The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung

Presented By: Noshin Nawar Sadat

15-Jan-19

UNIVERSITY OF WATERLOO

Google File System (GFS)

- A scalable distributed file system for large distributed data-intensive applications
- Provides fault tolerance while running on inexpensive commodity hardware
- Delivers high aggregate performance to a large number of clients.
- Widely deployed within Google as the storage platform

Assumptions

- Monitor, detect, tolerate, and recover promptly from component failures on a routine basis
- Stored files are mostly large (100 MB or larger)
- Large streaming reads and small random reads
- Mostly large, sequential writes that append data to files
- Efficiently handle multiple clients that concurrently append to the same file
  - Atomicity with minimal synchronization overhead is essential.
- High sustained bandwidth is more important than low latency

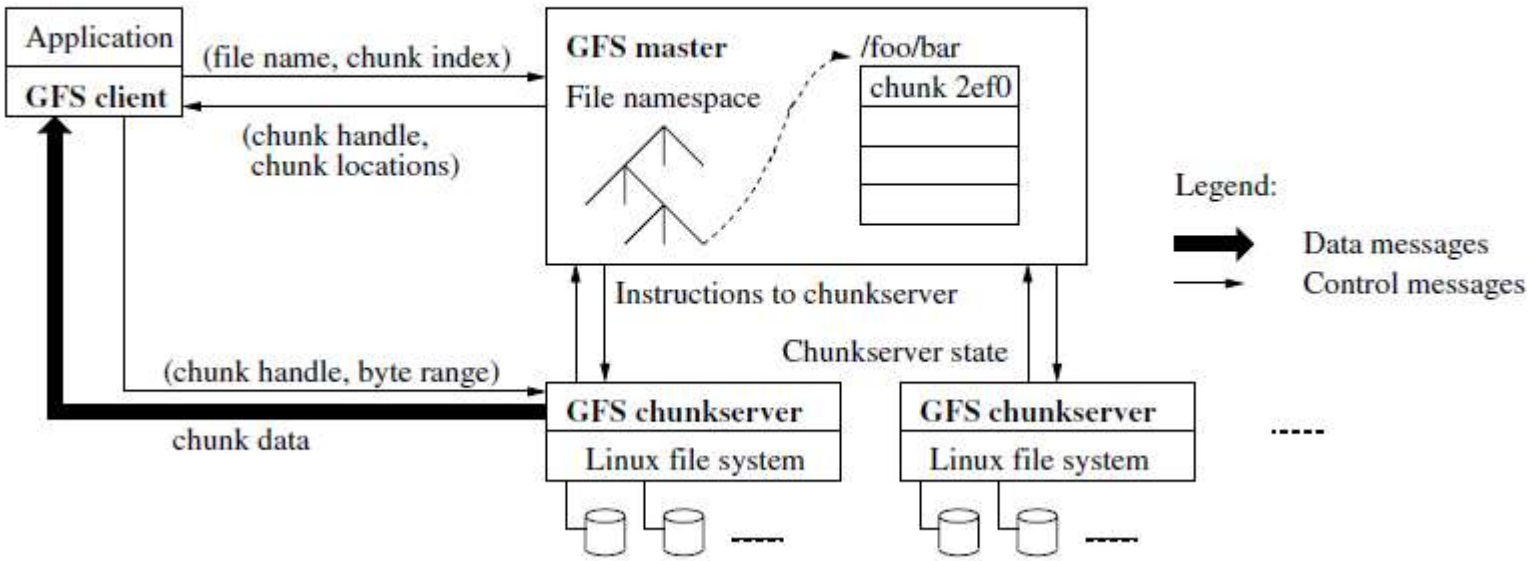
Interface

- Provides a familiar file system interface
  - Does not implement a standard API
- File organization
  - Hierarchically in directories
  - Identified by path-names
- Operations
  - *create*, *delete*, *open*, *close*, *read*, and *write* files
  - *snapshot*
  - *record append*

Architecture

- A GFS cluster consists of
  - a single *master*
  - multiple *chunkservers*
  - accessed by multiple *clients*
- Files divided into fixed-size *chunks (64 MB)*
  - identified by 64 bit *chunk handle* assigned by the master at the time of *chunk* creation
  - replicated on multiple *chunkservers* to ensure reliability (3 replicas, by default)

