A case for scaling HPC metadata performance through de-specialization

Kai Ren

Swapnil Patil, Garth Gibson

PARALLEL DATA LABORATORY

Carnegie Mellon University

Why Scalable Metadata Service?

- Applications use directories as a light-weight database (e.g. check pointing)
- Concurrent execution of data-intensive apps
  - By 2020, Exascale-era clusters expected to have up to one billion cores [DARPA08]
- Need to scale metadata service for cluster FS
  - GIGA+: directory partitioning [PDSW07, FAST11]
  - SkyeFS: layered GIGA+ on top of PVFS [Chivetta12]
    - Exploiting PVFS atomic rename across servers

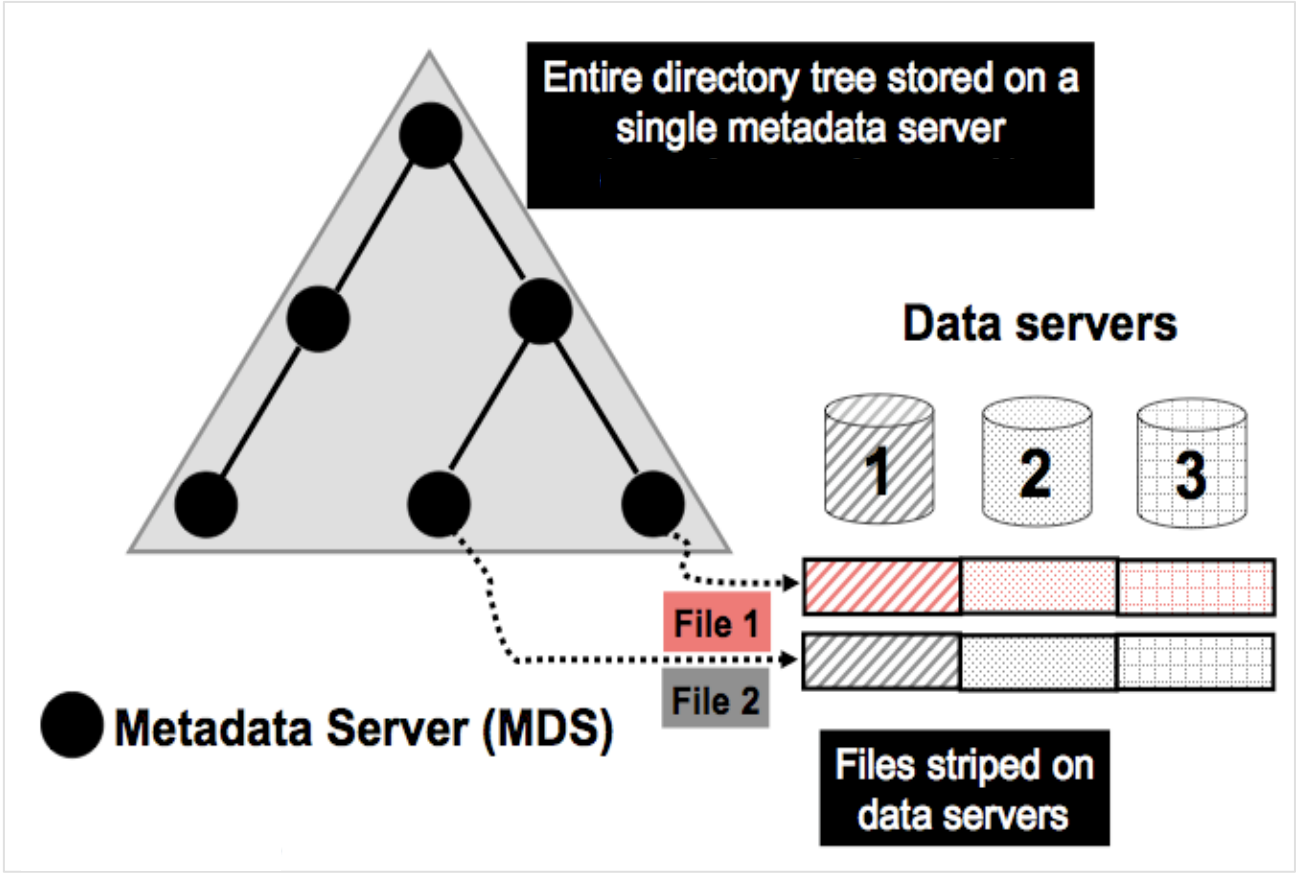
Scaling Metadata in Existing CFSs

- Goal: want a layered (de-specialized) solution
  - Add scalable metadata to many cluster file systems
    - Cluster FS with single metadata server (Lustre)
    - Federated Cluster File System (PanFS, HDFS v2)

- Solution:
  - Apply GIGA+ without dependence on rename
  - Use a new directory representation to decouple metadata distribution from data distribution

Directories in Cluster File Systems

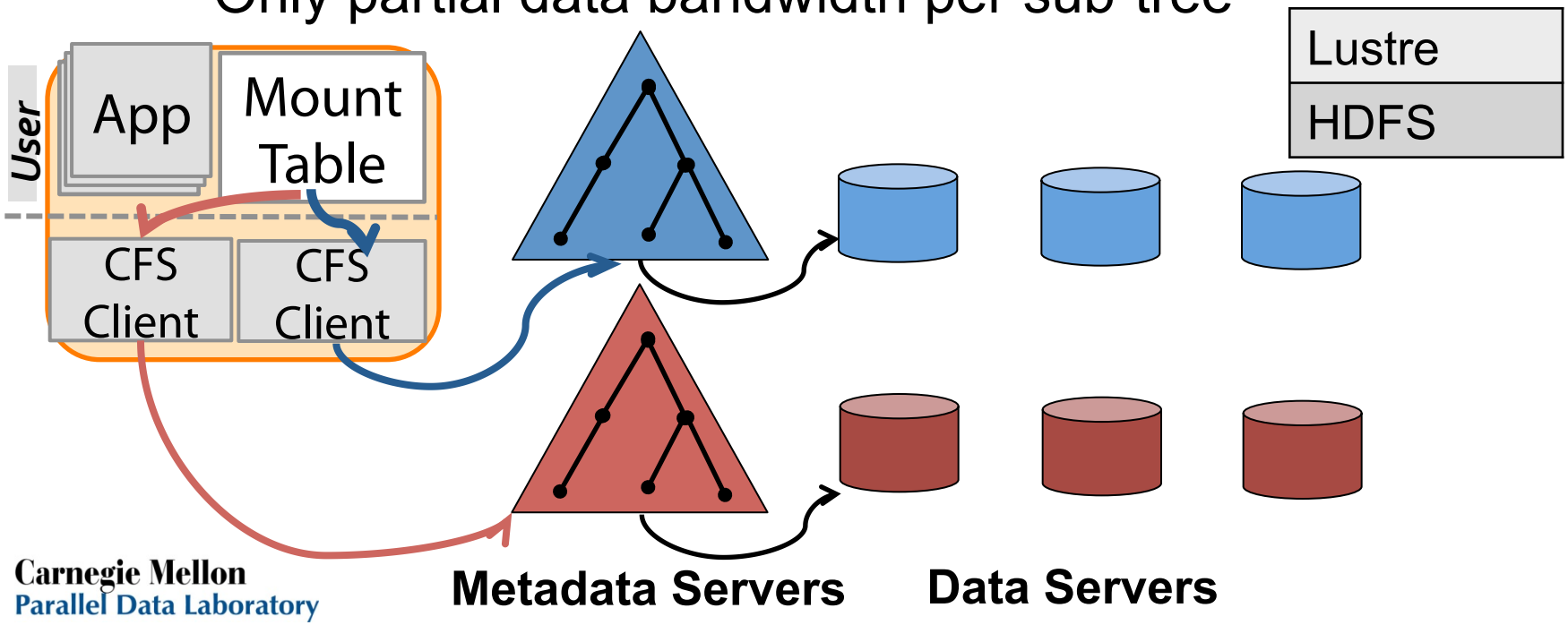
- Entire namespace in a single metadata server



