was defeated by this hot spot



How To Mitigate Hot Spots?

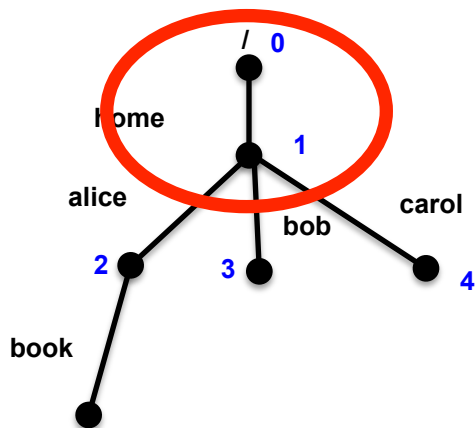
- Client caches directory entry in many FSs
- Problem with invalidation callbacks at scale
 - Cache invalidation waits for responses of all clients
 - Clients may fail / may cause invalidation storm
 - Many parallel file systems disable cache when busy
- Idea: let the “clock” do the invalidation

How To Mitigate Hot Spots?

- Client caches directory entry in many FSs
- Problem with invalidation callbacks at scale
- Idea: let the “clock” do the invalidation
 - Clients use ***read-only*** cache for directory entries
 - Servers remember only ***expiration deadline***
 - Assume ***clock synchronization*** within data center
 - Directory mutations wait for expiration
 - *rmdir, rename, chmod* can be slow

Timeout Based Lease

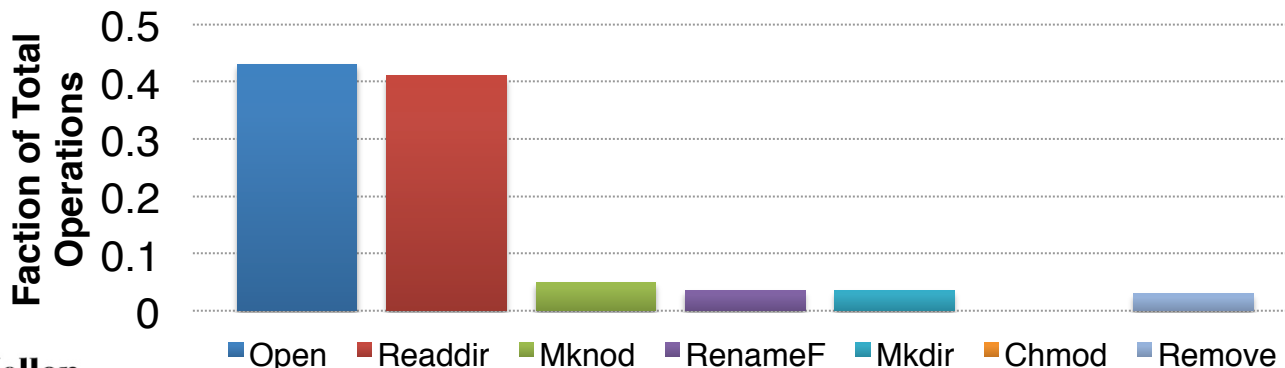
- Inspiration:
 - Hot spots at top of tree has few changes



- Explored two expiration time choices:
 - tree depth: fixed duration K / depth
 - rate based: $L * \text{read rate} / (\text{read rate} + \text{write rate})$