Architecture



GFS Master

- Maintains all file system metadata in main memory
  - capacity of whole system limited by memory
- Periodically communicates with each *chunkserver* through *HeartBeat* messages
- Makes sophisticated *chunk* placement and replication decisions using global knowledge

Chunk Size (64 MB)

- Advantages
  - reduces *clients*' need to interact with the *master*
  - reduces network overhead by keeping a persistent TCP connection to the *chunkserver* over an extended period of time
  - reduces the size of the metadata stored on the *master*
- Disadvantages
  - A small file may lead to creation of *hotspots*

Metadata

- Three major types of metadata
  - the file and *chunk* namespaces (persistently stored)
  - the mapping from files to *chunks* (persistently stored)
  - the locations of each *chunk's* replicas (not persistently stored)
    - poll *chunkservers* at startup and monitor *Heartbeat* messages
- *Operation log*
  - logs of mutations to keep metadata persistently
  - stored on the *master's* local disk
  - replicated on remote machines
  - allows to update the *master* in the event of a *master* crash

Consistency Model

- Atomic file namespace mutations
  - handled exclusively by the *master*
- State of a file region after a data mutation depends on
  - the type of mutation
  - whether it succeeds or fails
  - whether there are concurrent mutations

Consistency Model

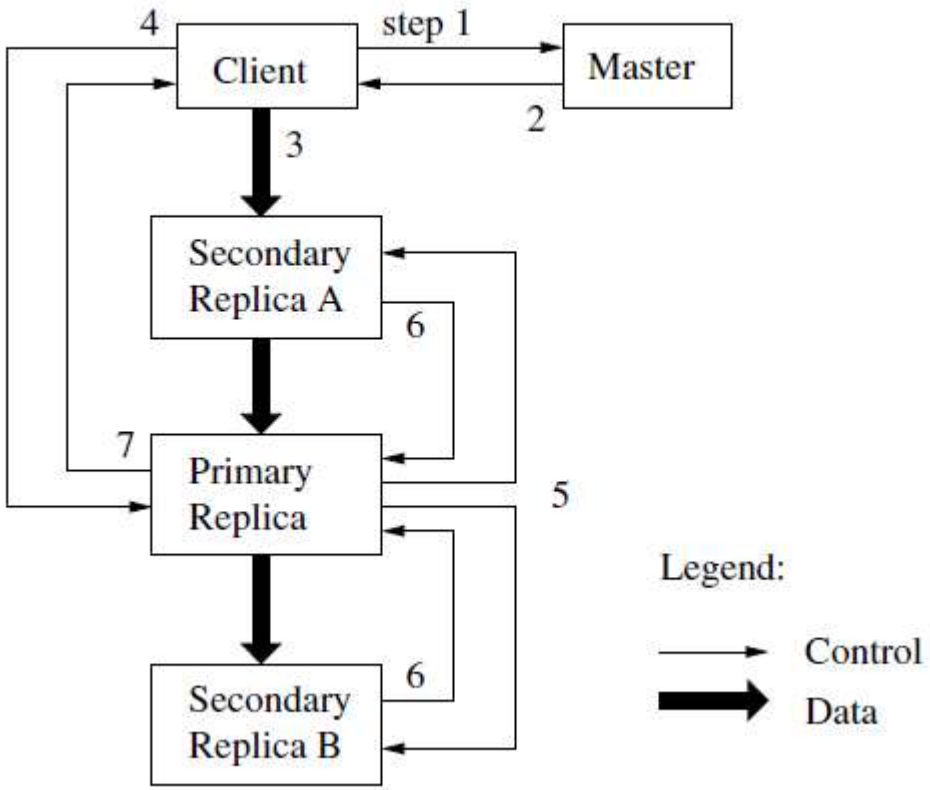
	Write	Record Append
Serial success	*defined*	*defined* interspersed with *inconsistent*
Concurrent successes	*consistent* but *undefined*	
Failure	*inconsistent*	

Table 1: File Region State After Mutation

consistent - all *clients* will always see the same data, regardless of which replicas they read from

defined - after a file data mutation, it is consistent and *clients* will see what the mutation writes in its entirety

Leases and Mutation Orders



UNIVERSITY OF WATERLOO

Why Separate Data Flow?

