The Google File System

Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung
ACM SIGOPS 2003
{Google Research}

Vaibhav Bajpai
NDS Seminar 2011

Looking Back

time

- **Classics**
  - **Sun NFS (1985)**
    - stateless server
    - fully POSIX
  - **CMU Andrew FS (1988)**
    - stateful server
    - location independent
- **Fault Tolerant**

  CMU Coda
- **Parallel [HPC]**

  Google FS, Oracle Lustre, IBM zFS
- **Serverless [Peer-to-Peer]**

  xFS, Frangipani, Global FS, GPFS

Outline

- Design Overview
- System Interactions
- Performance
- Conclusion

Design Overview

- Assumptions
- Design Decisions
  - files stored as chunks
  - single master
  - no data caching
  - familiar, customizable API
- Example
- Metadata
  - in-memory DS
  - persistent DS

Design Overview

Assumptions

- component failures are a norm rather than an exception.
- multi-GB files are a common case.
- mutation on files:
  - random read|writes are practically non-existent.
  - large concurrent|sequential appends are common.
  - large streaming reads are common.
- high bandwidth > low latency

optimize

Design Overview

Design Decisions

- files stored as chunks
- reliability through replication | across 3+ chunk servers
- single master
- no data caching
- familiar but customizable API

Design Decisions

Files stored as Chunks

- files divided into **64MB** fixed-size chunks
- 64-bit chunk handle to identify each chunk
- replicated on multiple chunk-servers (default 3)
- replication level is customizable across namespaces

Design Decisions

Chunk Size (64MB)

- advantages:
  - reduces interaction with the master!
    - reduces network overhead
      - likely to perform many OPs on a chunk
    - reduces size of the stored metadata

Design Decisions

Chunk Size (64MB)

- issues:
  - internal fragmentation within chunk
    - use lazy space allocation
- chunk-servers may become hotspots
  - use higher replication factor
  - stagger application start times
  - enable client-client communication (in future ...)

Design Decisions

Single Master

- centralization for simplicity!
- global knowledge helps in:
  - sophisticated chunk placement
  - making replication decisions
- master also maintains all FS metadata.

... but this does NOT complement with our learning from distributed systems!

Design Decisions

Single Master

solves: single-point of failure

- shadow masters: read-only access
- minimize master involvement

solves: scalability bottleneck

- data never moves through the master
- only metadata moves and is also cached
- large chunk size
- chunk leases: to delegate data mutations.

Design Decisions

No Data Caching

- clients do **NOT** cache file data.
  - working-set too large to be cached locally.
    - simplifies the client; no cache-coherence issues.
- chunk-servers do **NOT** cache file data.
  - linux buffer-cache already caches the data.

Design Decisions

FS Interface

- support for usual operations:
  - create, delete, open, close, read, write.

beyond POSIX

- snapshot operation:
  - copy a file/directory tree at low cost.
- record append operation:
  - atomically and concurrently append data to a file from multiple clients.