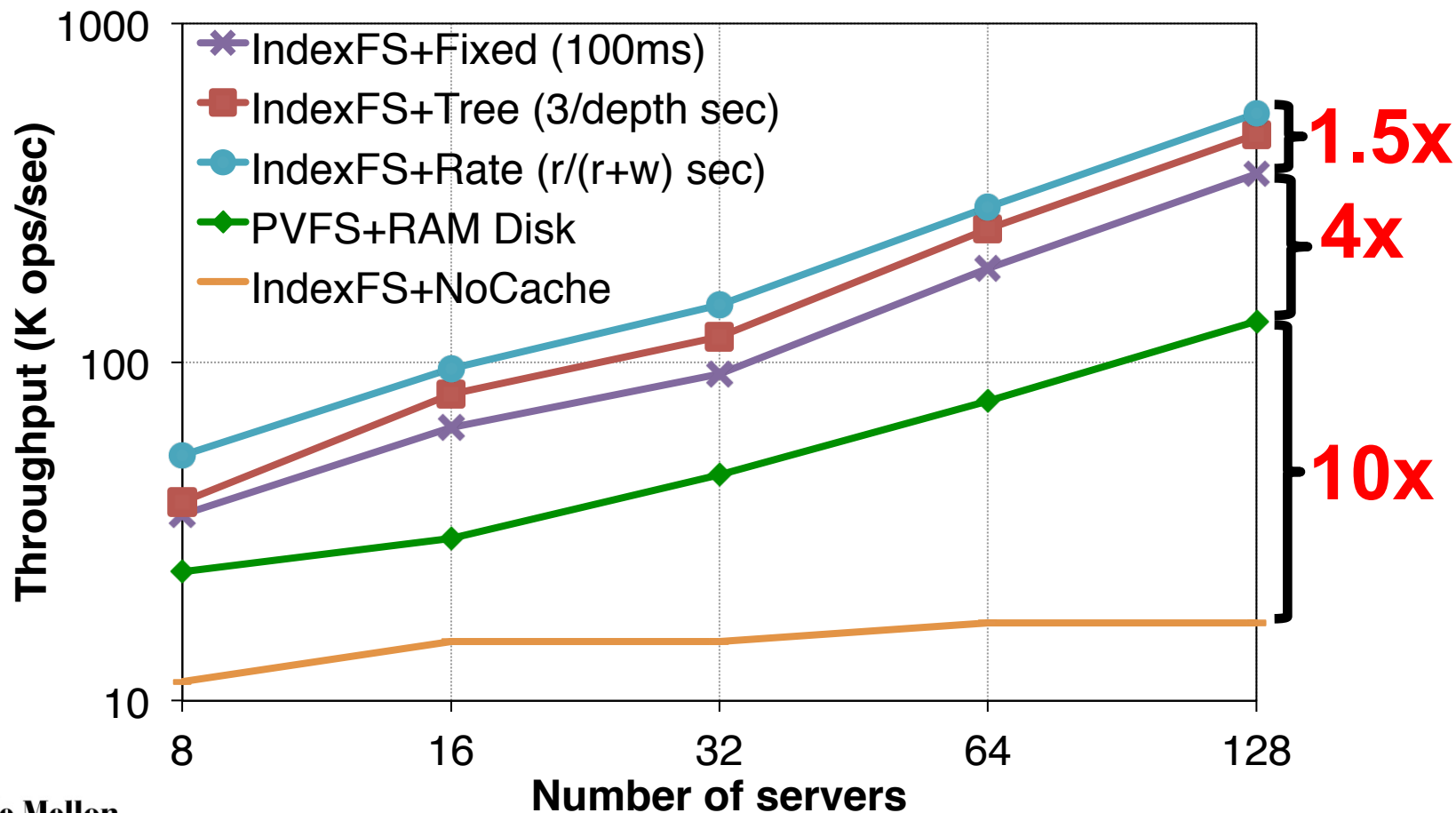
Trace Replay Experiment

- IndexFS run on top of a 128-node PVFS cluster
 - All metadata and data stored in PVFS as files
- Workload:
 - Replay 1 million ops/server from LinkedIn trace
 - Pre-create namespace in the empty file system
 - One day trace: 10M objects and 130M operations
 - Distribution: 90% reads, 10% mutations