- To fully utilize each machine's network bandwidth
  - data is pushed linearly along a chain of *chunkservers* rather than distributed in some other topology
- To avoid network bottlenecks and high-latency links
  - each machine forwards the data to the "closest" machine in the network topology that has not received it
  - "distances" can be accurately estimated from IP addresses.
- To minimize latency
  - pipelining the data transfer over TCP connections

Atomic Record Append

- Same control flow as *write*
- Process
  - *client* pushes the data to all replicas of the last *chunk* of the file and sends request to *primary*
  - if *primary* finds *chunk* size > 64 MB after appending the record to current *chunk*
    - pads the *chunk* to the maximum size
    - tells *secondaries* to do the same
    - asks *client* to retry operation on the next *chunk*
  - else
    - appends the data to its replica
    - tells the *secondaries* to write the data at the exact offset where it has
    - replies success to the *client*

Master Operations

Namespace Management and Locking

- Logically represents its namespace as a lookup table mapping full pathnames to metadata
- Each node in the namespace tree has an associated read-write lock
- Allows concurrent mutations in the same directory
  - each operation acquires a read lock on the directory name and a write lock on the file name

Master Operations

Chunk Creation

- Chooses where to place the initially empty replicas
- Considers several factors
  - place new replicas on *chunkservers* with below-average disk space utilization.
  - limit the number of "recent" creations on each *chunkserver*
  - spread replicas of a *chunk* across racks

Master Operations

Chunk Re-replication

- Prioritized based on
  - how far it is from its replication goal
  - live files vs. recently deleted files
  - boost the priority of any *chunk* that is blocking *client* progress

Master Operations

Re-balancing Chunk Replicas

- Examines the current replica distribution
- Moves replicas for better disk space and load balancing
- Chooses which existing replica to remove

Master Operations

Garbage Collection

When a file is deleted by the application

- *master* logs the deletion immediately
- the file is renamed to a hidden name that includes the deletion timestamp
- *master's* regular scan of the file system namespace: removes any hidden files that existed for more than three days, severing its links to all its *chunks*
- *master's* regular scan of the *chunk* namespace: identifies orphaned *chunks* and erases the metadata for those *chunks*
- *master* replies to *Heartbeat* messages of *chunkservers* with the identities of absent *chunks*
- *chunkserver* is free to delete its replicas of such *chunks*

Master Operations

Stale Replica Detection

- Stale replicas
  - when *chunkserver* fails and misses mutations to the *chunk* while it is down
  - removed during regular garbage collection
- *Chunk version number* to distinguish between up-to-date and stale replicas
  - whenever the *master* grants a new lease on a *chunk*, it increases the *chunk version number* and informs the up-to-date replicas

