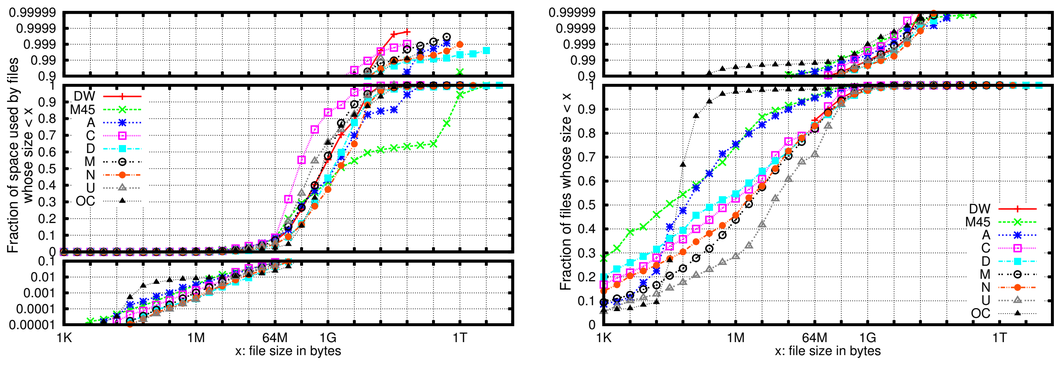
Scaling Metadata in HDFS

Lin Xiao, Garth Gibson

Need for Scalable Metadata

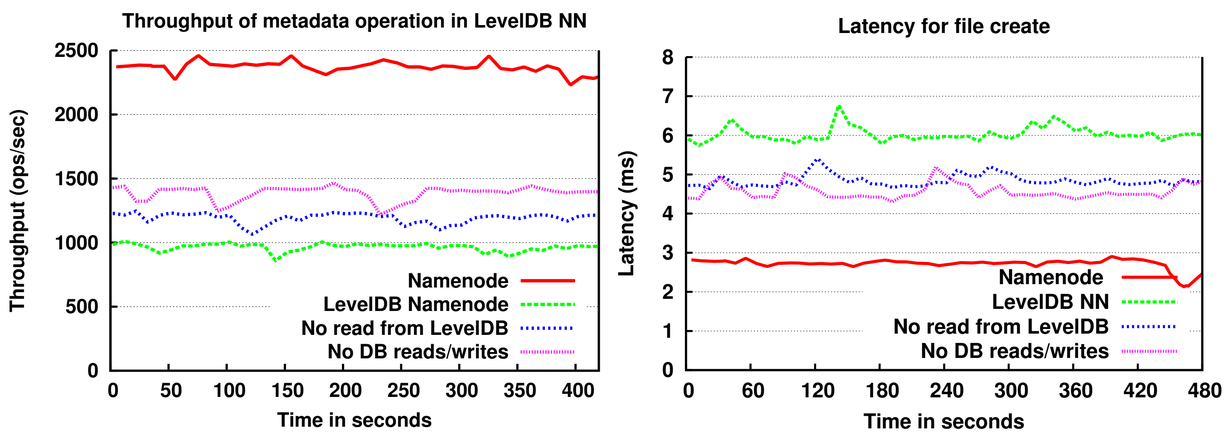
- Big Data is lots of data and lots of files too
  - Lots of files means lots of metadata operations



- Even modern file systems don't scale metadata well
  - HPC: federated independent subtrees is the norm
  - Hadoop: until recently only one Namenode
- We are exploring multiple strategies to scale metada
  - Federated namenodes with client middleware
    - Shard files and replicate namespace
    - Use a lock server for isolation and atomicity
  - Extend namenode to out-of-core metadata with LevelDB
  - Re-code client/namenode to wrap on distributed table

HDFS with Out of Core Metadata

- Modified Hadoop 0.23.1 NameNode with single node LevelDB
  - Store namespace & blocks map in LevelDB and inode cache
  - Lookup in LevelDB for misses in inode cache
- 6 clients, 2 threads each mkdir + create
- LevelDB stores on tmpfs to factor out disk performance
- High overhead for using LevelDB
  - Inode cache codepath is too long: redesign cache
  - Negative lookups for file creates: bloom filter helps

