Staggeringly Large Filesystems

Evan Danaher

CS 6410 - October 27, 2009

Outline

Outline

1. Large Filesystems
2. GFS
3. Pond

What is “Large”

- “Internet Scale”
- Web 2.0
- GFS
  - Thousands of machines[1]
  - Hundreds of active jobs[1]
  - 4 Petabyte filesystems[1]
  - 40 GB/s read/write load[1]
  - Future: Spanner: millions of machines, Exabytes of storage[1]
- Pond
  - $\sim$1000 machines
  - Designed to scale to millions.

[1] Jeff Dean, LADIS 2009

Different Approaches

- Local/Wide Area
- Communication Model
  - P2P
  - Tiered
  - Centralized
- Trusted/Untrusted Nodes

Outline

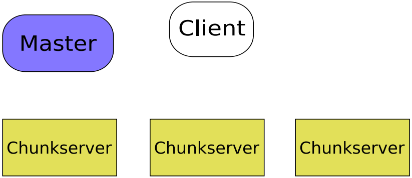
Motivation

- Component failures are common
  - Durability
  - Fast startup important
- Huge files
- Append, not random writes
- Large streaming reads or small random reads

Design

- Very pragmatic
- Fast startup after failures
- Co-designed with applications
- Favor bandwidth over latency
- Trusted Nodes
- Centralized

Key Points

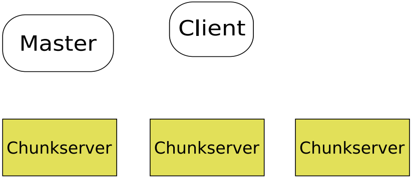


- Chunkservers store data
- Centralized master controls everything
- Relaxed consistency model

Interface

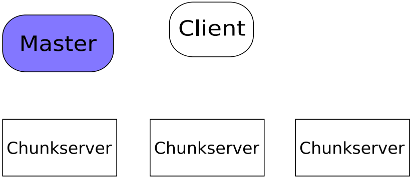
- Create, Delete, Open, Close, Read, Write
- Snapshot: Cheap copy at a point in time
- Record append
    - Safe concurrent append by multiple applications

Chunkservers



- Store fixed 64MB chunks
  - Indexed by chunkID
  - Checksummed
  - Fixed size simplifies design
  - Stored as files on local filesystem
  - Replicated (default 3 times)
- Authoritative on which chunks are stored

Master



- Single master with all metadata in memory
  - Fast and simple
  - Global decisions are easy
  - Less than 64 bytes / 64MB chunk
- Requests chunk data from chunkservers on startup
- Authoritative on file to chunk mapping

Master Durability

- Single point of failure
- Log all operations to replicated log
- Checkpointed for quick recovery
- Readable shadow master replicas

Garbage Collection

- On deletion, rename to a hidden file
- Remove hidden files after 3 days
- Periodically scan and erase orphan chunks
- Each chunkserver HeartBeat sends some stored chunks
- In master response, send unknown ones
- Simple cleanup for all cases

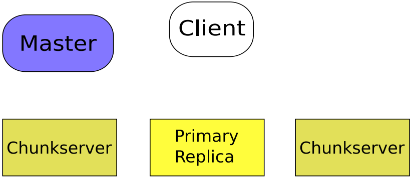
Consistency Model

- Relaxed consistency model
- Namespace mutations are atomic
- Data is less clear
  - Consistent: All clients see same data
  - Defined: Consistent, mutation is correct
  - Concurrent writes: consistent, not defined
  - Concurrent record appends: defined interspersed with inconsistent
- Record append has weak guarantees
  - Record was appended atomically at least once
  - May also be junk the applications should ignore

Locking

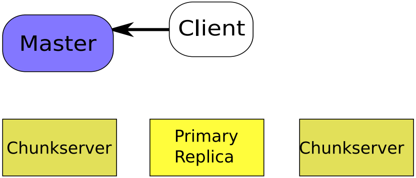
- Files are identified by paths
- No directory structure
- To lock a file
    - Read locks on ancestor directories
    - Read or write lock on requested file(s)
    - Order by depth, lexicographically

Mutation Control



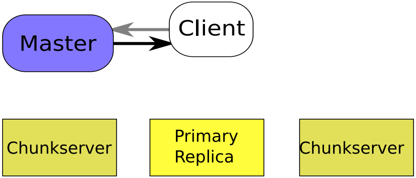
- One replica, the primary, gets a chunk lease.
- The primary picks the ordering of mutations.
- Times out after 1 minute, usually renewed.
- Can be revoked (e.g., for snapshots)