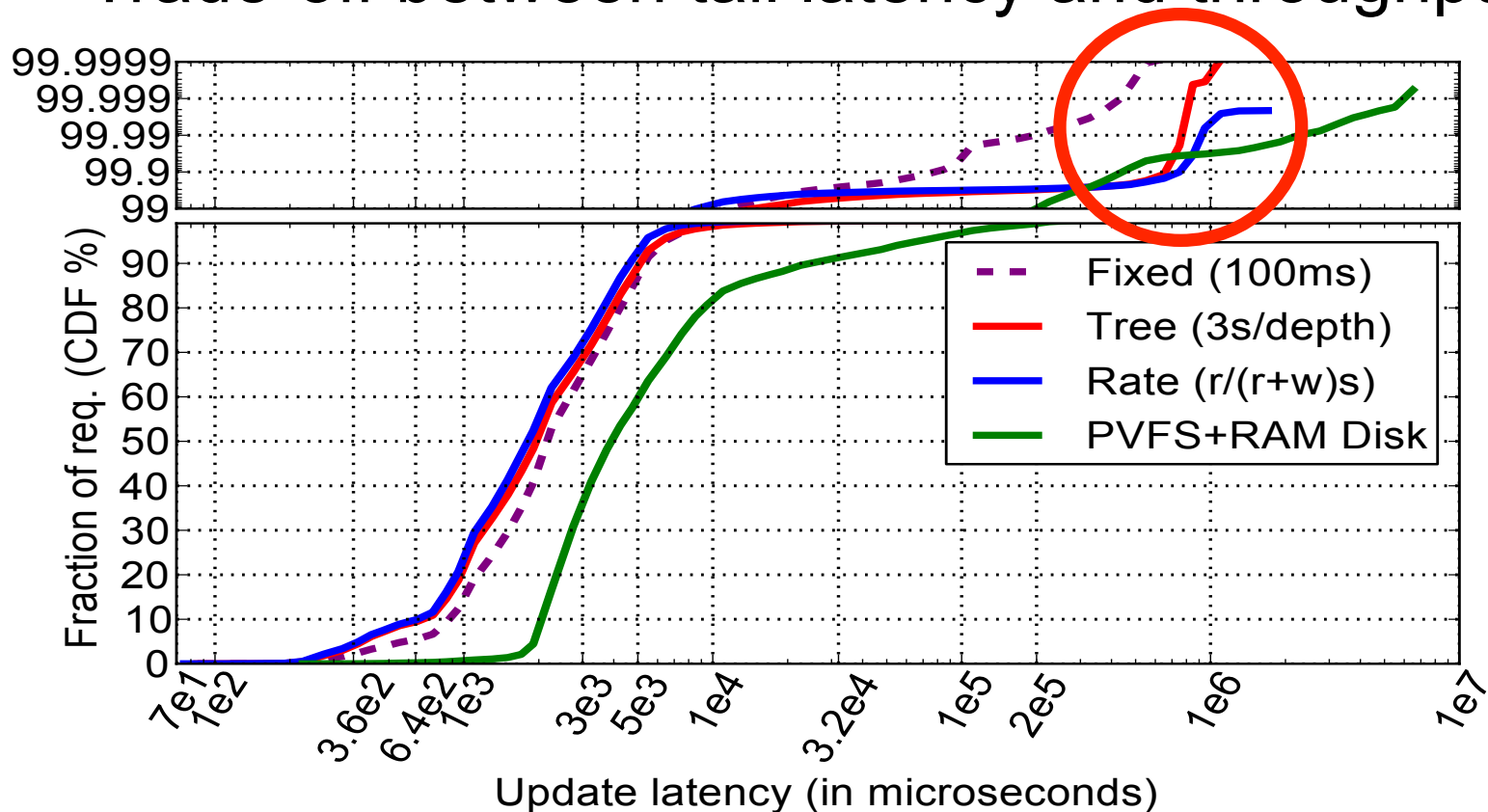
Replay Result: Throughput

- Directory entry cache mitigates hot spots



Replay Result: Chmod Latency

- Fixed duration lease gets the lowest tail latency
 - Trade-off between tail latency and throughput



Outline

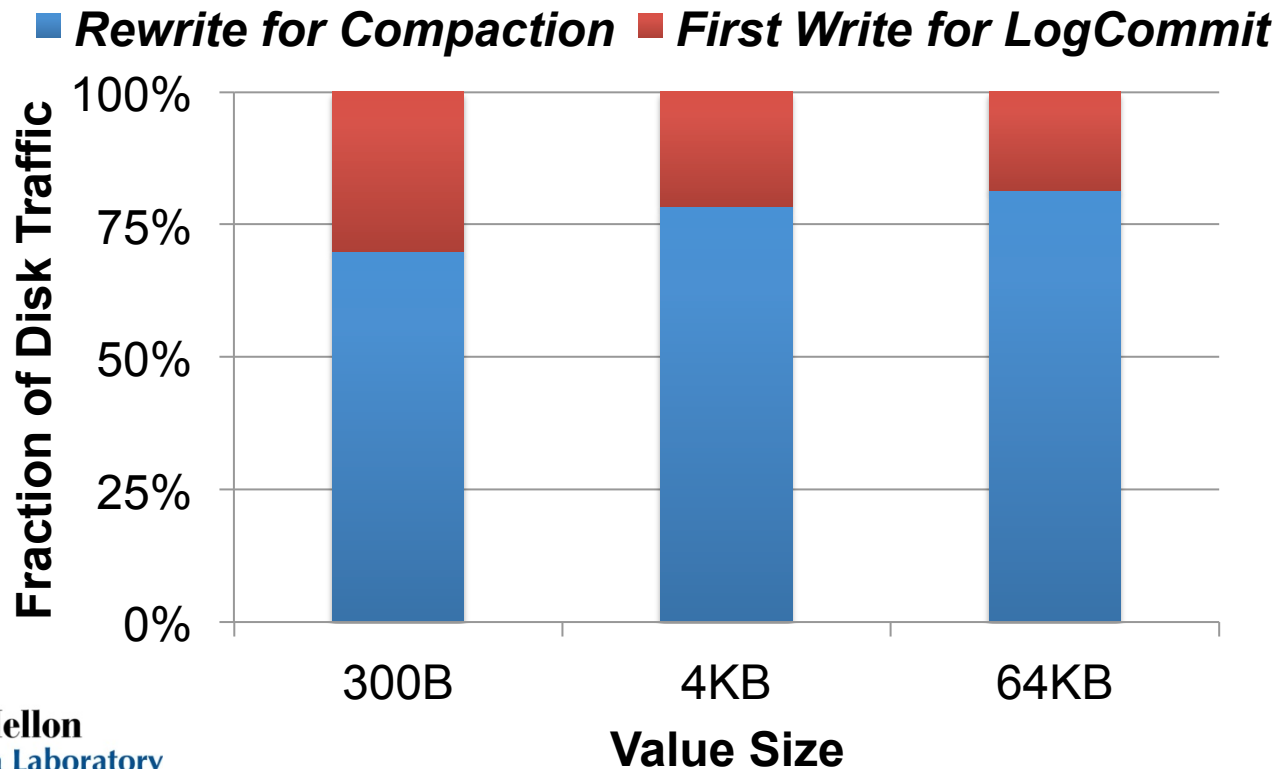
- Motivation
- ✓ IndexFS: Distributed Metadata Service
 - Architecture
 - Namespace Distribution
 - Hotspot Mitigation with Storm-free Caching
 - ✓ Column-style Table and Bulk Insertion
- Summary

