CalvinFS: Consistent WAN Replication and Scalable Metdata Management for Distributed File Systems

Background

- Scalable solutions provided for data storage, why not file systems?

Motivation

- Often bottlenecked by the metadata management layer
- Availability susceptible to data center outages
- Still provides expected file system semantics

Key Contributions

- Distributed database system for scalable metadata management
- Strongly consistent geo-replication of file system state

Calvin: Log

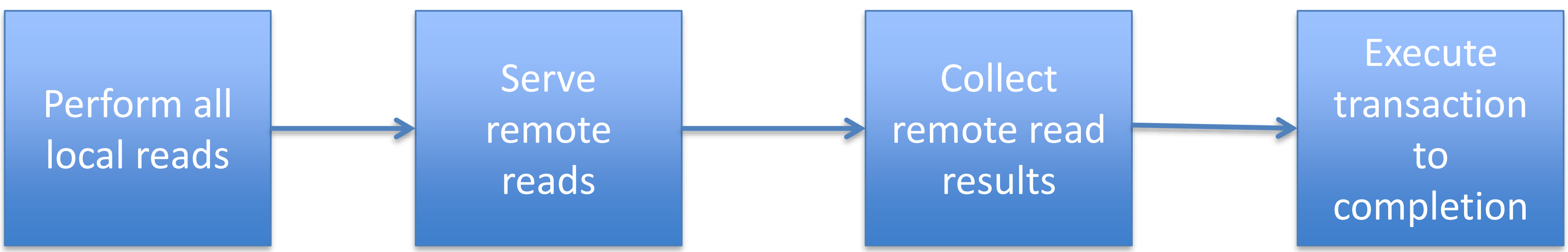
- Many front end servers
- Asynchronously-replicated distributed block store
- Small number of “meta-data” log servers
- Transaction requests are replicated and appended, in order, by the “meta log”

Calvin: Storage Layer

- Knowledge of physical data store organization and actual transaction semantics
- Read/write primitives that execute on one node
- Placement manager
- Multiversion key-value store at each node, plus consistent hashing mechanism

Calvin: Scheduler

- Drives local transaction execution
- Fully examines transaction before execution
- Deterministic locking
- Transaction protocol:



- No distributed commit protocol

CalvinFS Architecture

- Design Principles:
  - Main-memory metadata store
  - Potentially many small files
  - Scalable read/write throughput
  - Tolerate slow writes
  - Linearizable and snapshot reads
  - Hash-partitioned metadata
  - Optimize for single-file operations

- Components
  - Block store
  - Calvin database
  - Client library

CalvinFS Block Store

- Variable-size immutable blocks
  - 1 byte to 10 megabytes
- Block storage and placement
  - Unique ID
  - Block "buckets"
  - Global Paxos-replicated config file
  - Compacts small blocks

CalvinFS Metadata Management

- Key-value store
  - Key: absolute path of file/directory
  - Value: entry type, permissions, contents

```
KEY:
  /home/calvin/fs/paper.tex
VALUE:
  type:           file
  permissions:    rw-r--r-- calvin users
  ancestor-
    permissions: rwxr-xr-x calvin users
                 rwxr-xr-x calvin users
                 rwxr-xr-x root root
                 rwxr-xr-x root root
  contents:       0x3A28213A 0 65536
                  0x6339392C 0 65536
                  0x7363682E 0 34061
```

Metadata Storage Layer

- Six transaction types:
  - Read(path)
  - Create{File, Dir}(path)
  - Resize(path, size)
  - Write(path, file_offset, source, source_offset, num_bytes)
  - Delete(path)
  - Edit permissions(path, permissions)