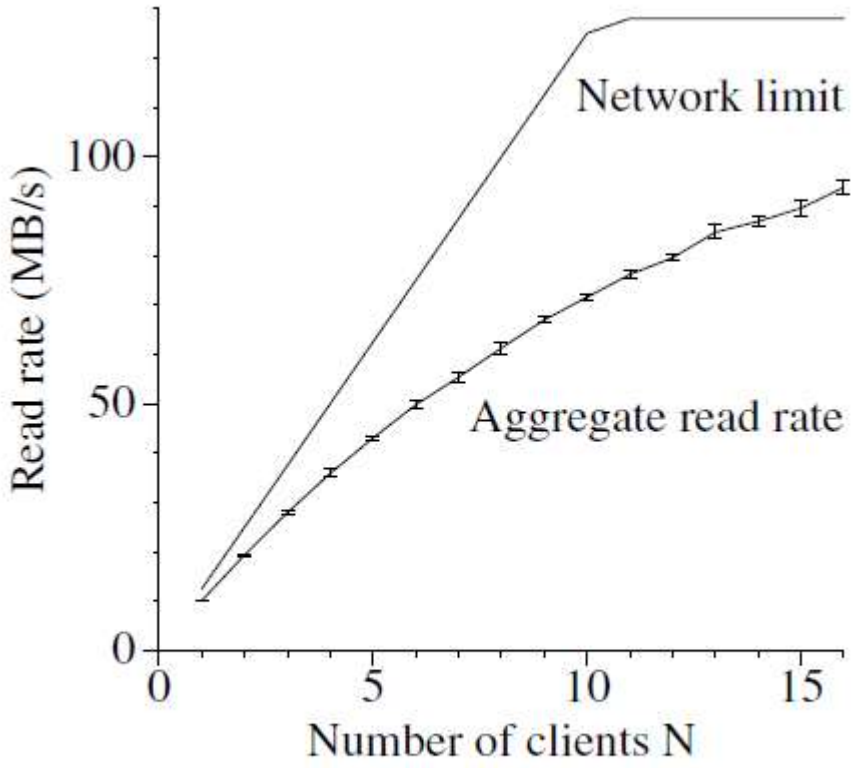
Fault Tolerance and Diagnosis

- High availability
  - Fast recovery
  - Chunk replication
  - Master replication
- Data Integrity
  - Checksum to detect corrupted data
- Diagnostic Tools
  - Extensive and detailed diagnostic logging

Performance Measurement

- A GFS cluster consisting of
  - one master, two master replicas, 16 chunkservers, 16 clients
- Machine configuration
  - dual 1.4 GHz PIII processors
  - 2 GB of memory
  - two 80 GB 5400 rpm disks
  - 100 Mbps full-duplex Ethernet connection to an HP 2524 switch
- Connections
  - all 19 GFS server machines are connected to one switch
  - all 16 client machines to the other
  - two switches are connected with a 1 Gbps link.