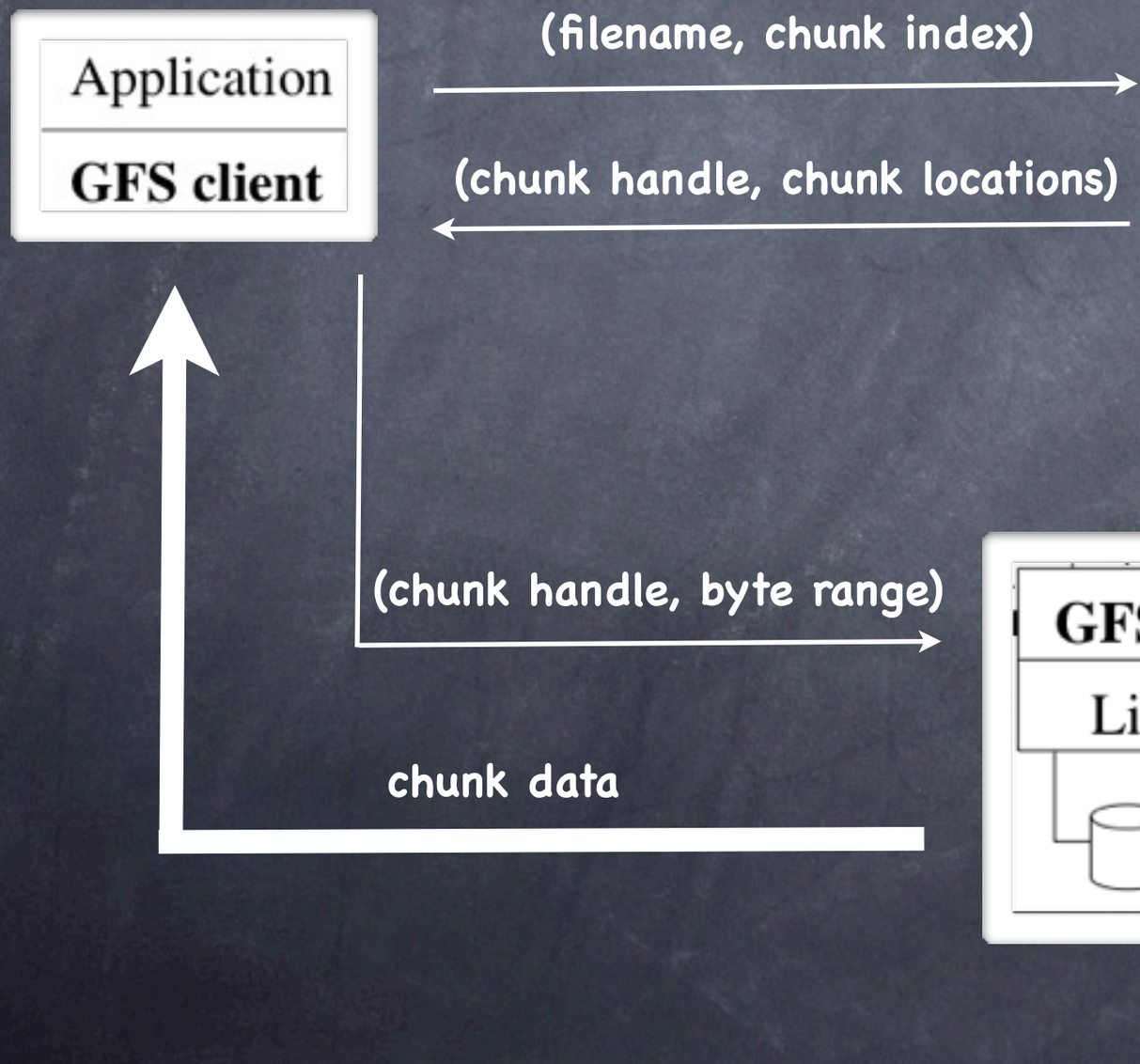
Design Overview

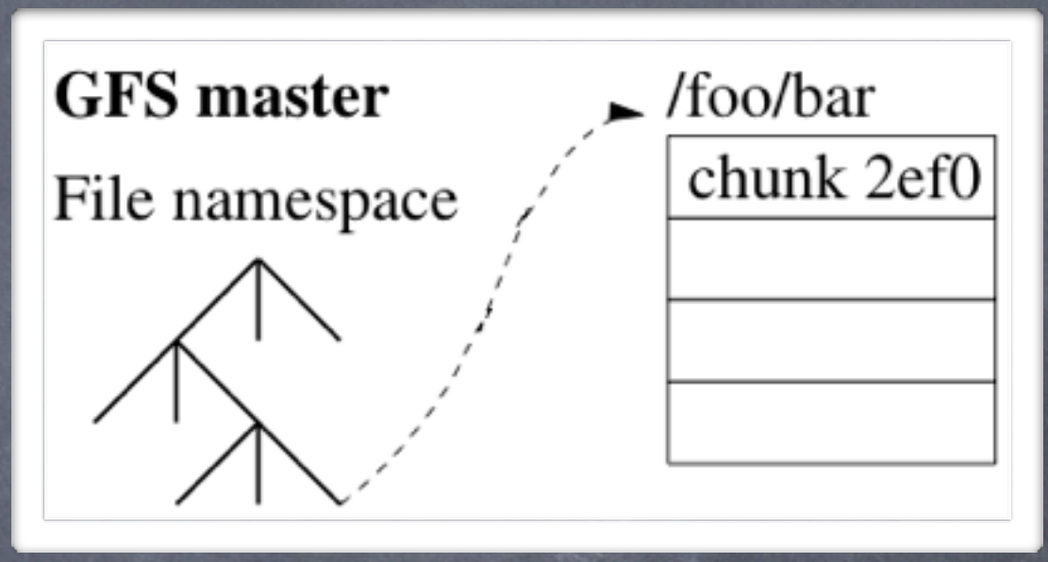
Example

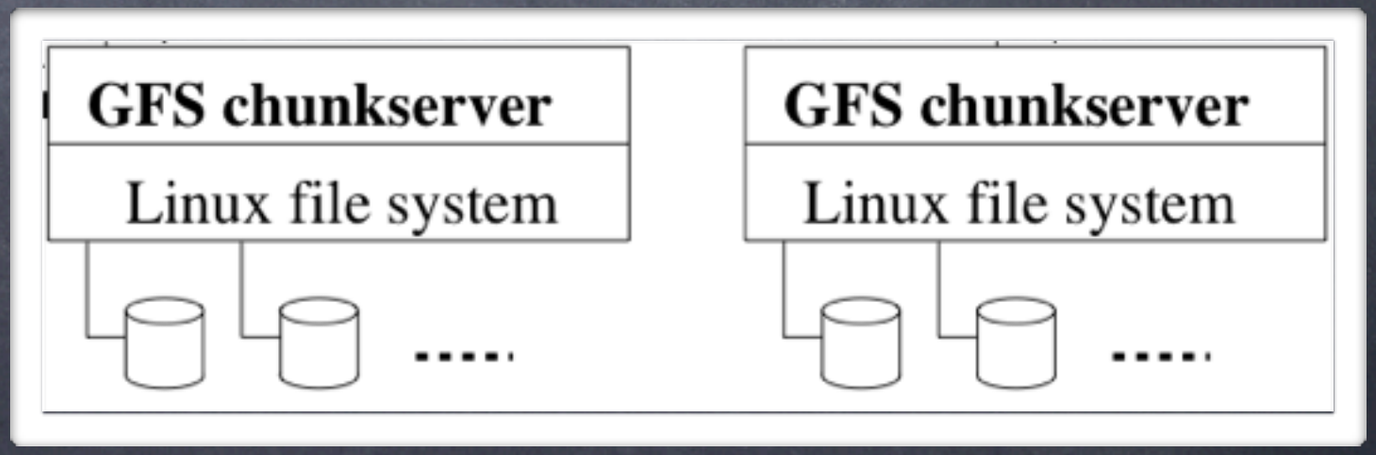
cache as key:value

key=(filename, chunk index)

value=(chunk handle, chunk locations)







Design Overview

Metadata

- master stores three types of metadata:
  - file and chunk namespaces — persistent
  - file-to-chunk mappings — persistent
  - chunk replica locations — in-memory

Design Overview | Metadata

In-Memory Data Structures

- makes the master operations FAST!
- allows efficient periodic scan of the entire state
  - for chunk garbage collection
  - for re-replication on chunk-server failures
  - for load balancing by chunk migration
- capacity of the system limited by the master's memory size?
  - NO, a 64MB chunk needs less than 64B metadata.
  - that is, 640TB data needs less than 640MB memory!

Chunk Locations

- chunk locations do NOT have a persistent record
- to know the chunk locations, the master asks:
  - each chunk-server at startup.

    or
  - whenever a chunk-server joins the cluster
- eliminates the issue of syncing master and chunk-server

Design Overview | Metadata

Operation Log

- defines the order of concurrent operations.
- replicated on multiple machines
- fast recovery using checkpoints (stored in B-trees)
- create checkpoints in separate thread

Example

set of mutations
to the master

X

single master copy of
metadata is now corrupted!

have an operation log for
things you're going to write
before you write it

START
X
Y
END

Consistency Model

- **consistent:** all clients see the same data, regardless of which replica they read from

- **defined:** consistent AND clients see what the mutation has written in its entirety

Guarantees by GFS

- after a sequence of successful mutations, the region is guaranteed to be defined
  - by mutating in same order on all replicas
  - by chunk versioning to detect stale replicas
- data corruption is detected by checksums.

