The Google File System

By Sanjay Ghemawat, Howard Gobioff, and
Shun-Tak Leung
(Presented at SOSP 2003)

Introduction

- Google – search engine.
- Applications process lots of data.
- Need good file system.
- Solution: Google File System (GFS).

Motivational Facts

- More than 15,000 commodity-class PC's.
- Multiple clusters distributed worldwide.
- Thousands of queries served per second.
- One query reads 100's of MB of data.
- One query consumes 10's of billions of CPU cycles.
- Google stores dozens of copies of the entire Web!

Conclusion: Need large, distributed, highly fault-tolerant file system.

Topics

- Design Motivations
- Architecture
- Read/Write/Record Append
- Fault-Tolerance
- Performance Results

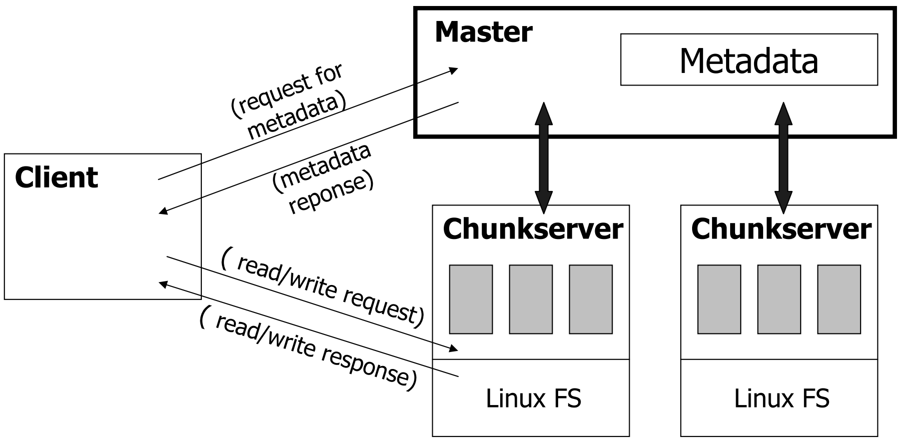
Design Motivations

1. Fault-tolerance and auto-recovery need to be built into the system.
2. Standard I/O assumptions (e.g. block size) have to be re-examined.
3. Record appends are the prevalent form of writing.
4. Google applications and GFS should be co-designed.

GFS Architecture *(Analogy)*

- On a single-machine FS:
  - An upper layer maintains the metadata.
  - A lower layer (i.e. disk) stores the data in units called "blocks".
  - Upper layer store
- In the GFS:
  - A master process maintains the metadata.
  - A lower layer (i.e. a set of chunkservers) stores the data in units called "chunks".

GFS Architecture



GFS Architecture

What is a chunk?

- Analogous to block, except larger.
- Size: 64 MB!
- Stored on chunkserver as file
- Chunk handle (~ chunk file name) used to reference chunk.
- Chunk replicated across multiple chunkservers
- Note: There are hundreds of chunkservers in a GFS cluster distributed over multiple racks.

GFS Architecture

What is a master?

- A single process running on a separate machine.
- Stores all metadata:
  - File namespace
  - File to chunk mappings
  - Chunk location information
  - Access control information
  - Chunk version numbers
  - Etc.

GFS Architecture

Master <-> Chunkserver Communication:

- Master and chunkserver communicate regularly to obtain state:
  - Is chunkserver down?
  - Are there disk failures on chunkserver?
  - Are any replicas corrupted?
  - Which chunk replicas does chunkserver store?
- Master sends instructions to chunkserver:
  - Delete existing chunk.
  - Create new chunk.