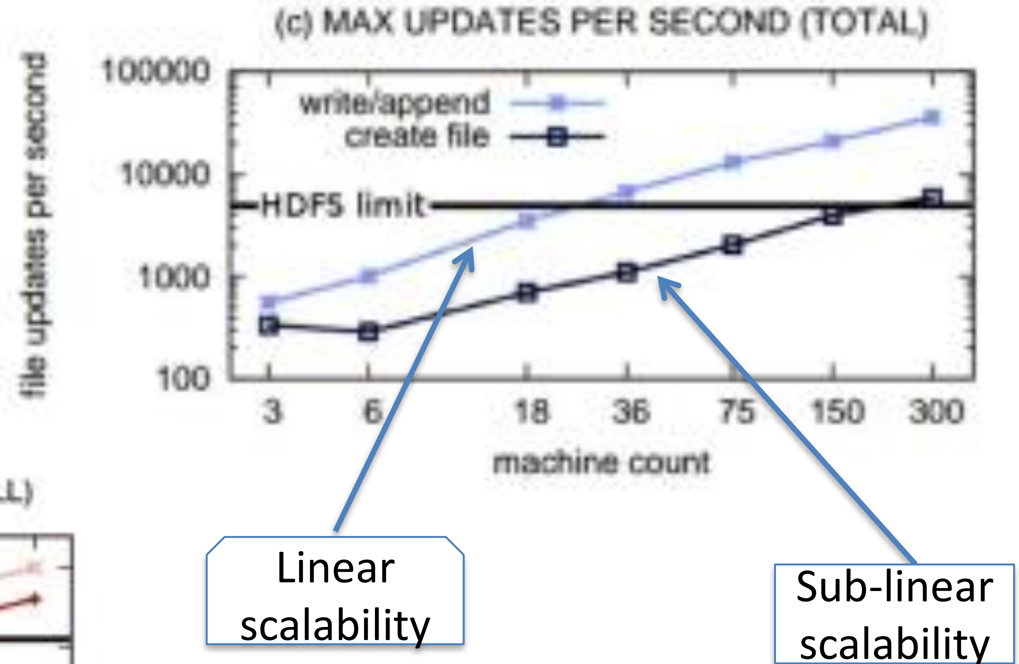
Recursive Operations on Directories

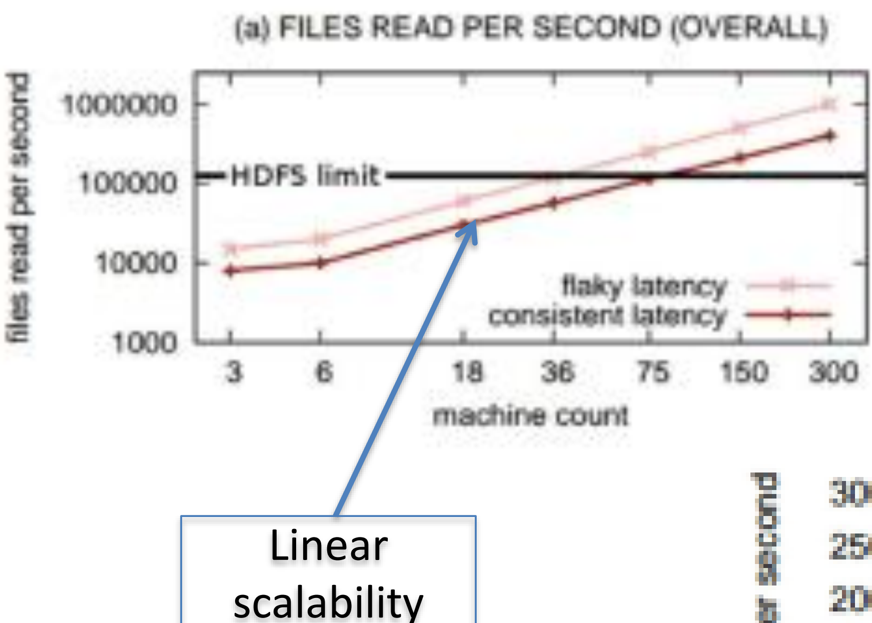
- Use OLLP
- Analyze phase
  - Determines affected entries and read/write set
- Run phase
  - Check that read/write set has not grown

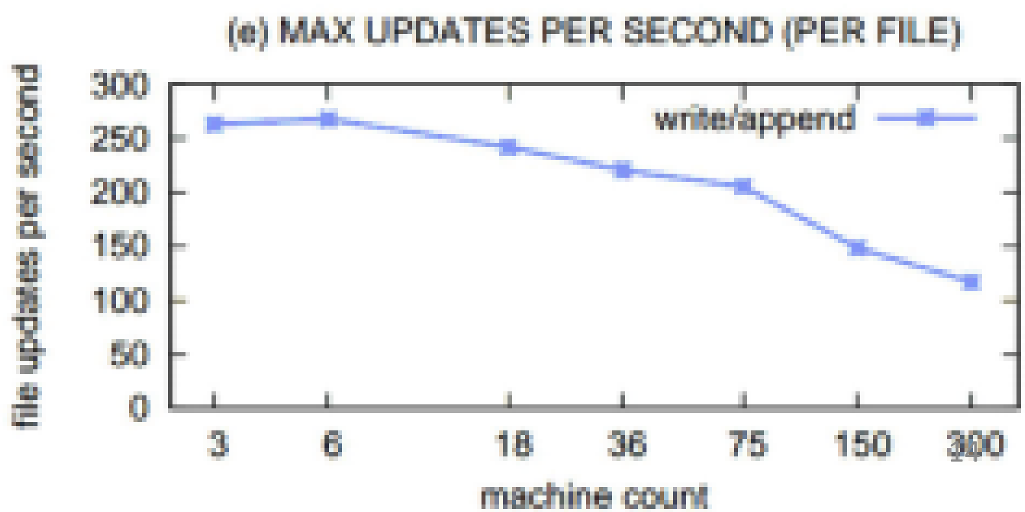
Performance: File Counts and Memory Usage