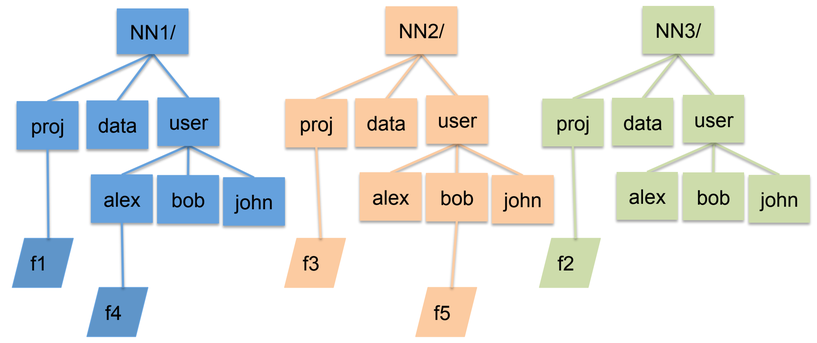


Future:Distributed Out-of-core Namenodes

- Use distributed table as the global storage for metadata
  - Higher latency:
    - Longer code path, more than one RPC, disk access
  - Auto capacity and load balancing, uncontrolled by HDFS
  - Interesting balancing questions: should HDFS control balancing, or ignore it and allow table to handle it?
- Explore replicated state machine (EPaxos) for metadata
  - Fast, scalable Epaxos: faster failover than primary/backup

Scaling Federated HDFS: ShardFS

- Middleware layer on top of HDFS distributes files
  - Each namenode only sees its own namespace
  - Datanodes can serve all namenodes (federated)
  - Distribute files: by hash(fullpath) or hash(filename)
  - Only one namenode to handle a single file operation
  - Client middleware replicates namespace (directories) and implements distributed transactions (mkdir,rename)

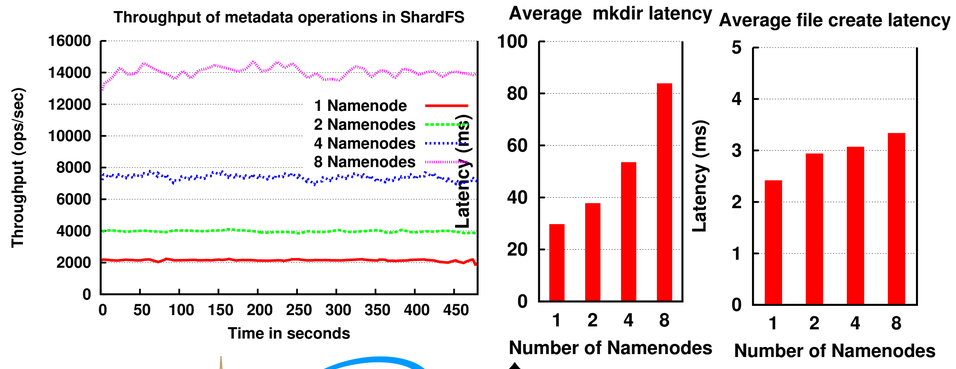


Concurrency Challenges

- Making changes to middleware-replicated namespace
  - client1: mkdir /a/b/c (/a/b on NN1 then /a/b/c on NN2)
  - client2: create /a/b/c/f1 (NN2) &create /a/b/c/f2 (NN1)
  - client2 fails to create f2 b/c parent dir doesn't exist on NN1
- Fine-grained tree locking (slow) for replicated directory ops
  - Use a locking service that parallels the namespace
  - Read lock root to parent directory of targeted directory
  - Write lock only the directory in question
- Be optimistic (fast) for file operations
  - Try with no locks at all for files, eg. /a/b/c/f1 succeeds
  - If operation fails, check for/wait on parent lock (recur up as needed) e.g., client2 waits for /a/b/c to be created before creating f2
- Write-ahead-log and primary/backup NN for failures
  - Client failures detected by reading zookeeper lock
  - Zookeeper lock is also a write-ahead log

ShardFS Experiment

- ShardFS implemented on Hadoop 0.23.1
  - Client-side implementation,tree locking w/zookeeper
  - Hash(fullpath) as the sharding function
- Workload:
  - 2 threads/client each mkdir and create files w/o conflicts
  - 6 clients/namenode
- Throughput almost increases linearly
- Higher mkdir latencies



Intel Science & Technology Center for Cloud Computing

Carnegie Mellon University

Georgia Tech

intel

PRINCETON UNIVERSITY

UC Berkeley