Implications for Apps

- single writer appends
  - writer periodically checkpoints the writes
  - readers ONLY process region upto last checkpoint
- multiple writer appends
  - writers use write-at-least-once semantics
  - readers discard paddings using checksums
  - readers discard duplicates using unique IDs

System Interactions

- Chunk Leases and Mutation Order

consistent

- Data and Control Flow

- Special Operations
  - atomic record append
  - snapshot

defined

- Master Operations
  - namespace management and locking
  - replica placement
  - creation, re-replication, rebalancing
  - garbage collection
  - stale replica detection

System Interactions

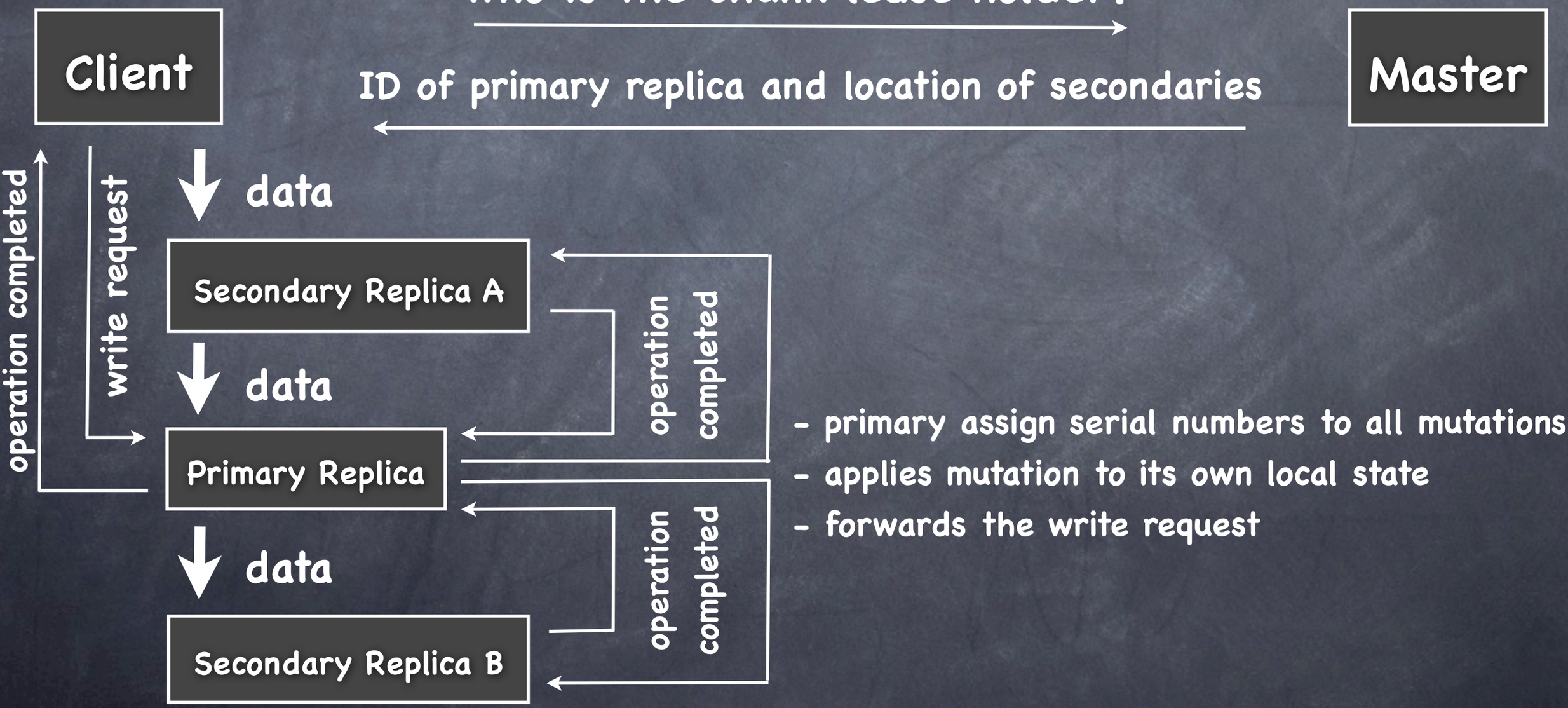
Chunk Leases

- used to minimize the master involvement.
- primary replica: one granted with a chunk lease
  - it picks a serial order of mutation.
  - with initial timeout of 60s (can be extended)
  - can be revoked by the master

