The Google File System

Sanjay Ghemawat, howard Gobioff, and Shun-Tak Leung

Presented by Hongyu Gao
Northwestern University

Outline

- Motivation
- Key observations/assumptions
- Design overview
- System interactions
- Master operation
- Fault tolerance and diagnosis
- Experimental results
- Conclusions

Motivation

- To build a distributed file system above a cluster of cheap machines.
- The system guarantees:
  - Performance
  - Scalability
  - Reliability
  - Availability

Key observations/assumptions

- Component failures are normal
- Large files are common case
- Most files are mutated by appending new data
- Co-designing the applications and the file system API benefits the overall system

Key observations/assumptions, cont'd

- The workloads primarily consist of two kinds of reads: large streaming reads and small random reads
- High sustained bandwidth is more important than low latency

Design overview

- Interface
  - Files are organized hierarchically in directories and identified by pathnames
  - Support operations of *create*, *delete*, *open*, *close*, *read*, *write*, *snapshot* and *record append*

Design overview, cont'd

- Architecture
  - A single master to make control decisions
  - Multiple chunkservers to store data
  - Multiple clients to access the system
  - Files are divided into fixed-size chunks
  - Each chunk is replicated on multiple chunkservers (reliability/availability)

Design overview, cont'd

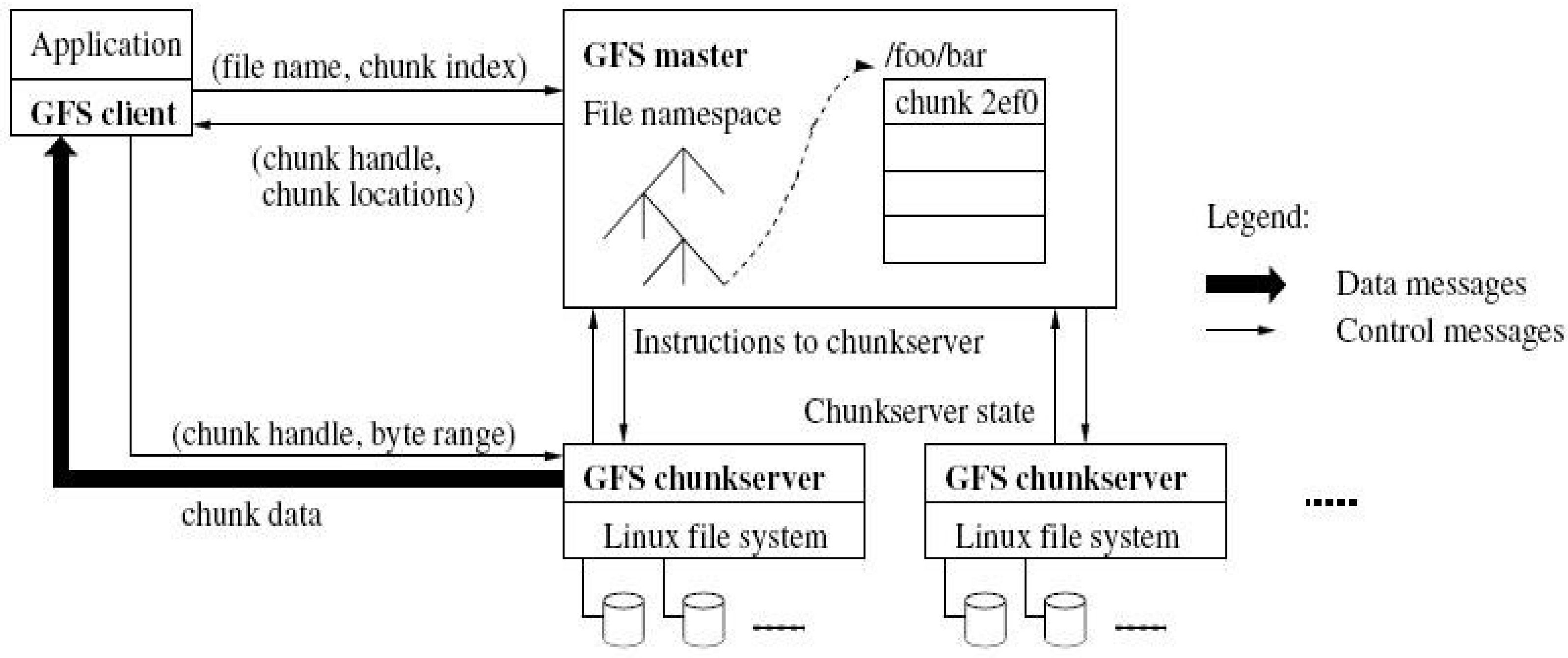


Figure 1: GFS Architecture

Design overview, cont’d

- An illustrative example

Design overview, cont’d

- Other considerations:
  - 64MB chunk size (large files are common)
  - Metadata:
    - File and chunk namespace
    - The file-to-chunk mapping
    - The locations of each chunk’s replicas
  - Operation log