GoogleFS
Lustre
HDFS

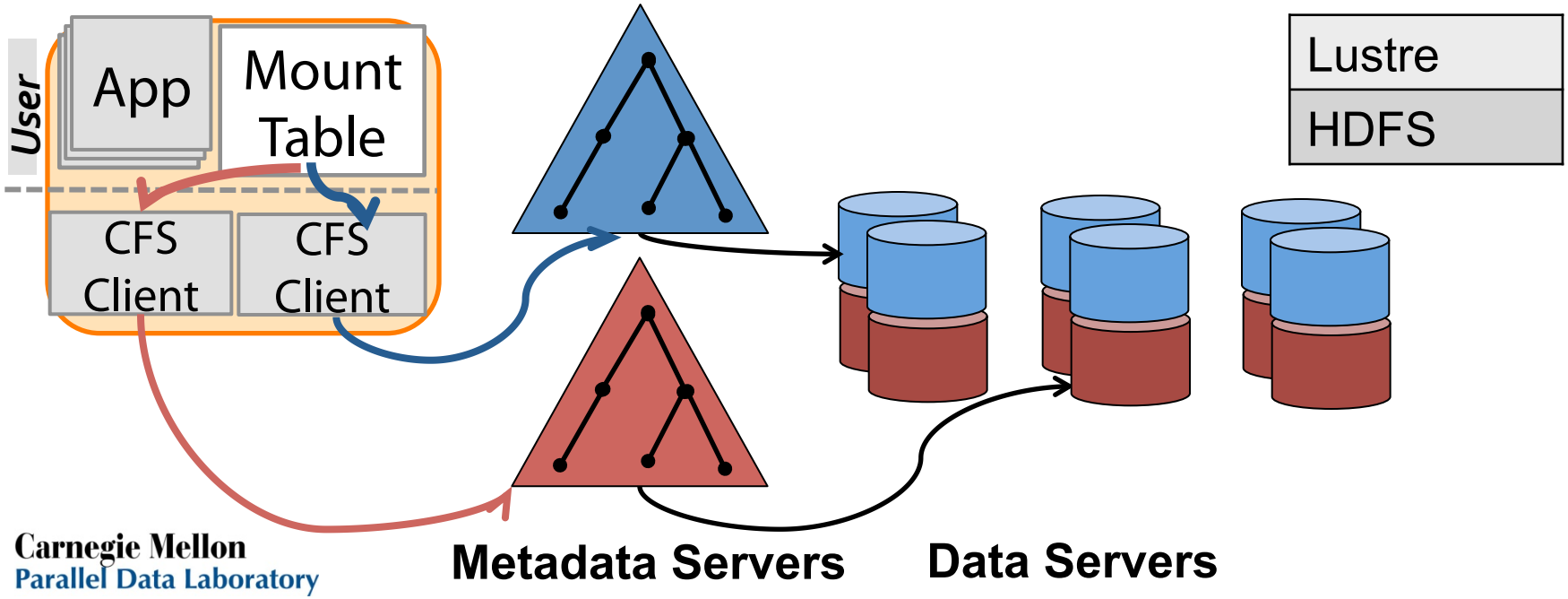
Cluster FS with Multi-Mount Points

- Mount multiple original metadata servers
  - Partition namespace and data servers
  - Client-side mount table for sub-tree access
  - No namespace balance (MDS CPU or capacity)
  - Only partial data bandwidth per sub-tree



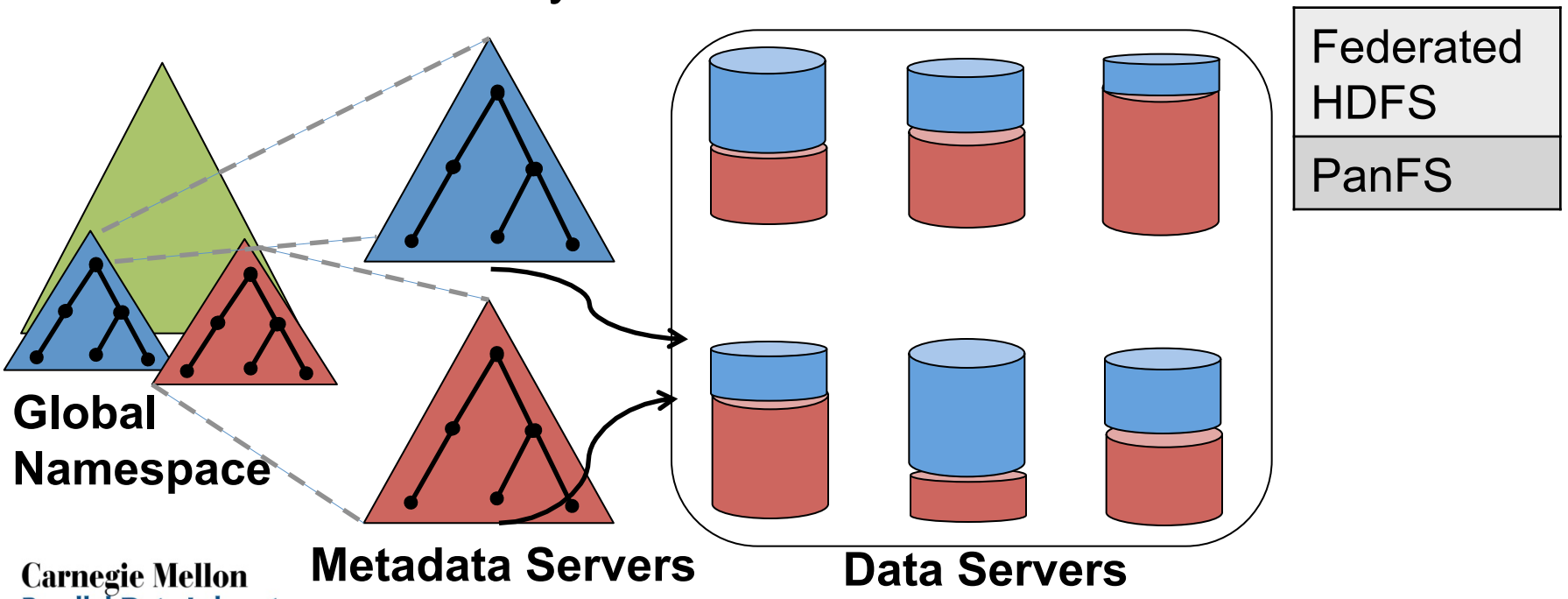
Cluster FS with Multi-Mount Points

- Mount multiple original metadata servers
  - Partition namespace into sub-trees
  - Client-side mount table for sub-tree access
  - No namespace balance (MDS CPU or capacity)
  - Statically partition each data node for full BW



