The Google File System

Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung
Google

Vijay Kumar

Outline

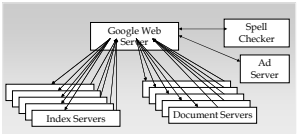
- Introduction
- Goals
- Architecture
- Operations Supported
- Master Operations
- Performance
- Conclusion

File System for Google

- A new distributed file system developed to meet the demands of Google's application workloads and technological environment
- Shares many of the same goals as the other distributed file systems

Technological environment

- Clusters of more than 15,000 commodity class PCs
- Hardware prone to failure
- Fault tolerant software



Source: "Web Search for a Planet: The Google Cluster Architecture"

Application Workloads

- Sequential reads
  - Indexer reading the contents of web pages
- Frequent Appends
  - Crawler appending new pages
- Files used as Producer – Consumer queues
  - Indexer waits for the crawler to retrieve contents
- Files used for multi-way merging

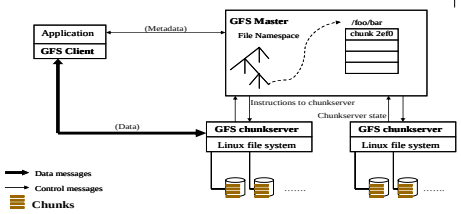
GFS – Motivation

- Component failures – norm rather than exception
- Efficient management of large files
- Optimization of frequently performed operations
- Flexibility of co-designing application and file system

GFS - Goals

- Reliability, availability, scalability…
- Tolerance to hardware failures
- Managing numerous files of large size
- Optimizing commonly performed operations

GFS Architecture



- Single master
- Multiple chunkservers
- Multiple clients
- Master stores metadata
- Chunkservers store data
- Communication via *heartbeat* messages

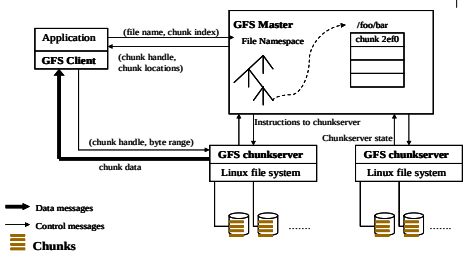
GFS Architecture (contd)

- Files are broken into fixed sized chunks
- Chunks are identified by unique chunk handles
- Chunks are stored in chunkservers
- Fixed chunk size – 64 MB

GFS Operations

- File open, close …
- Data reads
- Data Mutations
  - Random writes
  - Record appends
- Snapshot

Data Read



Source: Uma Murthy's presentation on GFS, Fall '04

Data Mutations

- Two kinds of data mutations are supported
  - Random writes
  - Record appends
- Record appends much more frequent than random writes
- Concurrent appends are common
- Leases used to maintain consistent mutation order

Data Mutations (contd)

- Relaxed consistency model to support applications
- Relatively simple and efficient to implement
- GFS guarantees
  - Atomic file namespace mutations
  - State of a file region after a data mutation depends on type of mutation, success or failure, presence or absence of concurrent mutations

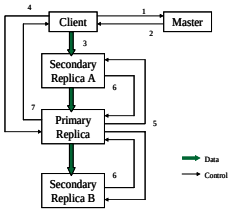
Data Mutations (contd)

	Write	Record Append
Serial success	*defined*	*defined* interspersed with *inconsistent*
Concurrent successes	*consistent* but *undefined*	
Failure	*inconsistent*	

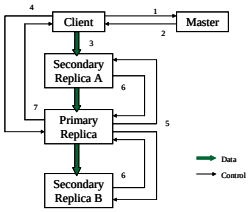
- Consistent file region – All client will see the same data, regardless of the replicas they read from
- Defined file region – It is consistent and clients see what mutations writes in its entirety
- Applications have to deal with the relaxed consistency

Data Writes

1. Client requests for lease holder and secondary replicas
2. Master responds
3. Client pushes out data to the replicas
4. Client issues write instruction to primary replica