Bulk File Creations

- Some applications want faster file creations
 - e.g. HPC checkpoint applications
- IndexFS removes two main bottlenecks:
 - LevelDB's background compaction
 - **Column-style storage schema**
 - Too many RPC round trips
 - **Bulk insertion**

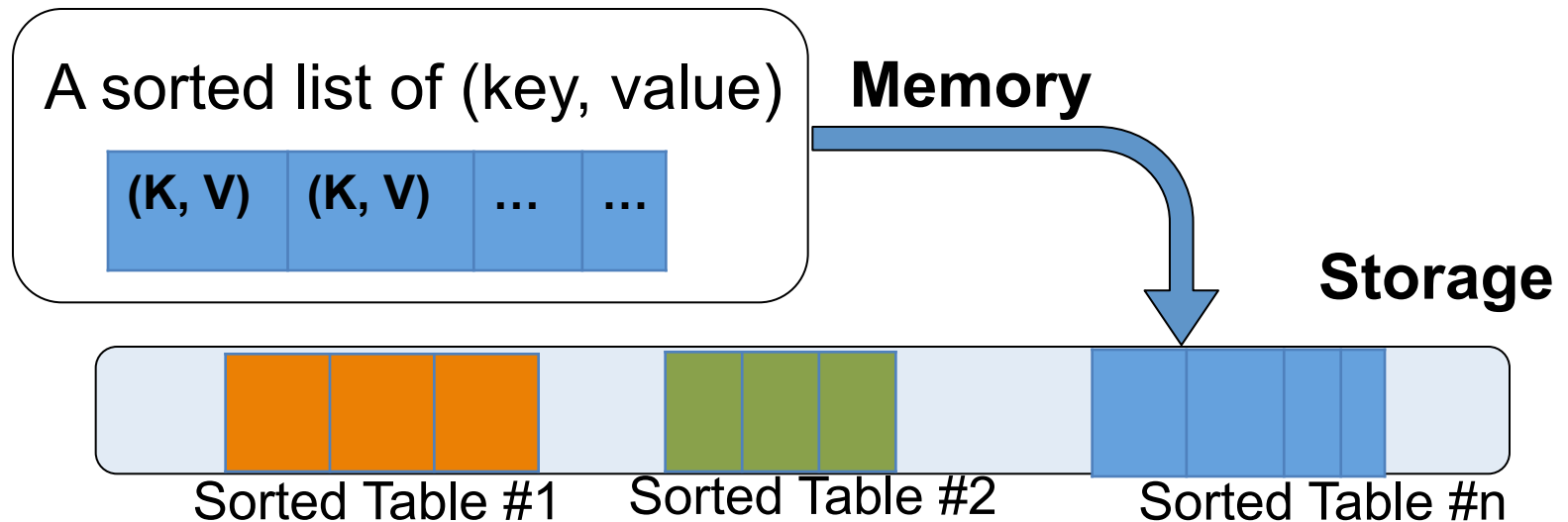
1st Bottleneck: Compaction in LevelDB

- Slow insertion for storing all metadata and data file
 - 75% of insertion disk traffic caused by compaction
 - Compaction of larger values wastes more bandwidth



Background: LevelDB Internal

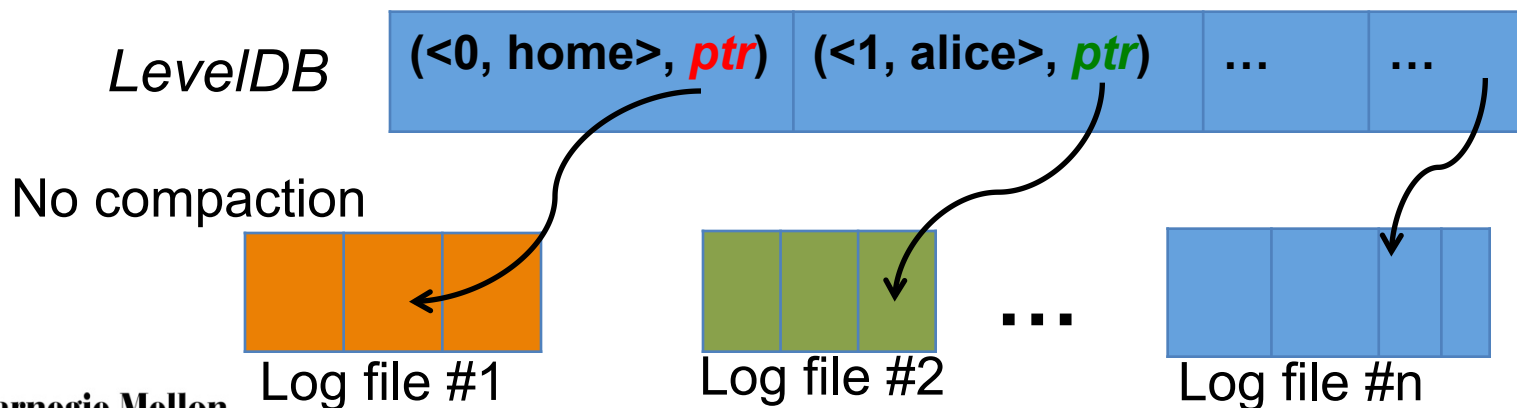
- LevelDB (LSM Tree): Insert and update ops:
 - Buffer and sort recent inserts/updates in memory
 - Flush buffer to generate immutable sorted tables
 - Perform compaction to reduce #tables for searching
- **Want to avoid rewrites while having fast lookup**