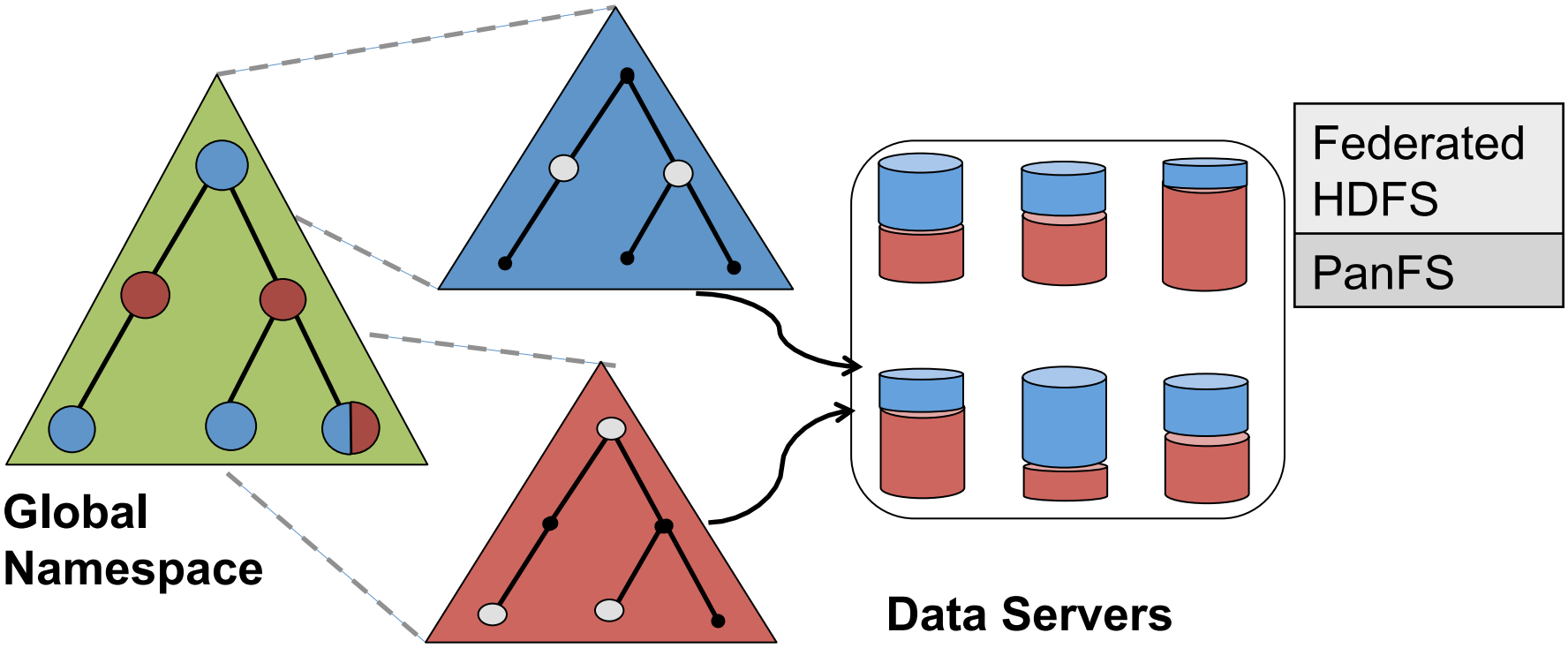
Federated Cluster File Systems

- Dynamically share all data servers
  - No capacity limitation for each sub-tree
- Statically partition namespace to sub-trees
  - Size of each sub-tree not auto-balanced
  - Each directory limited to one server



Scaling Metadata of Federated CFS

- Apply GIGA+ without dependence on rename
  - Balance the size of each sub-tree
  - Large directories are spread across servers

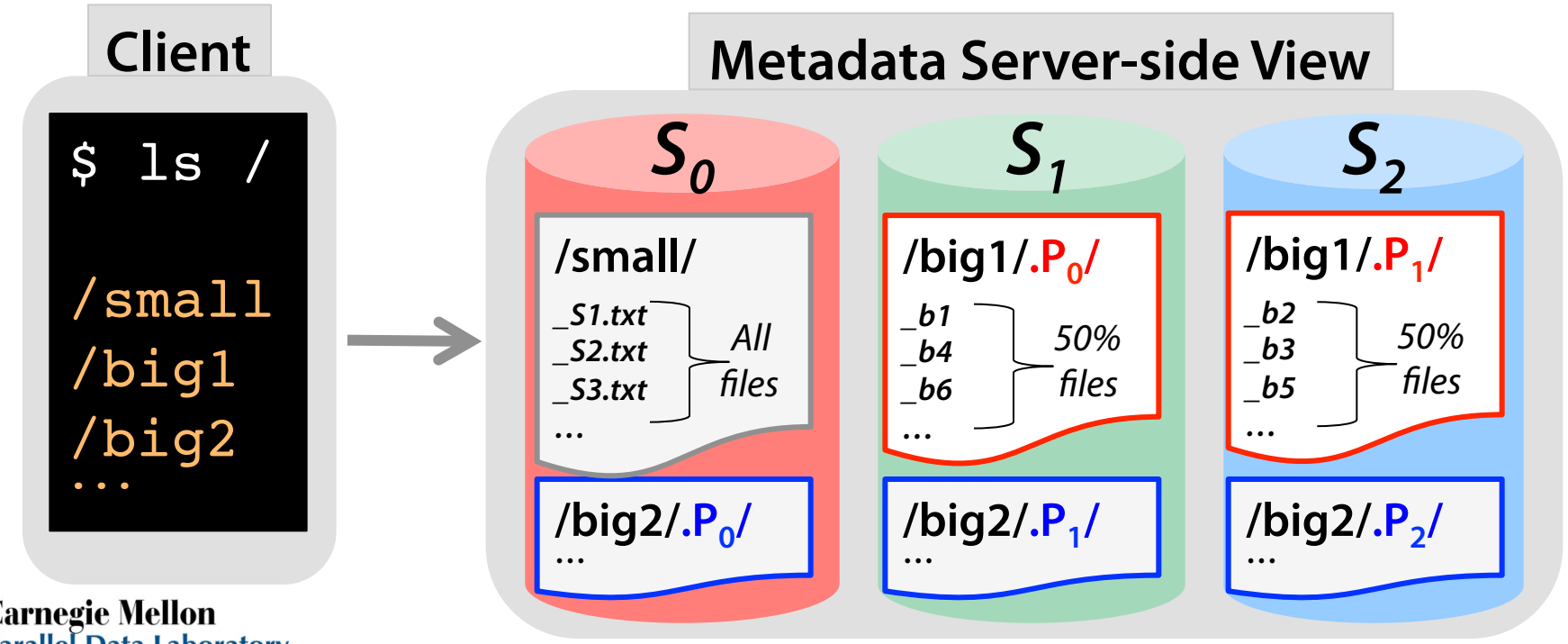


Outline

- Overview
- ✓ System Design
  - GIGA+: Distributed Index Technique
  - System Architecture
- Directory Representation
  - Design & Single Node Performance
- Integrating GIGA+ with TableFS
  - Directory Partition Migration
  - Preliminary Scalability Result
- Summary

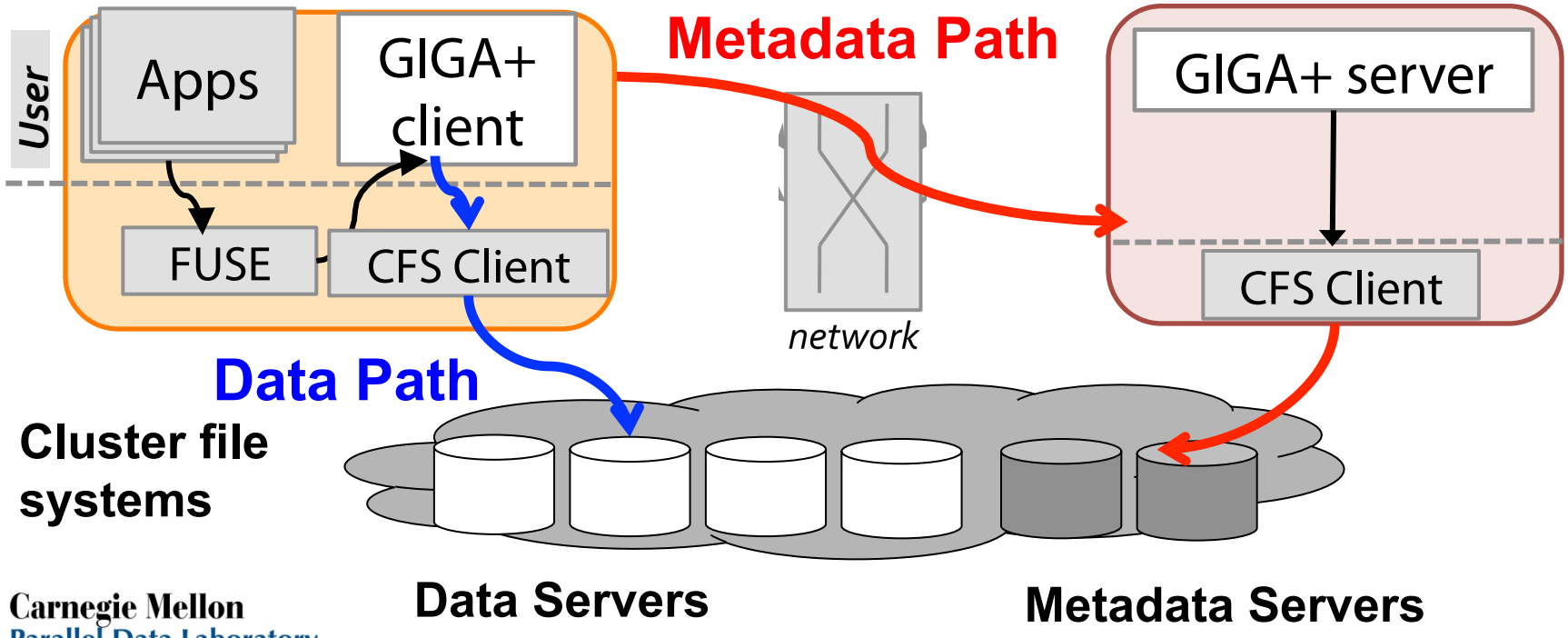
How GIGA+ Stores Directories?