Lifecycle of a Mutation



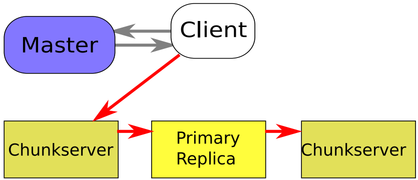
- Client requests lease holder from master
    - Created if necessary

Lifecycle of a Mutation



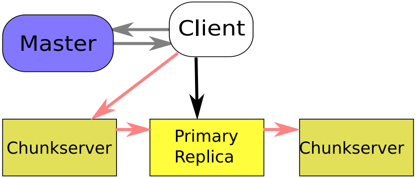
- Master sends primary, secondary replicas

Lifecycle of a Mutation



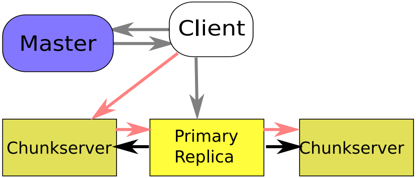
- Client pushes data to all replicas
  - Linear forwarding pipeline

Lifecycle of a Mutation



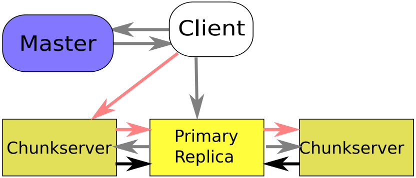
- Client sends write request to primary

Lifecycle of a Mutation



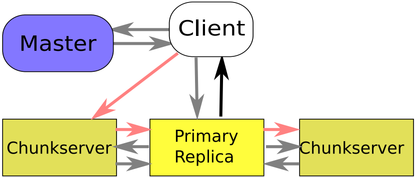
- Primary forwards write request to secondaries

Lifecycle of a Mutation



- Secondaries report completion to primary

Lifecycle of a Mutation



- Primary reports completion to client

Record Append

- Guarantees at least one atomic append
- Uses basic mutation control flow
- Failure at any replica leads to a retry
- Might leads to inconsistent data/duplicates
  - Filtered out by the application
- Can't span chunks
  - If it doesn't fit, start a new chunk
  - Limit to 1/4 chunk size to avoid fragmentation

Detecting Failure

- Chunk version numbers detect stale replicas
- Regular handshakes with master detect failed chunkservers
- Checksums detect corrupted data
  - Checksums are stored in memory
  - They are persisted in a log separate from data
  - Idle clients may checksum unused data
- Failed replicas are ignored by the master
  - Cloned as soon as possible to avoid data loss

Replica Placement

- Below-average disk utilization
  - Equalize load across chunkservers
- Limit “recent” creations per chunkserver
  - Avoid flooding a chunkserver with writes
- Spread replicas across racks
  - Redundancy, read bandwidth
- Periodically rebalance
  - Gradually fills up new chunkservers

High Availability

- Fast recovery
- Chunk replication
- Master replication
- Checksums

Performance

Test Clusters

Cluster	A	B
Chunkservers	342	227
Available disk space	72 TB	180 TB
Used disk space	55 TB	155 TB
Number of Files	735 k	737 k
Number of Dead files	22 k	232 k
Number of Chunks	992 k	1550 k
Metadata at chunkservers	13 GB	21 GB
Metadata at master	48 MB	60 MB

- Cluster A: R&D for over 100 engineers
- Cluster B: Production data processing

Performance

Test Clusters

Cluster	A	B
Chunkservers	342	227
Available disk space	72 TB	180 TB
Used disk space	55 TB	155 TB
Number of Files	735 k	737 k
Number of Dead files	22 k	232 k
Number of Chunks	992 k	1550 k
Metadata at chunkservers	13 GB	21 GB
Metadata at master	48 MB	60 MB