Design overview, cont'd

- Consistency model

<table>
<tr><th></th><th>Write</th><th>Record Append</th></tr>
<tr><td>Serial success</td><td>defined</td><td rowspan="2">defined interspersed with inconsistent</td></tr>
<tr><td>Concurrent successes</td><td>consistent but undefined</td></tr>
<tr><td>Failure</td><td colspan="2">inconsistent</td></tr>
</table>

Table 1: File Region State After Mutation

System interactions

- Leases and Mutation Order

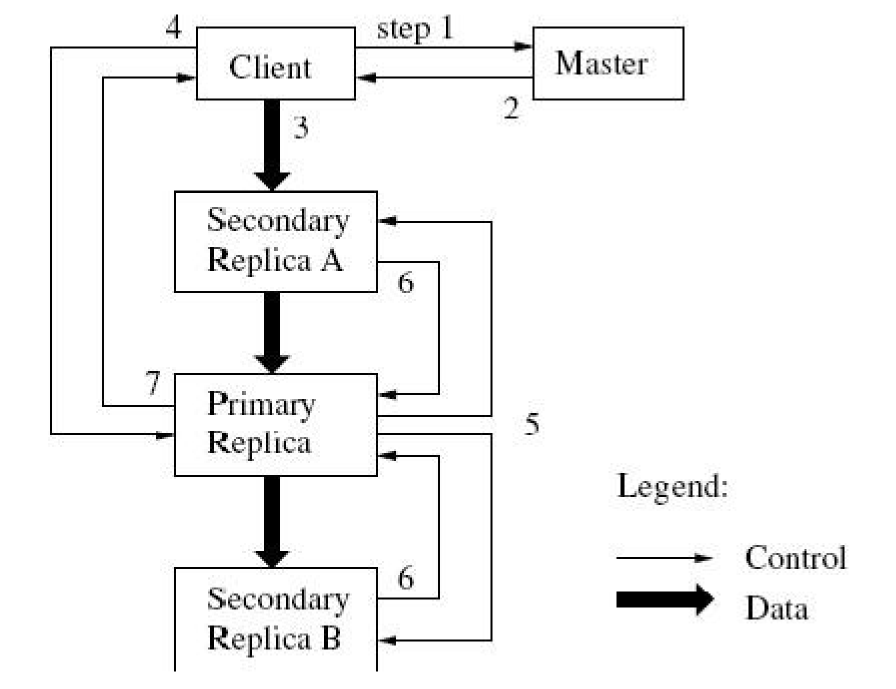


Figure 2: Write Control and Data Flow

System interactions, cont'd

- Control flow
  - Client-master, then from the client to the primary and then to all secondaries
- Data flow
  - Linear along a carefully picked chain of chunkservers in a pipelined fashion

System interactoin, cont'd

- Atomic record appends
  - Guarantees that the data is written at least once as an atomic unit
- Snapshot
  - First revoke all leases on the chunks about to snapshot
  - Duplicate metadata
  - Copy-on-write technique

Master operation

- Namespace management and locking
  - Each node in the namespace tree has an associated read-write lock
  - Each master operation on namespace acqures a set of locks before it runs
  - Locks are acquired in order
  - Example: /d1/d2/…/dn/leaf

    /d1, /d1/d2, …, /d1/d2/…/dn, read lock

    /d1/d2/…/dn/leaf, write lock

Master operation, cont'd

- Replica creation
  - Place replica on chunkservers with low disk space utilization
  - Limit the number of recent creation on each chunkserver
  - Spread replicas of a chunk across racks

Master operation, cont’d

- Re-replication
  - How far is it from its replication goal
  - Is it a chunk for live file
  - Is it blocking client progress
- Rebalancing

Master operation, cont’d

- Garbage collection
  - The master renames a deleted file with a hidden name with timestamp
  - The master deletes metadata after predefined time interval
  - The chunkservers delete orphaned chunks
  - Simple, reliable and do not generate additional network traffic

Master operation, cont'd

- Stale replica detection
  - Use chunk version number
  - The chunk replica with less advanced version number is stale
  - The higher version number is considered up-to-date

Fault tolerance and diagnosis

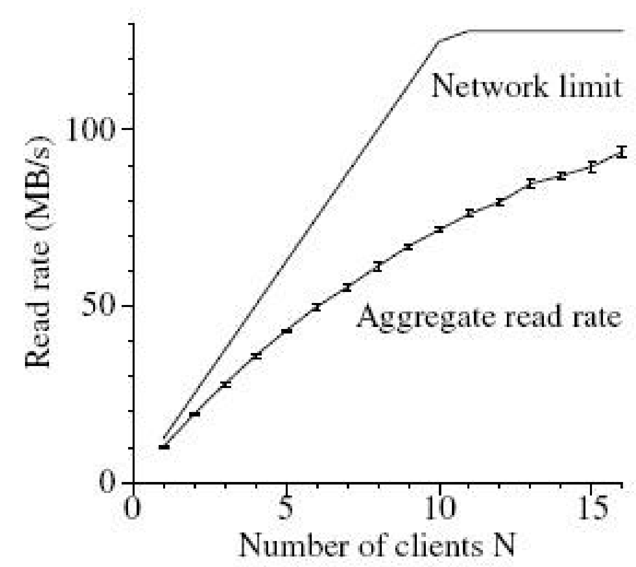
- High availability
  - Both master and chunkservers can do fast recovery
  - Both master and chunks have multiple replicas
  - Shadow masters provide read-only access to the file system when the primary master is down