Column-Style Metadata Schema

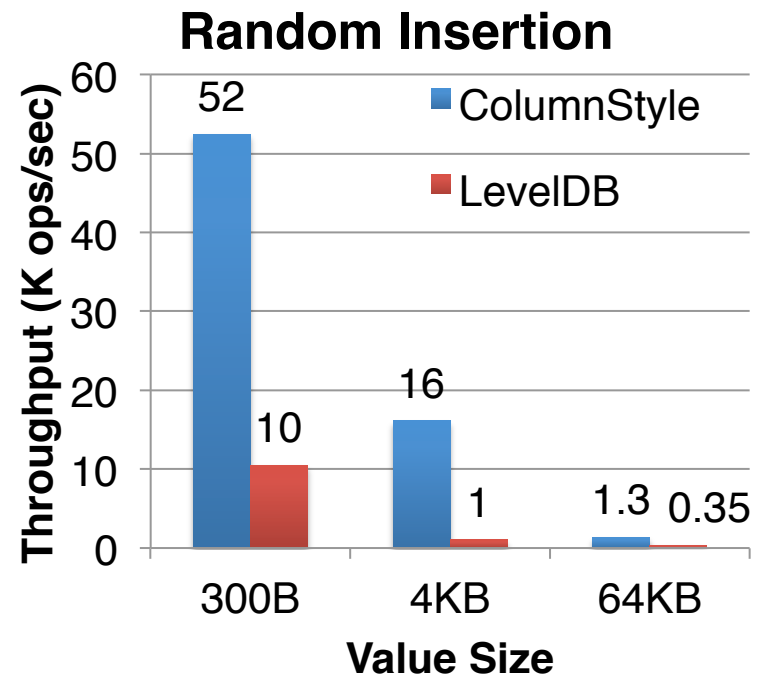
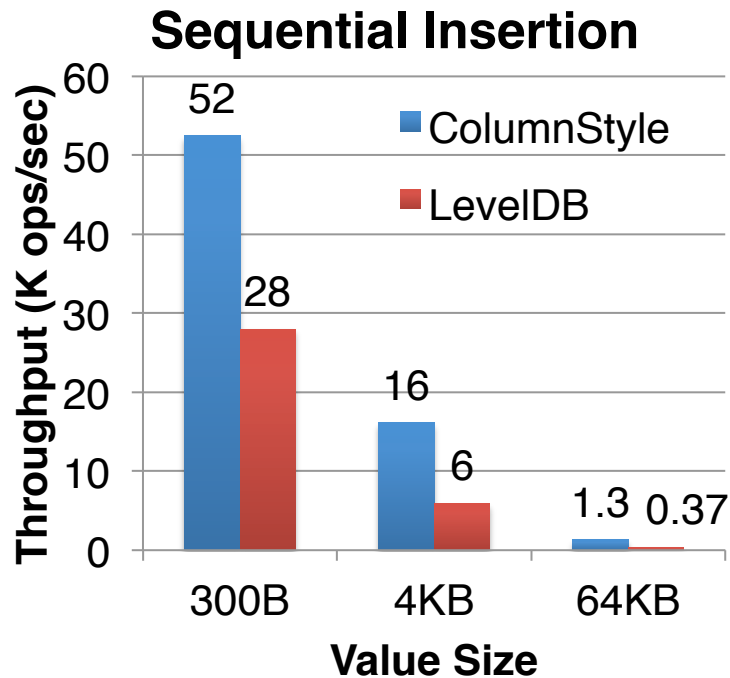
- Column-style approach:
 - Metadata/small files appended to non-LevelDB log files
 - LevelDB stores only pointers to metadata
 - Don't compact the metadata
 - Delay space reclamation for deleted/over-written values since metadata are small