- Large directories split into partitions across servers
  - Hash mapping: hash(filename) → partition (server ID)
  - Incremental growth: directory starts with one server
  - Binary splitting without global synchronization



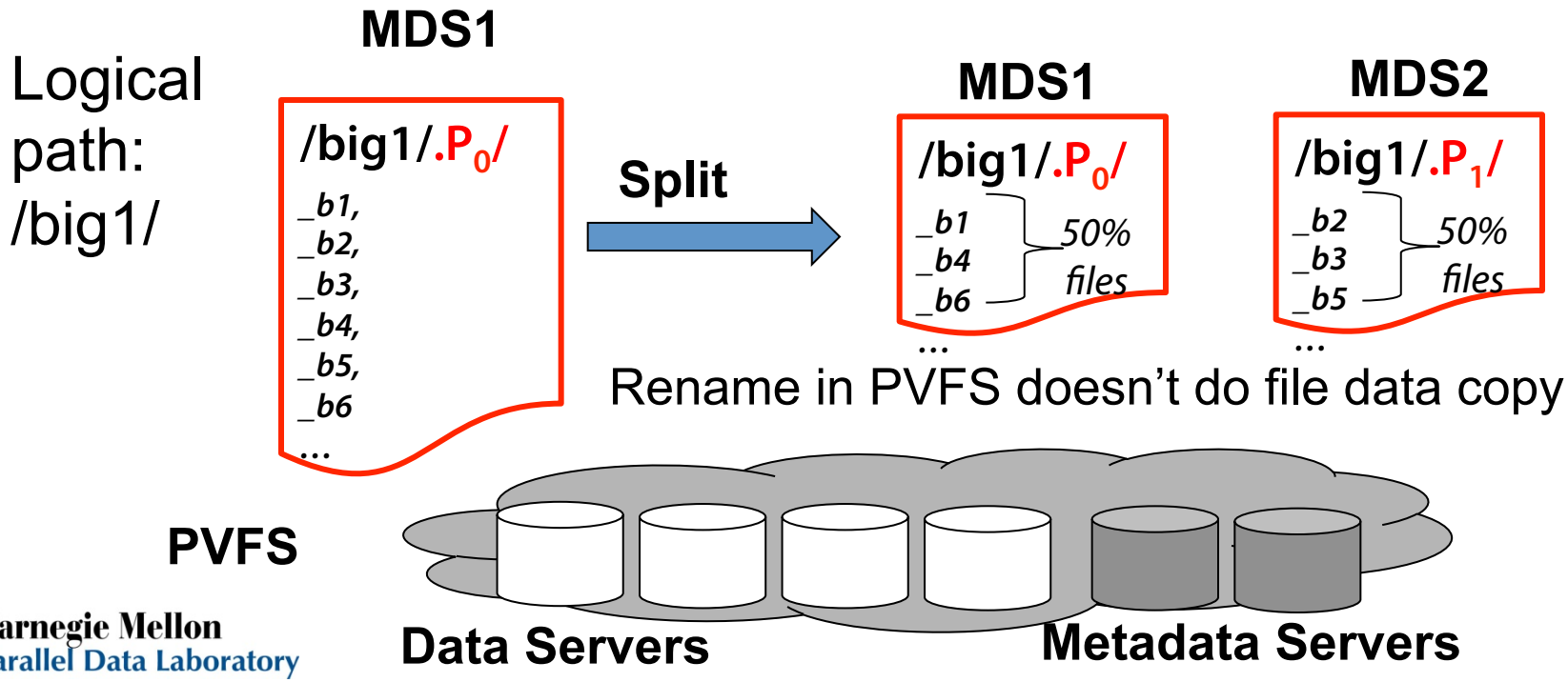
Middleware Layering Approach

- Decoupling Metadata and Data Path
  - Layerable on various federated cluster file systems
  - GIGA+ server load balances metadata
  - Simple scheme uses cluster file system for directories



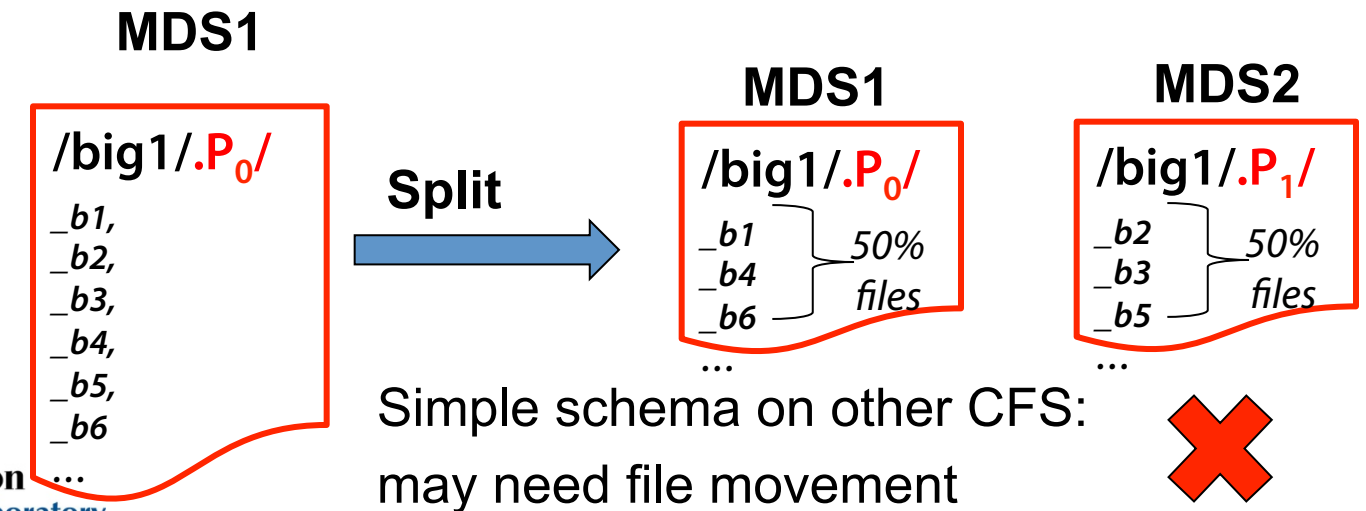
SkyeFS: GIGA+PVFS

- Previous implementation: SkyeFS [Chivetta12]
  - Directory partitions are stored as PVFS directories
  - Splits use atomic, no-copy rename across metadata servers (or GIGA+ would need to do copying)