- Metadata on chunkservers: mostly checksums
- Metadata on master: small

Performance

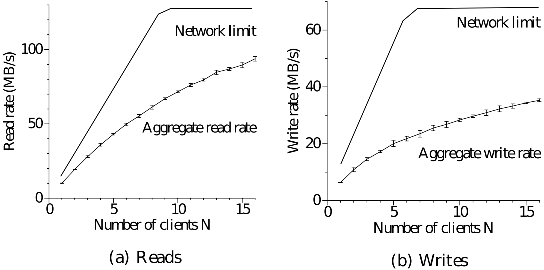
Test Clusters

Cluster	A	B
Chunkservers	342	227
Available disk space	72 TB	180 TB
Used disk space	55 TB	155 TB
Number of Files	735 k	737 k
Number of Dead files	22 k	232 k
Number of Chunks	992 k	1550 k
Metadata at chunkservers	13 GB	21 GB
Metadata at master	48 MB	60 MB

- Roughly 50-100 MB/server
  - Fast recovery

Performance

Throughput



- High throughput for reads and writes

Performance

Usage

Cluster	A	B
Read rate (last minute)	583 MB/s	380 MB/s
Read rate (last hour)	562 MB/s	384 MB/s
Read rate (since restart)	589 MB/s	49 MB/s
Write rate (last minute)	1 MB/s	101 MB/s
Write rate (last hour)	2 MB/s	117 MB/s
Write rate (since restart)	25 MB/s	13 MB/s
Master ops (last minute)	325 Ops/s	533 Ops/s
Master ops (last hour)	381 Ops/s	518 Ops/s
Master ops (since restart)	202 Ops/s	347 Ops/s

- Majority read
- Low master load

Key Points

- Chunkservers store 64M chunks
  - Stored redundantly
- Centralized master controls everything
  - Stores all metadata in memory
  - Logs replicated for durability
- Relaxed consistency model
  - Consistent/defined segments
  - Weak record append guarantee

Outline

Design Goals

- More general storage model
- Only trusts infrastructure in aggregate
  - Failures or malicious users
  - Churn
- Data is universally available
- Good consistency model

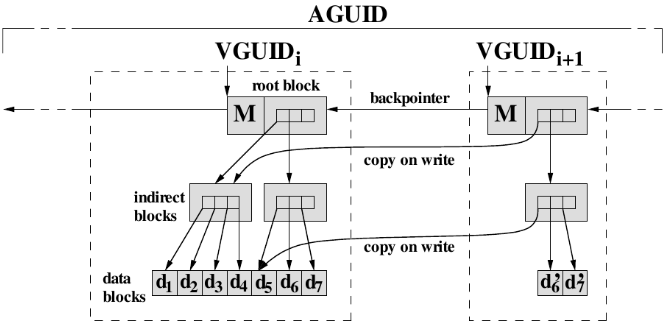
Key Points

- Versioned data model
- Byzantine agreement for primary copy
- Erasure-coded archive for durability

Data Model

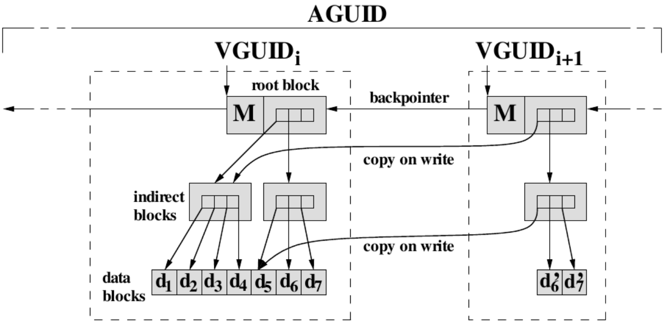
- Allow arbitrary writes via CoW.
- Data Object ("file")
  - Sequence of read-only versions
  - Simplifies caching, replication
  - Allows time travel

Data Object



- B-Tree-like structure
- Each block is referenced by a Block GUID
  - Cryptographically-secure hash of its contents

Data Object



- Root BGUID of a version is its Version GUID
- Active GUID describes a set of versions
- Shared copies for free

Tapestry

- Virtualizes Resources
- Host and resources are identified by GUIDs
- Hosts publish GUIDs of their resources
- Messages are addressed to GUIDs
- Tapestry routes messages to a nearby host with the GUID.

Primary Replica

- Each object has a “primary replica”
  - Maintains AGUID to VGUID mapping
  - Serializes updates.
- Really a collection of servers: the inner ring
- Uses a Byzantine agreement protocol
  - assume any failure of less than 1/3 of the group
- MAC within the group, public key outside
- Proactive threshold signatures allow churn within this group without breaking the public key

Replication/Consistency

- BGUID verifies contents of a block
  - Safe replication of blocks is easy
- AGUID to VGUID mapping changes
- Primary copy replication:
  - Primary replica applies all updates
  - Creates a certificate mapping AGUID to the most recent VGUID.
  - Client can verify most recent using a nonce.
  - Response contains nonce, is signed.