Only compact pointers



Speed Up Ingestion Rate

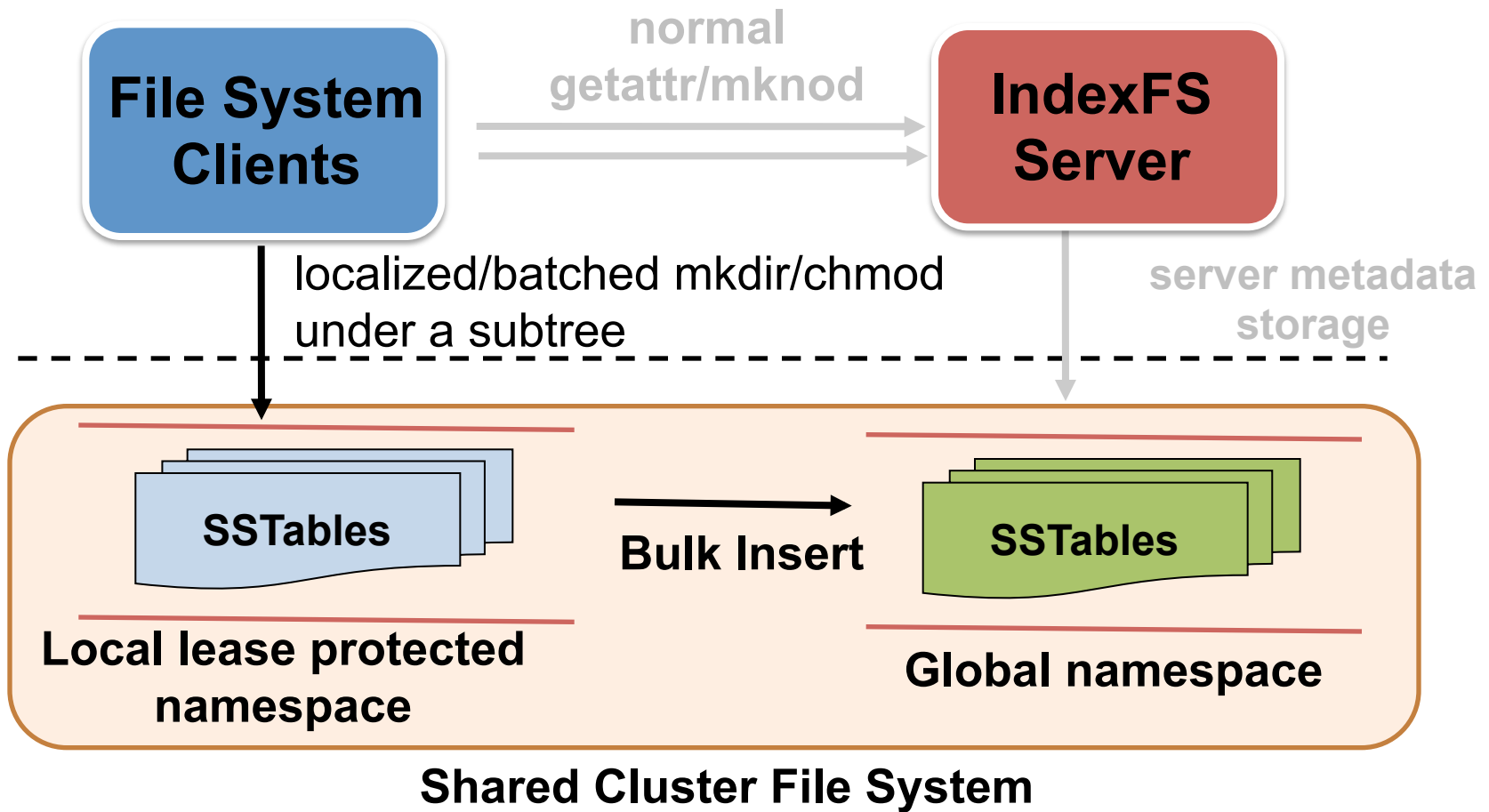
- Insert 30M entries sequentially or randomly
 - Limit memory to 350 MB, using single SATA disk
- About 2 to 15 times faster insertion