Fault tolerance and diagnosis, cont'd

- Data integrity
  - Each chunkserver use checksumming to detect corruption of stored data.
- Diagnostic tools
  - Use logs

Experimental results

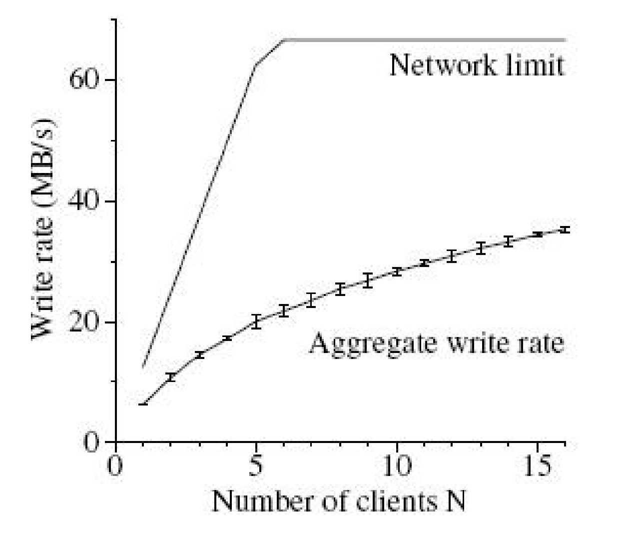
- Reads
  - Clients simultaneously reads a random 4MB region from a 320 G file set
  - Reach up to 80% of theoretical limit



(a) Reads

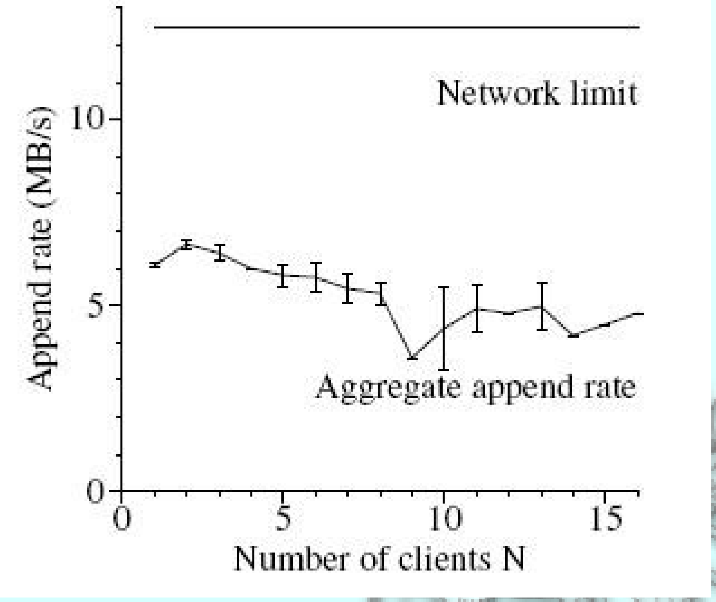
Experimental results, cont'd

- Writes
  - Clients simultaneously write to distinct files.
  - Each client writes 1GB data to a new file in a series of 1 MB writes
  - Reach up to half of theoretical limit



Experimental results, cont'd

- Record appends
  - Clients append simultaneously to a single file.
  - Performance starts a MB/s and drops to 4.8 MB/s.



Experimental results, cont'd

- Real world clusters
  - Cluster A for research and development
  - Cluster B for production data processing

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

Experimental results, cont'd

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

Master load

Not a problem

Experimental results, cont'd

- Recovery time
  - A single chunkserver restores in 23.2 minutes
  - When 2 chunkservers are killed, chunks restore to at least 2x replications in 2 minutes

Experimental results, cont'd

- Chunkserver workloads

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 (%). For reads, the size is the amount of data actually read and transferred, rather than the amount requested.

Experimental results, cont’d

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 (%). For reads, the size is the amount of data actually read and transferred, rather than the amount requested. The two may differ if the read attempts to read beyond end of file, which by design is not uncommon in our workloads.

- Operation on large files should be optimized

Experimental results, cont'd

- Master workload

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 (%)

Conclusions

- One sentence summary
  - The authors propose a mechanism to build a distributed file system above a cluster of cheap machines, and specially tune the design to the actually applications running in google
- Major flaws
  - The system do not guarantee performance on applications other than those running in google
  - Diagnostic tools are kind of weak. When the system scales, it will be prohibitively expensive to diagnose by looking into logs

Any questions?

The end

Thank you!