Archiving

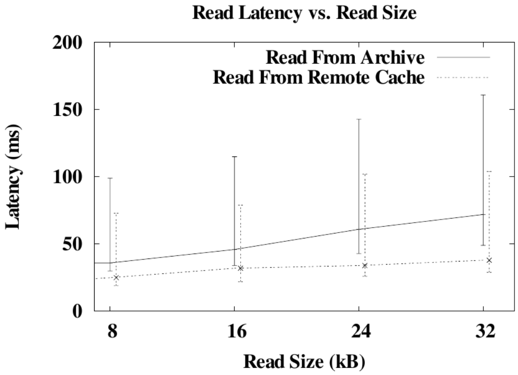
- Replication is space-inefficient.
- Erasure codes (Cauchy-Reed-Solomon):
  - split a block into m fragments
  - encode into $n > m$ fragments
  - any m of the n can reconstruct the file
- Fragments are distributed uniformly deterministically based on BGUID and fragment number

Caching

- To locate a block, look up its BGUID
- On failure, use fragments to reconstruct
- After reconstructing, publishes ownership
    - Future readers can use this copy
- Discarded whenever convenient
    - And unpublished from Tapestry

Performance

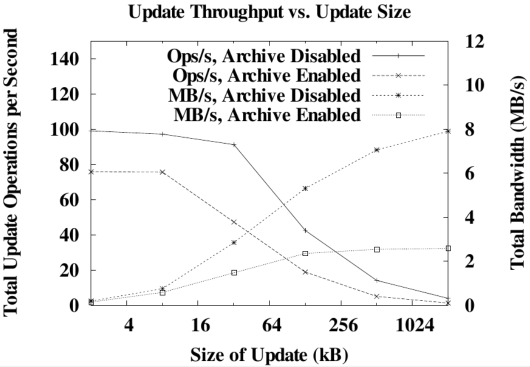
Local Archive Performance



- Reconstruction is within 1.7 times of reading a remote replica.

Performance

Local Throughput



- Computationally limited by inner ring

Performance

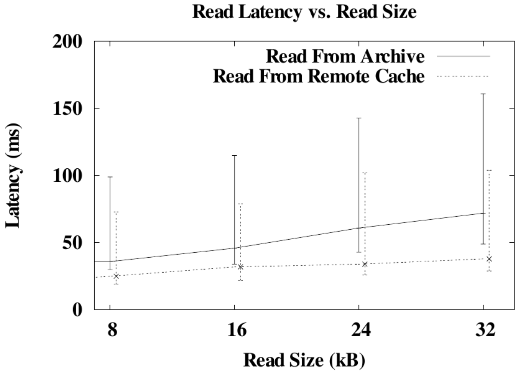
Wide Area Throughput

IR Location	Client Location	Throughput (MB/s)
Cluster	Cluster	2.59
Cluster	PlanetLab	1.22
Bay Area	PlanetLab	1.19

- Local cluster
- Reconstruction is within 1.7 times of reading a remote replica.

Performance

Archive Performance



- Local cluster
- Reconstruction is within 1.7 times of reading a remote replica.

Performance

Andrew Benchmark

	LAN			WAN		
	Linux	OceanStore		Linux	OceanStore	
Phase	NFS	512	1024	NFS	512	1024
I	0.0	1.9	4.3	0.9	2.8	6.6
II	0.3	11.0	24.0	9.4	16.8	40.4
III	1.1	1.8	1.9	8.3	1.8	1.9
IV	0.5	1.5	1.6	6.9	1.5	1.5
V	2.6	21.0	42.2	21.5	32.0	70.0
Total	4.5	37.2	73.9	47.0	54.9	120.3

Times in seconds

- NFS Loopback on Oceanstore vs. NFS
- NFS faster on LAN, slower on WAN.
  - Less computation, less efficient network

Key Points

- Versioned data model
  - Updates are by CoW
  - Simplifies caching
- Byzantine agreement for primary copy
  - No single point of failure
- Erasure-coded archive for durability
  - Provides space-efficient storage

Summary

- Tradeoffs in large filesystems
- GFS
  - Centralized Master
  - Trusted Nodes
  - Weak consistency
- Pond
  - Fully distributed
  - Untrusted Nodes
  - Efficient long-term storage