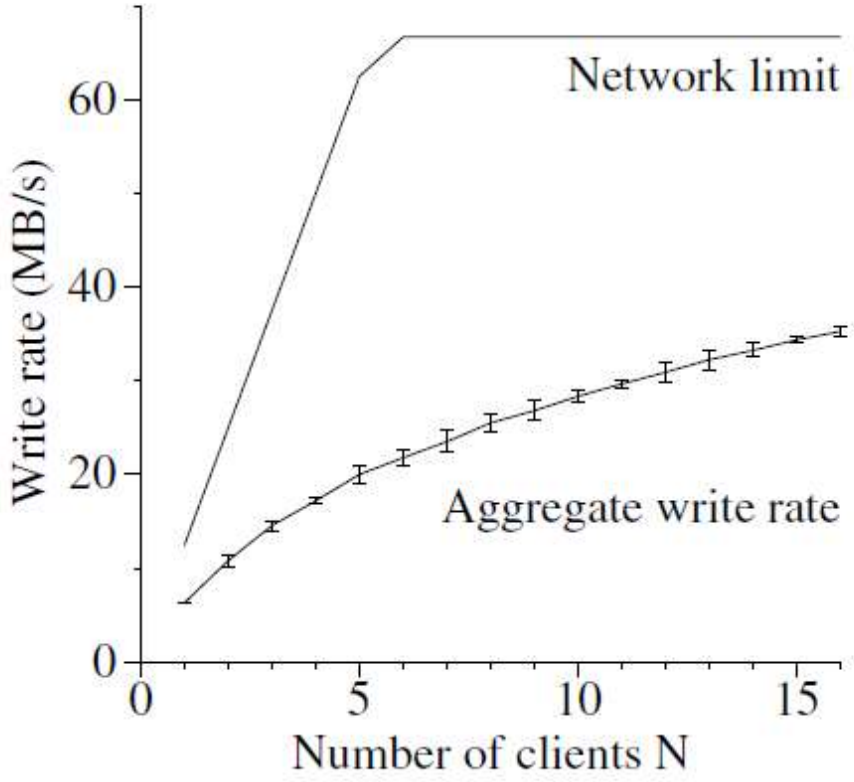
Performance Measurement



(a) Reads

UNIVERSITY OF WATERLOO

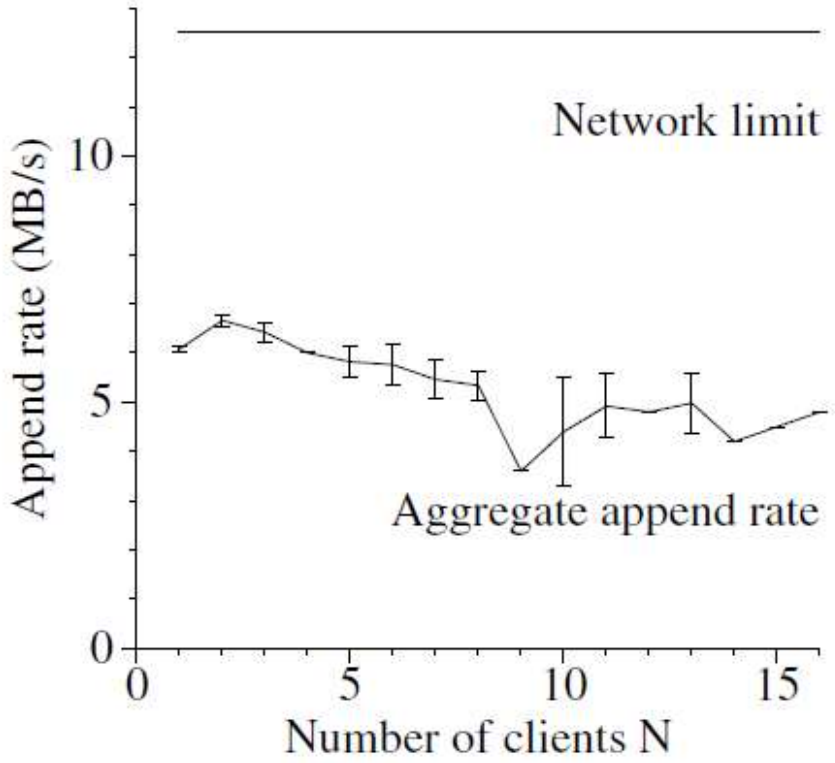
Performance Measurement



(b) Writes

UNIVERSITY OF WATERLOO

Performance Measurement



(c) Record appends

UNIVERSITY OF WATERLOO

Performance Measurement

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

Table 2: Characteristics of two GFS clusters

Performance Measurement

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

Table 3: Performance Metrics for Two GFS Clusters

Performance Measurement

Recovery Time

- Killed 1 *chunkserver*
  - 15,000 *chunks* (600GB data)
  - limited cloning operations to 40% *chunkservers* at 6.25 MBps
  - restored within 23.2 minutes at replication rate 440MBps
- Killed 2 *chunkservers*
  - 16,000 *chunks* (660 GB data) each
  - 266 *chunks* had single replica
  - restored to 2x replication in 2 minutes at high priority

Performance Measurement

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

Table 4: Operations Breakdown by Size (%)

Performance Measurement

Operation	Read		Write		Record Append	
Cluster	X	Y	X	Y	X	Y
1B..1K	< .1	< .1	< .1	< .1	< .1	< .1
1K..8K	13.8	3.9	< .1	< .1	< .1	0.1
8K..64K	11.4	9.3	2.4	5.9	2.3	0.3
64K..128K	0.3	0.7	0.3	0.3	22.7	1.2
128K..256K	0.8	0.6	16.5	0.2	< .1	5.8
256K..512K	1.4	0.3	3.4	7.7	< .1	38.4
512K..1M	65.9	55.1	74.1	58.0	.1	46.8
1M..inf	6.4	30.1	3.3	28.0	53.9	7.4

Table 5: Bytes Transferred Breakdown by Operation Size (%)

Performance Measurement

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

Table 6: Master Requests Breakdown by Type (%)

Benefits

- Centralized *master* server
  - simplified design - less complexity, greater flexibility
  - well-informed *chunk* placement and replication decisions
- Fault tolerance
  - *master* state small and fully replicated
- Scalability, high availability
  - use of shadow *masters*
- Tackle processing needs with existing cheap hardware
- High throughput
  - separation of control and data flow
- Allows for concurrent *appends*

Issues

- Applications have to deal with duplicates in the *chunks* (result of *record appends*)
- Problem delivering aggregate performance to a large number of *clients*
- System size limited by *master* server's main memory capacity
- File sizes < 64MB

Worked fine 15 years ago.

What about now?