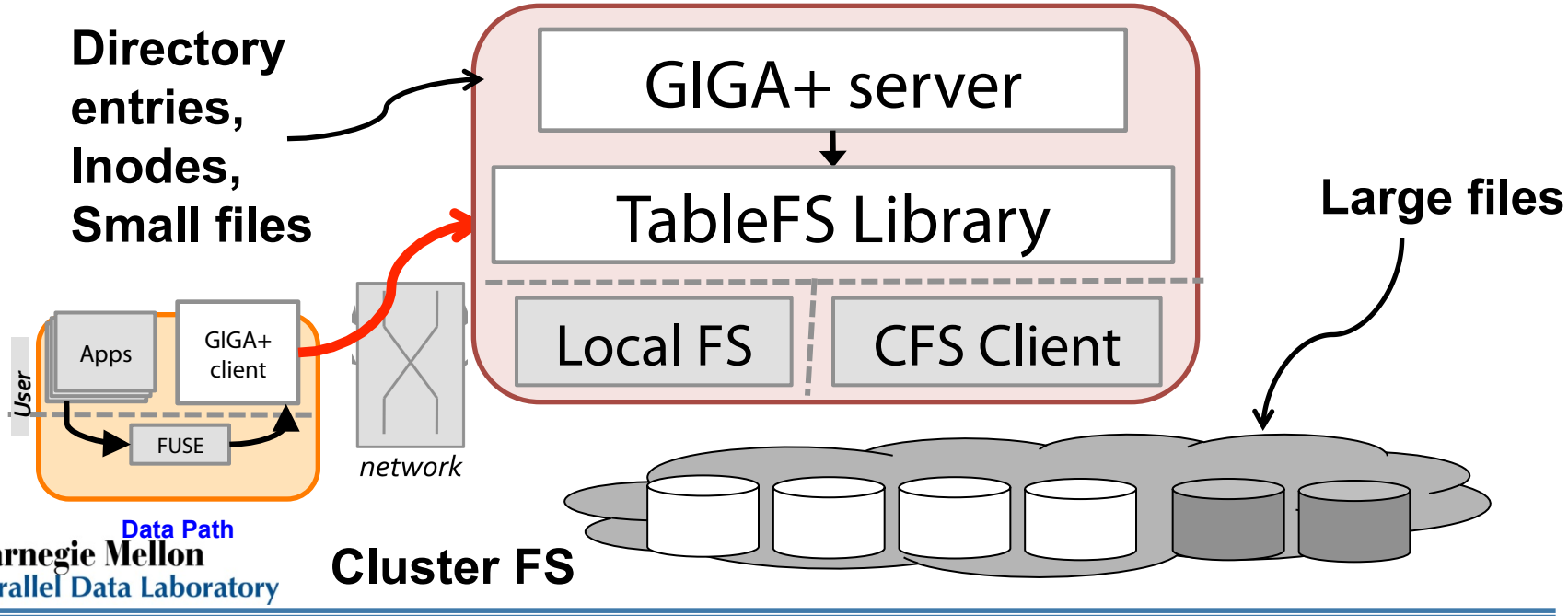
Directory Representaion

- Simple schema: partitions stored as CFS directories
- Implement directories above cluster file systems
  - Decouple rename from cluster file systems
  - Keeps a symbolic link to hidden CFS "objects"
  - Split is to move symbolic links not move files
    - Because HPC has big files, this is essential



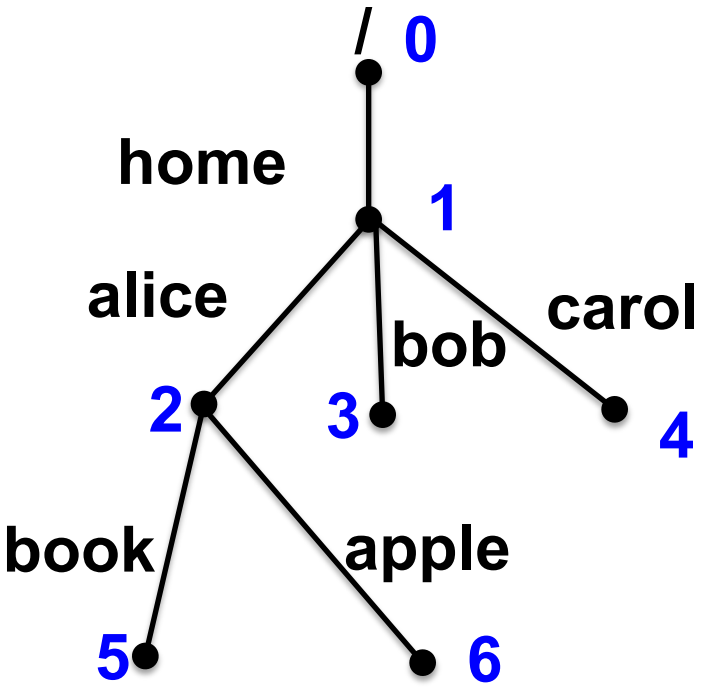
TableFS as Directory Representation

- TableFS [Ren11] manages directory partitions
  - Logically put metadata and small files into a table
    - Use direct CFS paths for large file access
  - Physically, table is managed as LSM Tree [O'Neil96]



Metadata Table Schema

- Key: <Parent inode number, hash(filename)>
- Value: filename, inode attributes, inlined file data (or symbolic link to object in cluster FS).



Lexicographic order

Key	Value
<0,hash(*home*)>	1, “home”, struct *stat*
<1,hash(*alice*)>	2, “alice”, struct *stat*
<1,hash(*bob*)>	3, “bob”, struct *stat*
<1,hash(*carol*)>	4, “carol”, struct *stat*
<2,hash(*book*)>	5, “book”, struct *stat,* Inline file data
<2,hash(*apple*)>	6, “apple”, struct stat, File data pointer

Metadata Table Schema (cont')

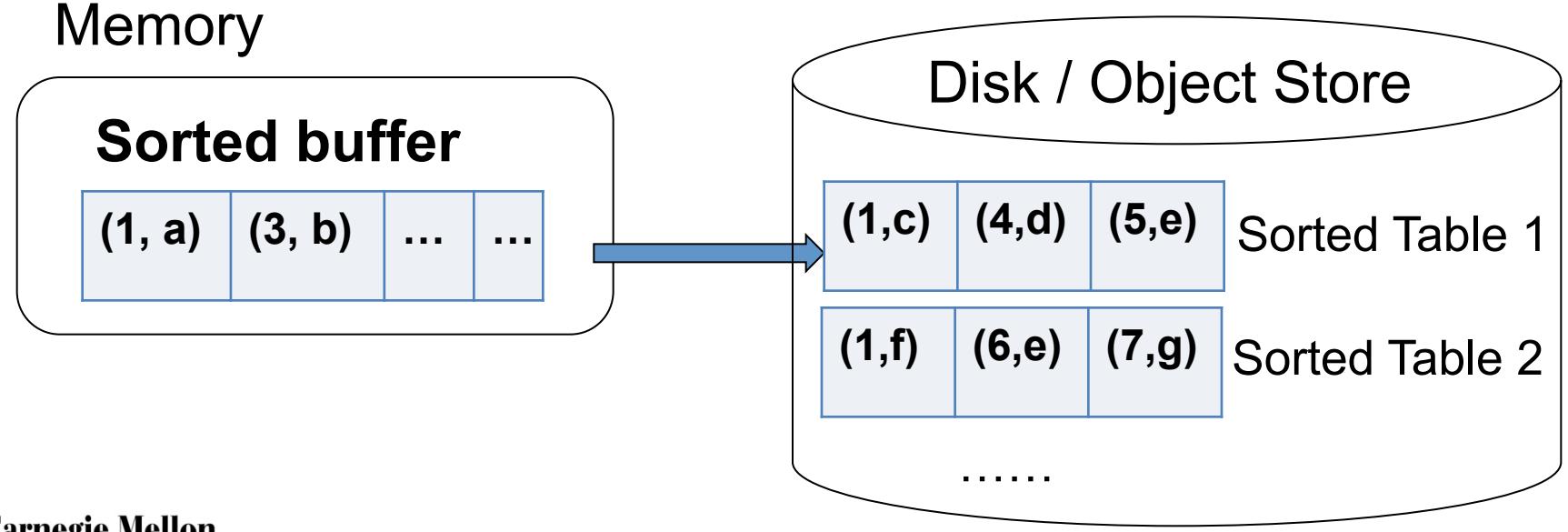
- Advantages:
  - Reduce random lookups by collocating directory entries with inode attributes, and small files
  - "*readdir*" performs sequential scan on the table