System Interactions

Mutation Order

cache as key:value
key=(chunk handle)
value=(primaryID, secondaries)



- primary assign serial numbers to all mutations
- applies mutation to its own local state
- forwards the write request

Example

client A

file: foo
chunks: {1, 3, 5}

client B

file: foo
chunks: {2, 3}

changelist

primary replica

{A, foo, 1}
~~{A, foo, 3}~~
{A, foo, 5}

{B, foo, 2}
{B, foo, 3}

<- consistent ->

secondary replicas

{A, foo, 1}
~~{A, foo, 3}~~
{A, foo, 5}

{B, foo, 2}
{B, foo, 3}

System Interactions

Data and Control Flow

- decouple data flow and control flow
- control flow: client → primary → secondary
- data flow: pushed linearly along carefully picked chain of chunk-servers
  - forward to the closest first
    - distances estimated from IP addresses
  - use pipelining to exploit full-duplex links

System Interactions | Special Operations

atomic record append

- traditional concurrent writes require a distributed lock manager
- this operation ensures concurrent appends are serializable
  - client only specifies the data
    - primary lease choses a mutation order
- if a record-append fails at any replica, client retries
  - some replicas may have duplicates
  - GFS does NOT guarantee replicas are byte wise identical
  - GFS ONLY guarantees data is written at-least once atomically

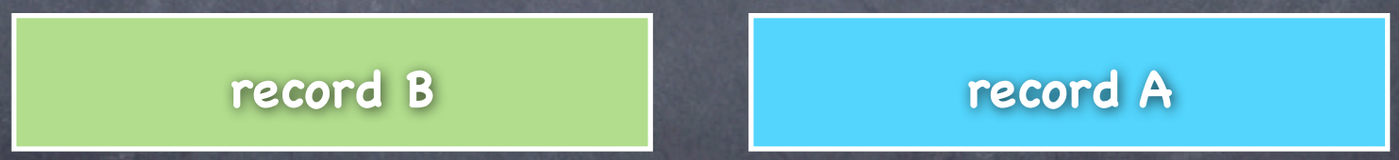
System Interactions | Special Operations

atomic record append

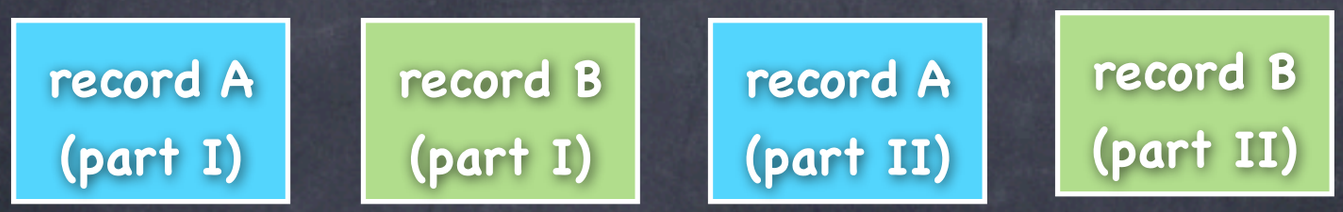
defined

- concurrent writes require a distributed lock manager
- concurrent appends to same file appear side-by-side

- does NOT guarantee order of concurrent appends



- does guarantee writes within a record are NOT interleaved



System Interactions | Special Operations

Snapshots

- makes a copy of a file/directory tree instantaneously
- uses standard copy-on-write scheme | inspired from AFS
- steps of master operation:
  - revokes outstanding leases on files
  - logs the operation to disk
  - duplicates the in-memory metadata state
  - duplicates the chunks ONLY on first write request