Fix 2nd Bottleneck: Bulk Insertion

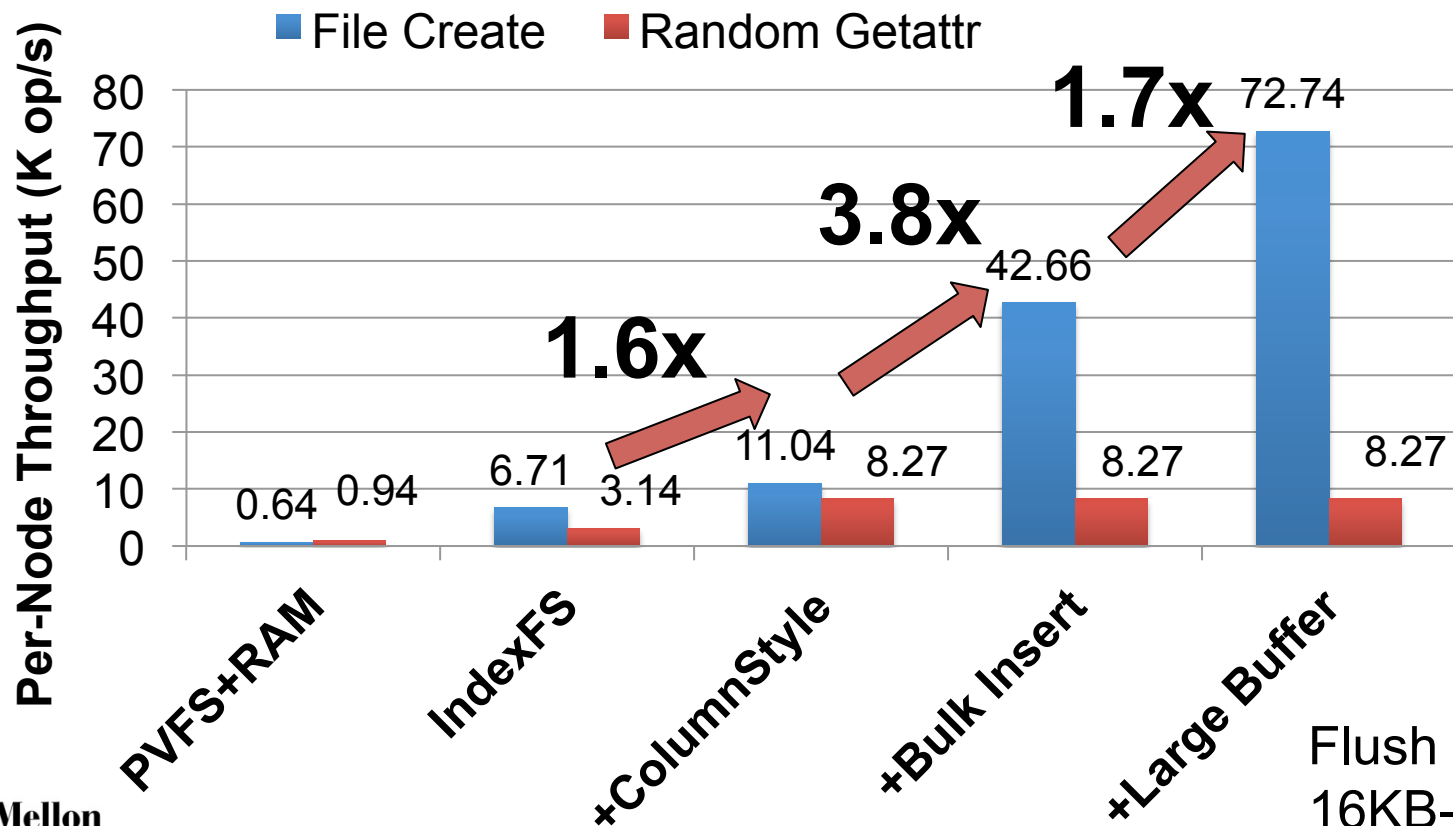
- Extend client cache to support write-back
 - No RPC overheads per metadata operation
 - Avoids constant synchronization at servers
 - Better utilization of the underlying bandwidth
- Assumptions
 - Clients have access to backend storage nodes
 - Only possible for clients to insert new subtrees
 - No namespace conflicts
 - Asynchronous error reporting is acceptable

Bulk Insertion Implementation



Bulk Insertion: Factor Analysis

- Evaluation bulk insertion on top of PVFS
 - Perform random insertion and stat in one directory