Entries in the same directory

Key	Value
<0,hash(*home*)>	1, "home", struct *stat*
<1,hash(*alice*)>	2, "alice", struct *stat*
<1,hash(*bob*)>	3, "bob", struct *stat*
<1,hash(*carol*)>	4, "carol", struct *stat*
<2,hash(*book*)>	5, "book", struct *stat,* Inline file data
<2,hash(*apple*)>	6, "apple", struct stat, File data pointer

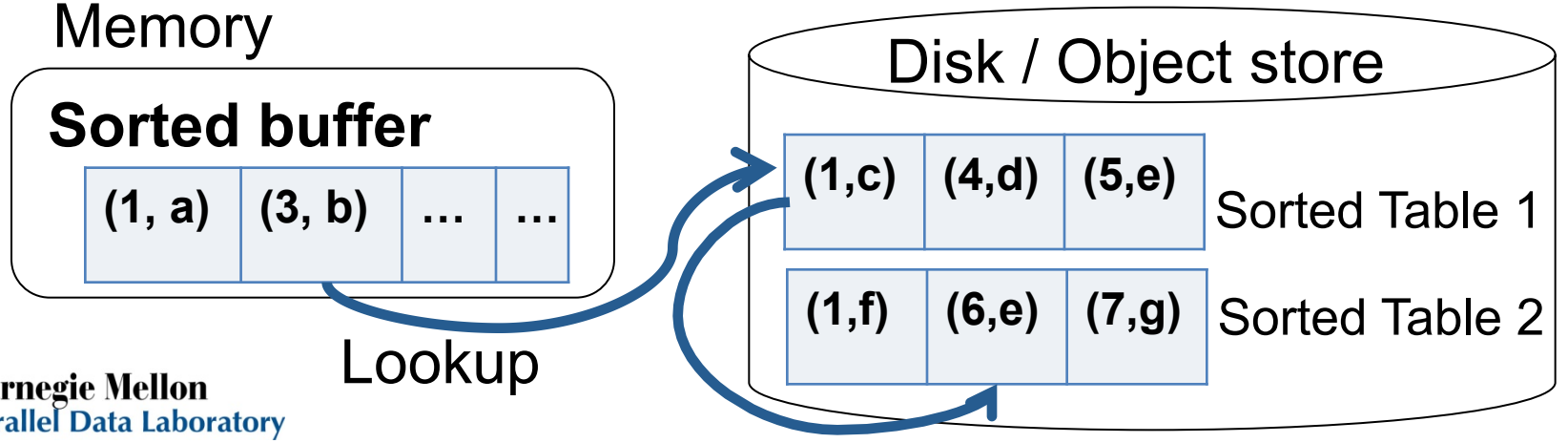
Log-structured Merge Tree

- Insert / Updates in LSM Tree
  - Buffer and sort recent inserts/updates in memory
  - Flush buffer to disk as immutable sorted log
  - Only performs large sequential writes
  - Many fewer random writes vs traditional B-Tree



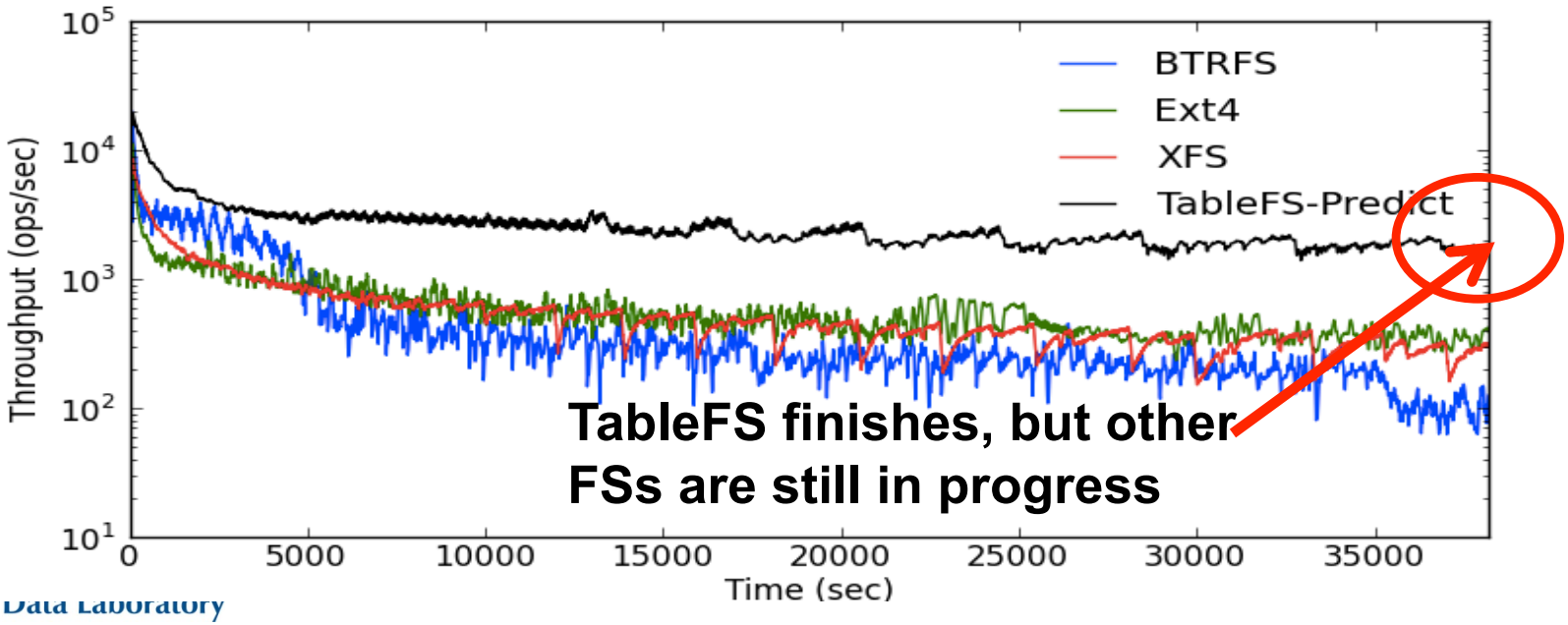
LSM Tree (cont')

- Lookup / Scan
  - Search sorted tables one by one from the disk
  - Bloom-filter and in-memory index reduce lookups
  - Read performance comparable to traditional B-Tree
- Background compaction
  - Merge sort sorted tables with overlapping key range
  - Remove deleted key-value pairs