Master Operations

- Namespace Management and Locking
- Policies
- Garbage Collection
- State Replica Detection

Master Operation

Namespace Management

- no per-directory data structure
- no file/directory aliasing

unlike UNIX FS

- namespace representation using a lookup table
  - maps full pathnames to metadata
  - using prefix-compression for efficient storage

Master Operation

Namespace Locking

- ability to lock over regions of namespaces: to allow multiple master operations to be active at once.
- example:
  - snapshot: /home/user to /save/user
    - write lock: /save/user and /home/user
  - create: /home/user/foo
    - read lock: /home and /home/user

	read	write
read	TRUE	FALSE
write	FALSE	FALSE

- locks acquired in a consistent total order to prevent deadlocks

Master Operation

Policies | Replica Creation

- place new replicas on chunk-servers with below average disk space utilization
- limit number of recent creations on chunk-servers
- spread replicas of chunks across racks

Master Operation

Policies | Re-replication

- replicate chunks further away from their replication goal FIRST
- replicate chunks for live files FIRST
- replicate chunks blocking clients FIRST

Master Operation

Garbage Collection

- lazy reclamation
  - master logs the deletion immediately
- the storage is NOT reclaimed immediately
  - renamed to hidden name with deletion stamp
  - removed 3 days later (configurable)
  - can be read/undeleted during this time

Master Operation | Garbage Collection

Steps of Operations

- scan the FS namespace: remove hidden files
- remove file-chunk mapping from in-memory metadata
- scan the chunk namespace: identify orphaned chunks
- remove the metadata for orphaned chunks
- send message to chunk-servers to delete chunks

Advantages

- simple and reliable
  - chunk creation may fail
  - replica deletion messages maybe lost
- done in batches and cost is amortized
- ONLY done when the master is relatively free
- safety net against accidental, irreversible deletion

Disadvantages

- deletion delay hinders when storage is tight
  - different reclamation policy for separate namespaces
- cannot reuse the storage right away
  - delete twice: expedites storage reclamation

Master Operation

Stale Replica Detection

- stale replica: when chunk-server fails and misses mutation to a chunk while it's down
- chunk version numbers to detect stale replicas
- reclaimed in a regular garbage collection

Fault Tolerance and Diagnosis

- High Availability
- Data Integrity
- Diagnostic Tools

Fault Tolerance and Diagnosis

High Availability

- Fast Recovery
- Chunk Replication
- Master Replication

Fast Recovery

- designed to restore state and start in seconds
- no distinction: normal and abnormal termination

Fault Tolerance and Diagnosis | High Availability

Chunk Replication

- different replication levels for separate namespaces

- keep each chunk ALWAYS fully replicated
  - on chunk-server failure
  - on replica corruption

Master Replication

- operation log and checkpoints are replicated
- on master failure!
  - monitoring infrastructure outside GFS starts a new master process
  - shadow masters: read-only access to FS
    - enhance read availability

Fault Tolerance and Diagnosis

Data Integrity

- checksums to detect data corruption
- chunk(64MB): 64KB blocks + 32bit checksum
- checksum verification prevents error propagation
- optimized for appends: incremental checksum update
- little effect on performance:
  - normal reads span multiple blocks anyway
  - reads aligned at checksum block boundaries
  - checksum calculations can be overlapped with IO

Fault Tolerance and Diagnosis

Diagnostic Tools

- diagnostic logging on:
  - chunk-servers going up and down
  - RPC requests and replies
- helps reconstruct history and diagnose problem
- serve as traces for load testing and performance analysis
- minimal performance impact: written sequentially and asynchronously

Performance

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

lot of occupied disk space

CS metadata: well under 1%

master metadata: can fit in RAM

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

efficient usage of resources

Conclusions

- demonstrates how to support large scale processing workloads on commodity hardware
- designed to tolerate frequent h|w failures
- optimized for huge files: appends and streaming reads
- relax consistency model and extended FS interface
- simple solutions: single master are good enough

References

- Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. 2003. The Google file system. In Proceedings of the nineteenth ACM symposium on Operating systems principles (SOSP '03). ACM, New York, NY, USA, 29-43.