Data Writes (contd)



5. Primary forwards write request to other replicas
6. Secondaries respond to primary reporting the status of the write
7. Primary reports the status to the client

Record Appends

- Appends record to the file atomically (i.e., as a continuous sequence of bytes)
- Appended at an offset chosen by primary
- Pads chunks if append is expected to cross the boundary. Secondaries also do so. New chunk has to be allocated in this case
- Offset returned to the client on success
- Client retries on failure

Snapshot

- Makes a copy of a file or directory tree
- Based on copy-on-write technique
- Minimal overhead involved

Master Operation

- Master is responsible for
  - Metadata management
  - Namespace management
  - Replica management
  - Garbage Collection

Metadata Management

- Master stores the following metadata
  - File and chunk namespaces – stored persistently
  - File to chunk mapping – stored persistently
  - Chunk replica locations – queries chunkserver
- Operation Log – historic record of metadata changes
- Replicated on multiple remote machines
- Size kept small by checkpointing

Name Space Management

- Does not have a per directory data structure
- No support for aliases
- Namespace represented as lookup
- Files and directories have associated *read* and *write* locks

Replica Management

- Chunk replicas created for
  - New chunk creation
  - Chunk re-replication
  - Chunk rebalancing
- Chunk replicas placement based on
  - Maximizing data reliability and availability
  - Maximizing network utilization
- Chunkserver's *disk utilization*, *count of recent chunks*, *rack position* affect desicions

Garbage Collection

- Reclamation of physical storage
  - Files renamed to hidden names upon deletion
  - File metadata deleted during namespace scan
  - Physical storage freed during exchange of *Heart Beat* messages
- Stale Replica Detection
  - Stale chunks detected using *chunk version number*

GFS Goals revisited

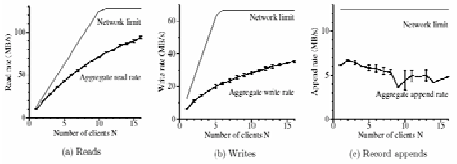
- Availability
  - Fast recovery
  - Chunk replication
  - Master replication
- Scalability
  - Master replication
  - Keeping master's involvement limited in data transactions

GFS Goals revisited (contd)

- Fault Tolerance
  - Replication, constant monitoring, fast recovery
- Data Integrity
  - Checksumming used to detect corruption
- Optimization for frequent operations
  - Relaxed consistency model

Performance

- Micro benchmarks
  - Aggregate read rate – 75% of theoretical limit
  - Aggregate write rate – 50% of theoretical limit



(a) Reads (b) Writes (c) Record appends

Performance (contd)

- Real world 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	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

Performance (contd)

- Workload breakdown

Operation	Read		Write		Record Append	
Cluster	X	Y	X	Y	X	Y
0K	0.4	2.6	0	0	0	0
1B..1K	0.1	4.1	6.6	4.9	0.2	9.2
1K..8K	65.2	38.5	0.4	1.0	18.9	15.2
8K..64K	29.9	45.1	17.8	43.0	78.0	2.8
64K..128K	0.1	0.7	2.3	1.9	< .1	4.3
128K..256K	0.2	0.3	31.6	0.4	< .1	10.6
256K..512K	0.1	0.1	4.2	7.7	< .1	31.2
512K..1M	3.9	6.9	35.5	28.7	2.2	25.5
1M..inf	0.1	1.8	1.5	12.3	0.7	2.2

Cluster	X	Y
Open	26.1	16.3
Delete	0.7	1.5
FindLocation	64.3	65.8
FindLeaseHolder	7.8	13.4
FindMatchingFiles	0.6	2.2
All other combined	0.5	0.8

Conclusion

- Demonstrates qualities needed to support large scale data processing workloads on commodity hardware
- Delivers high throughput
- Successfully meets Google's storage needs