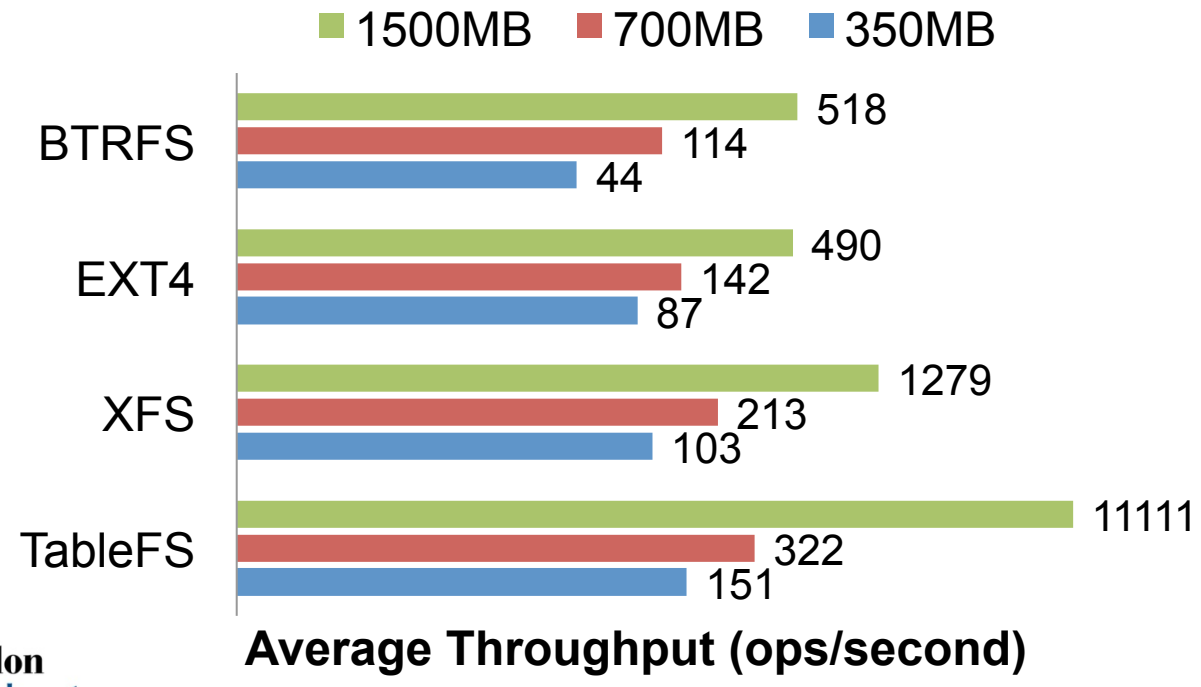
Single Node TableFS Performance

- Creating 100 million zero-length files in one dir.
  - Memory size is 16GB; layered on ext4 local FS
  - Starting from an empty file system
  - Moving throughput average over 10 second window
- TableFS runs *10X faster* than today's local FSs



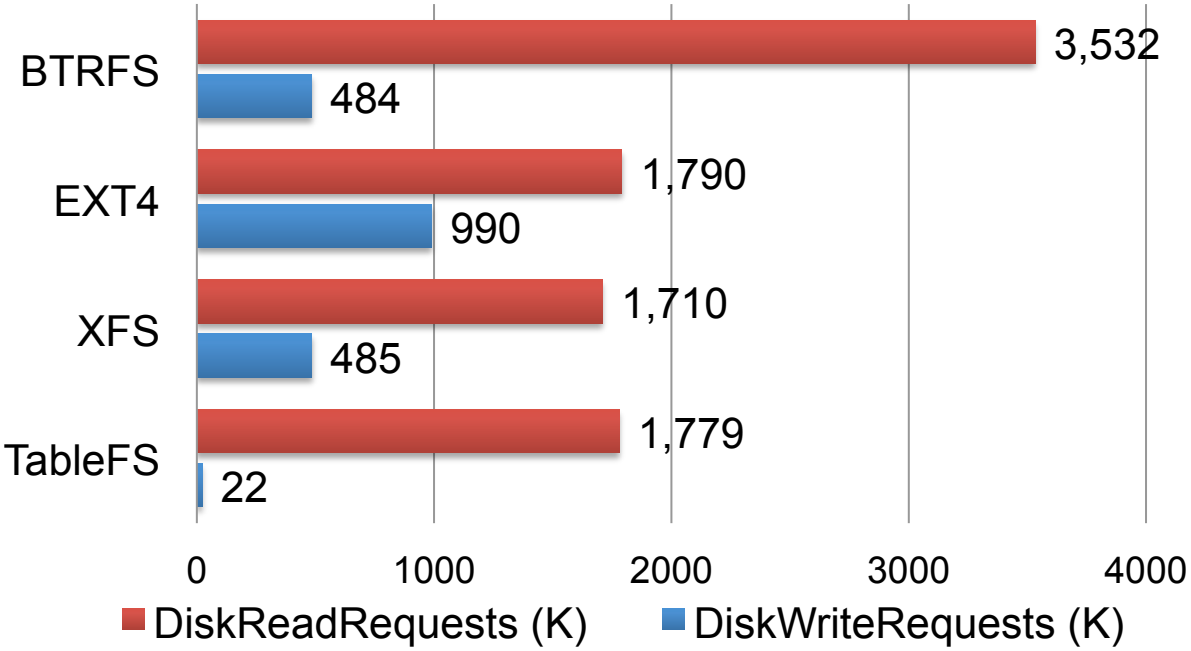
Metadata-only Benchmark

- Workload: issue 2 million random lookups (*stat*), or updates (*chmod* /*utime*) in one node
- TableFS is *1.5 to 10X faster* than local file systems.



Metadata-only: Disk Traffic

- LSM Tree reduces random disk writes
  - Memory size: 350MB

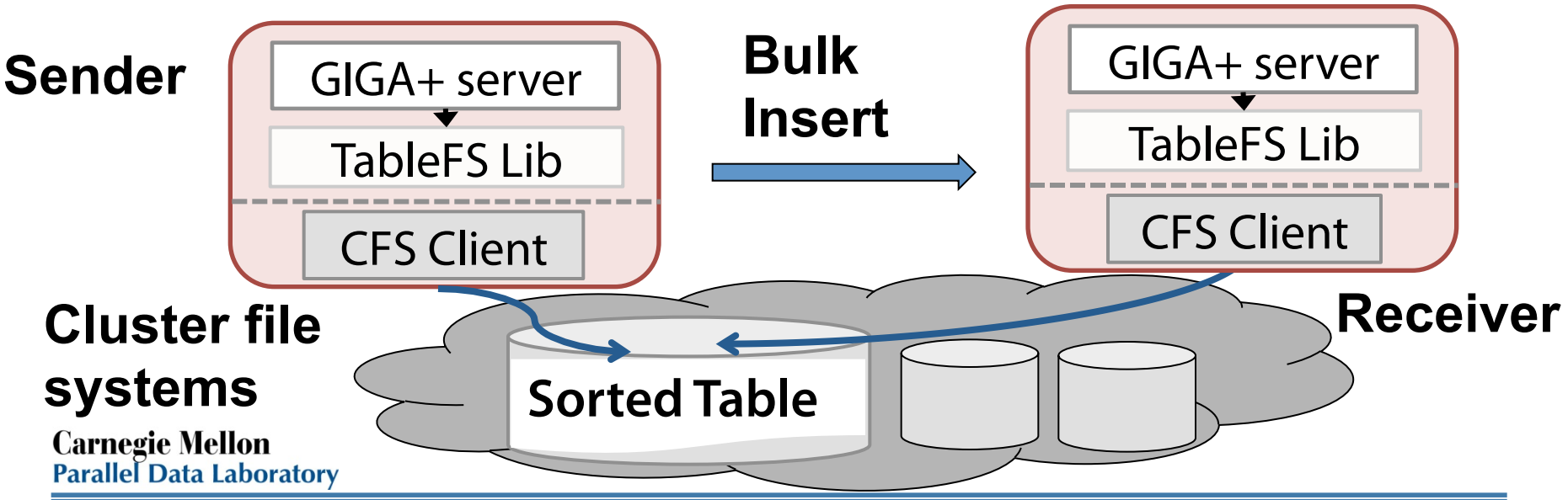


Outline

- Overview
- System Design
  - GIGA+: Distributed Index Technique
  - System Architecture
- Directory Representation
  - Design & Single Node Performance
- ✓ Integrating GIGA+ with TableFS
  - Directory Partition Migration
  - Scalability Result
- Summary