- 10 million files of varying size per machine
- Far less memory used per machine
- Handles many more files than HDFS

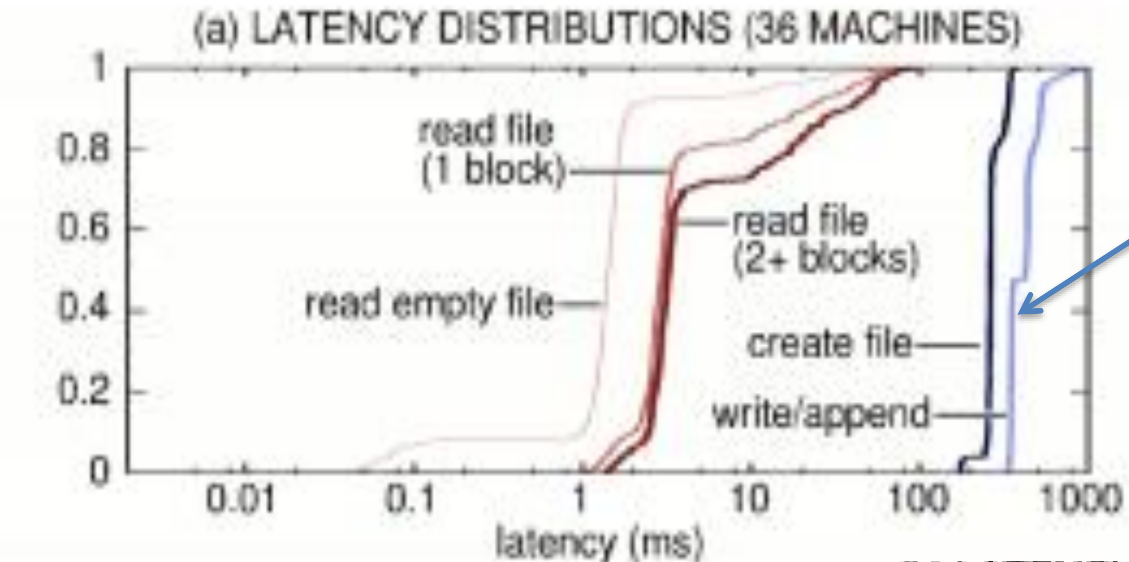
Performance: Throughput



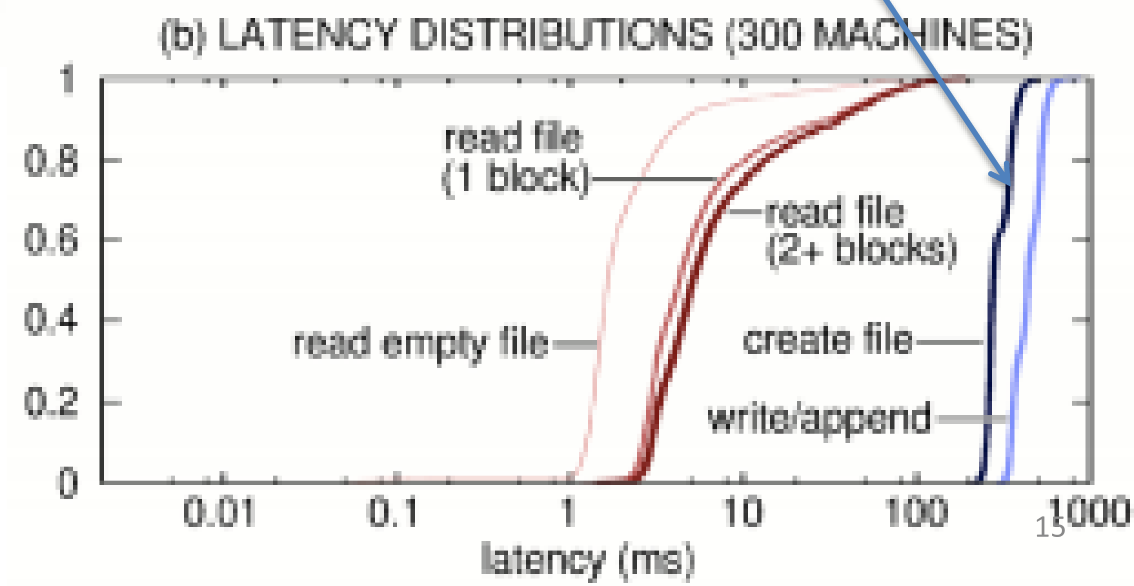




Performance: Latency

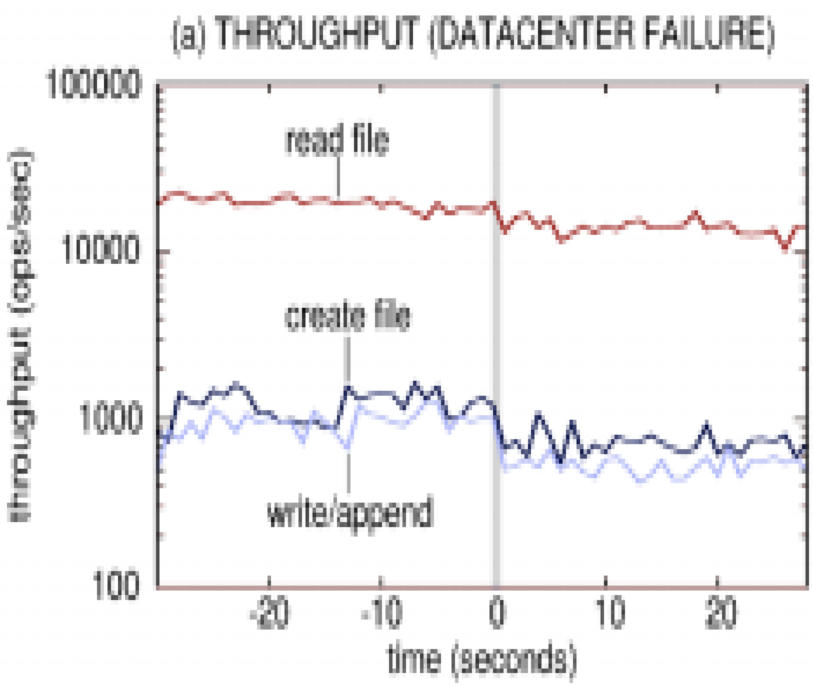


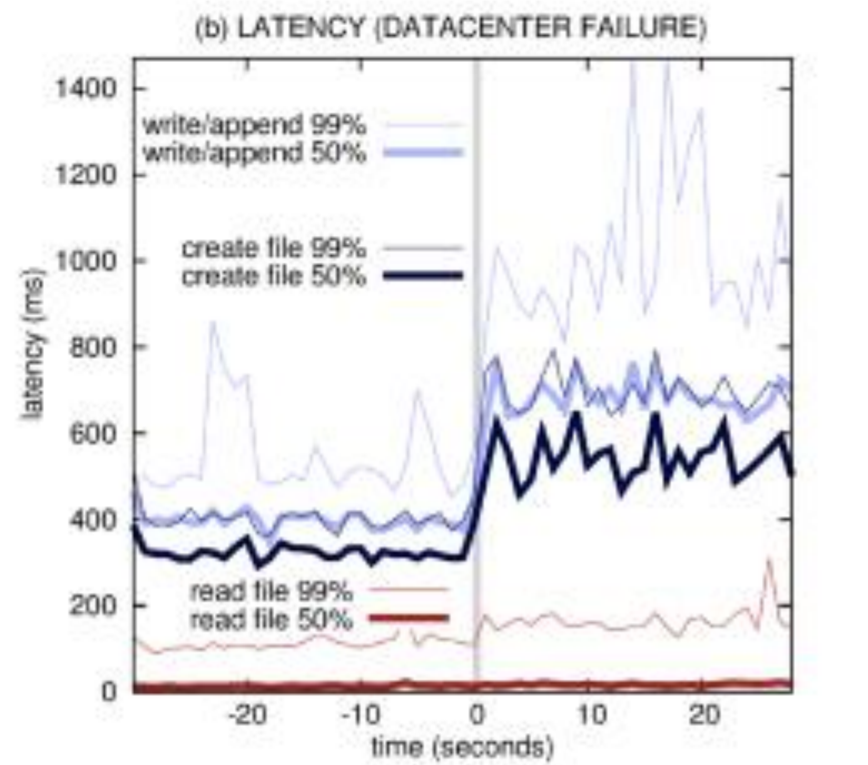
Write/append latency dominated by WAN replication



Performance: Fault Tolerance

- Able to tolerate outages with little to no hit to availability





Discussion

Cons

- File creation is distributed transaction, doesn't scale well
- Metadata operations have to recursively modify all entries in affected subtree
- File-fragmentation addressed using mechanism that entirely rewrites files

Pros

- Fast metadata management
- Deployments are scalable on large clusters
- Huge storage capabilities
- High throughput of reads and updates
- Resistant to datacenter outages