Summary

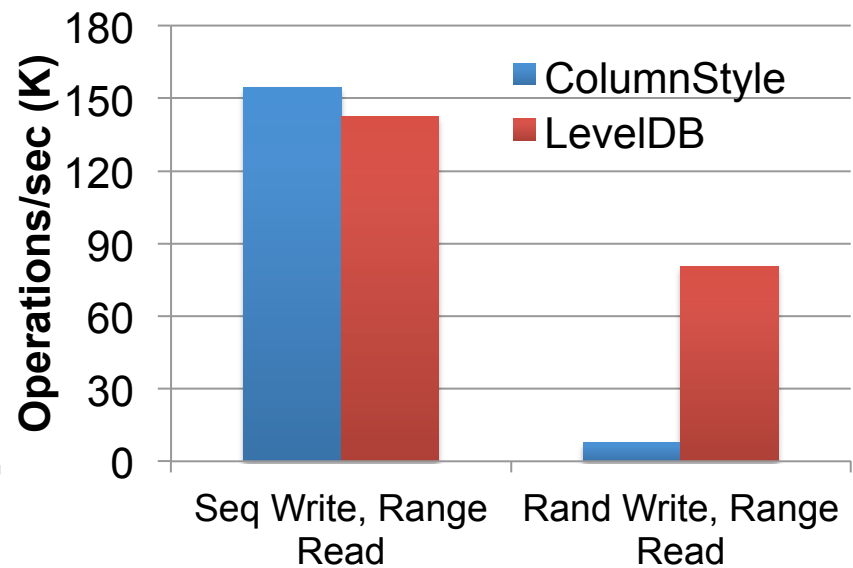
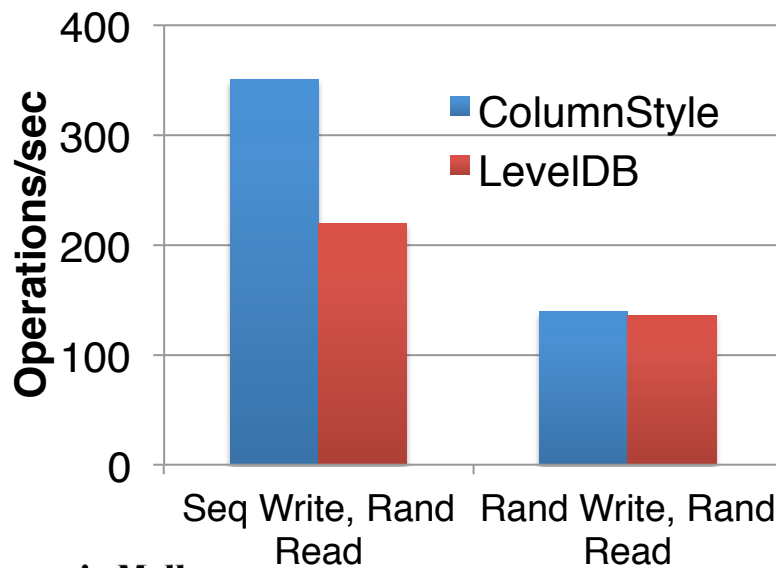
- IndexFS: a **middleware** approach to scale metadata path, portable to HDFS, Lustre, PVFS
- Scale out: **partition tree on directory basis**
 - GIGA+ for incremental partition of giant directories.
 - Read-only consistent caching without per-client state
- Scale up: **optimize log-structured merge tree**
 - Column-style schema reduces compaction overhead
 - Bulk insertion to avoid server coordination
- Code available: <http://www.pdl.cmu.edu/indexfs>

Reference

- [Ren14] *IndexFS: Scaling File System Metadata Performance with Stateless Caching and Bulk Insertion*. Kai Ren, Qing Zheng, Swapnil Patil and Garth Gibson. SC 2014
- [Ren13] *TableFS: Enhancing metadata efficiency in local file systems*. Kai Ren and Garth Gibson. USENIX ATC 2013
- [Welch13] *Optimizing a hybrid ssd/hdd hpc storage system based on file size distributions*. Brent Welch and Geoffrey Noer. *29th IEEE Conference on Massive Data Storage*, 2013.
- [Meister12] *A Study on Data Deduplication in HPC Storage Systems*. Dirk Meister, Jurgen Kaiser, Andre Brinkmann, Toni Cortes, Michael Kuhn, Julian Kunkel. Supercomputing 2012
- [Patil11] *Scale and Concurrency in GIGA+: File System Directories with Millions of Files*. Swapnil Patil and Garth Gibson. FAST 2011
- [Dean11] *LevelDB: A fast, lightweight key-value database library*. Jeff Dean and Sanjay Ghemawat. <http://leveldb.googlecode.com>.
- [Meyer11] *A Study of Practical De-duplication*. Dutch T. Meyer, and William J. Bolosky. FAST 2011
- [Ganger97] *Embedded Inodes and Explicit Grouping: Exploiting Disk Bandwidth for Small Files*. Gregory Ganger and Frans Kaashoek. Usenix ATC 1997.
- [O'Neil96] *The log-structured merge-tree (LSM-tree)*. Patrick O'Neil and et al. *Acta Informatica*, 1996
- [Rosenblum91] *The design and implementation of a log-structured file system*. Mendel Rosenblum and John Ousterhout. SOSP 1991.

Comparable Read Throughput

- Read after rand./seq. insertion of 4KB entries
 - *Random Read*: read randomly over all entries
 - *Range Read*: read randomly in 1% adjacent range
- No compaction, so no clustering for range read
 - Only use compaction for workload-specific prefetch



Cluster Specification

- PRObE Kodiak
 - CPU: AMD Opteron 252, Dual core, 2.6GHz
 - Memory: 8GB
 - Network: 1GE NIC
 - Storage: Western Digital 1TB hard drive
 - OS: Ubuntu 12.10 Kernel 3.6.6 x86-64
- LANL Smog
 - CPU: AMD Opteron 6136, 16-core, 2.4GHz
 - Memory: 32GB
 - Network: Torus 3D, bandwidth: 4.7GB/s
 - Storage: Hardware RAID array, bandwidth 8GB/s
 - OS: Cray Linux
 - Lustre version: 1.8.6