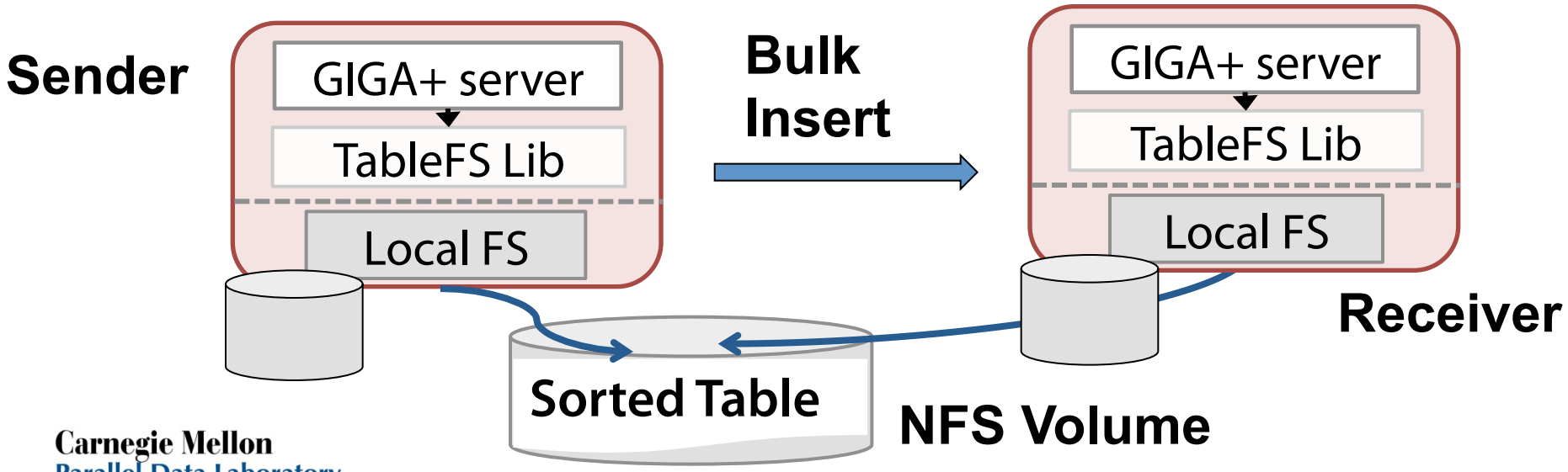
Directory Partition Migration

- TableFS stores partitions as sorted tables in CFS
- Splitting a directory partition:
  - Bulk insert sorted tables into TableFS of target server
    - By passing only pathnames in RPC messages
  - No file data movement during split of directory partition



Current Implementation

- Use Google LevelDB (a variant of LSM Tree)
  - Store sorted tables in local file system
- Splitting a directory partition
  - Move split sorted tables through NFS volume
- In-progress: port to Federated HDFS and PanFS



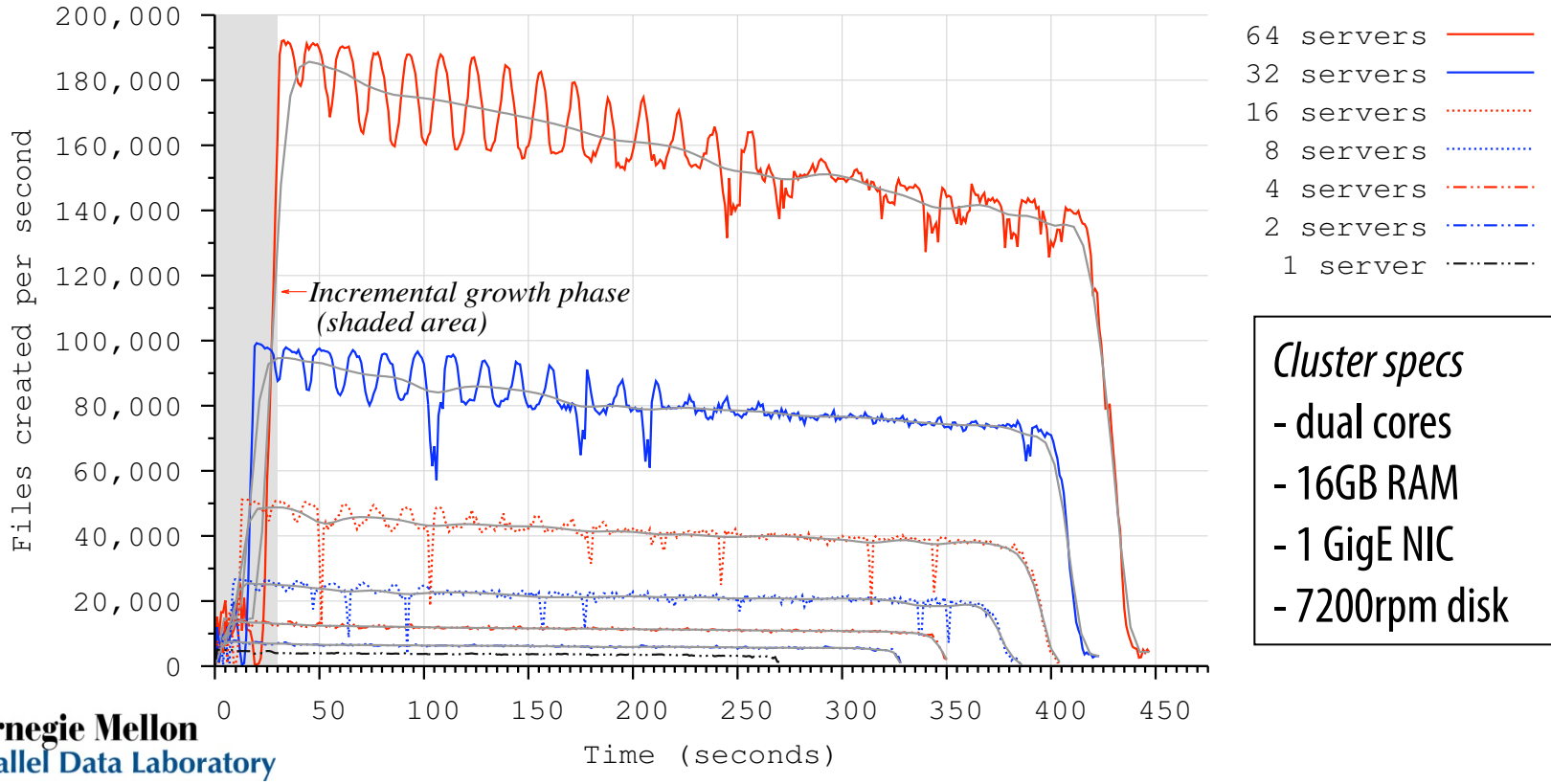
Prelim. Giga+TableFS Scalability

- GIGA+TableFS run on a 64-node cluster
  - Uses local file systems (Ext4) for TableFS
  - NFS to move split directory partitions

- Workload:
  - Concurrent create in a strong scaling experiment
  - Creating 1 million files per server
  - All files are created in one directory
  - Starting from an empty file system

Prelim. Giga+TableFS Scalability

- Performance result
  - Scales to 64 nodes and ~160K files creates/s
  - Achieve almost linear scalability



Cluster specs
- dual cores
- 16GB RAM
- 1 GigE NIC
- 7200rpm disk

Summary

- GIGA+ for scalable, parallel directory indexing
  - Incremental growth without global synchronization
- TableFS to represent directory partitions
  - Packing metadata and small files into large objects
- Combine the two to scale metadata service

- On-going work
  - Layering GIGA+TableFS on PanFS, Federated HDFS
  - Improve partition splitting and LevelDB compaction

Reference

- [Ren12] *TableFS: Enhancing metadata efficiency in local file systems.* Kai Ren and Garth Gibson. CMU Techical report CMU-PDL-12-110
- [Chivetta12] *SkyeFS: Distributed Directories using Giga+ and PVFS*. Anthony Chivetta, Swapnil Patil & Garth Gibson. Technical Report CMU-PDL-12-104, May 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.
- [DARPA08] *ExaScale Computing Study:Technology Challenges in Achieving Exascale Systems*. Peter Kogge and et al.
- [Patil07] *GIGA+: Scalable Directories for Shared File Systems.* Swapnil V. Patil, Garth A. Gibson, Sam Lang, Milo Polte, PDSW 2007
- [O'Neil96] The log-structured merge-tree (LSM-tree). Patrick ONeil and et al. *Acta Informatica,* 1996