GFS Architecture

Serving Requests:

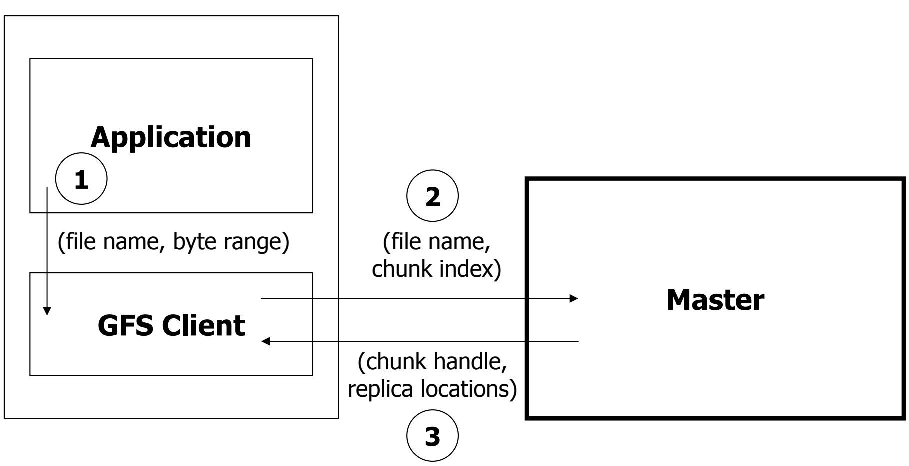
- Client retrieves metadata for operation from master.
- Read/Write data flows between client and chunkserver.
- Single master is not bottleneck, because its involvement with read/write operations is minimized.

Overview

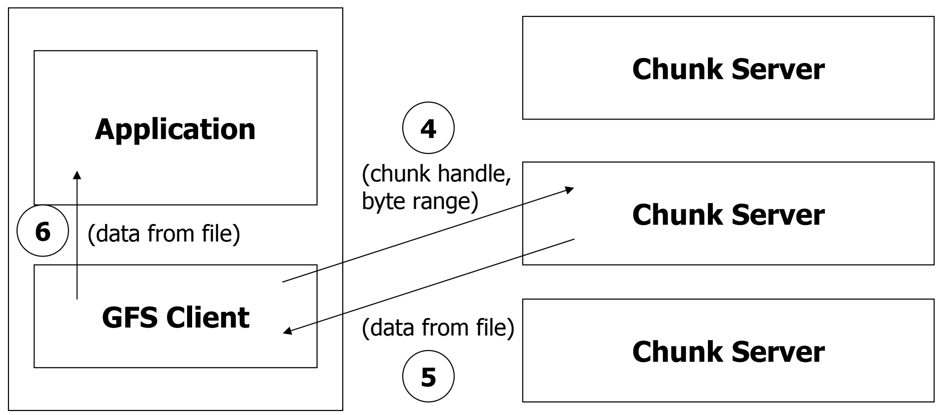
- **Design Motivations**
- **Architecture**
  - **Master**
  - **Chunkservers**
  - **Clients**
- Read/Write/Record Append
- Fault-Tolerance
- Performance Results

And now for the Meat…

Read Algorithm



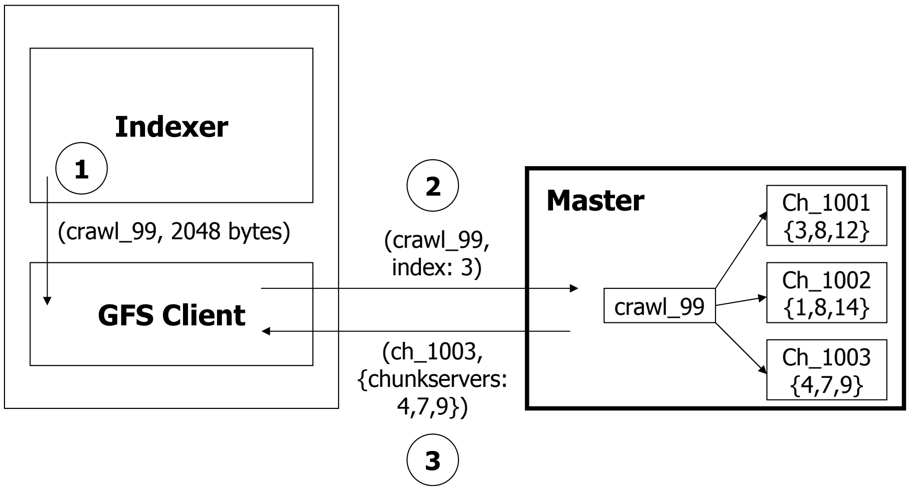
Read Algorithm



Read Algorithm

1. Application originates the read request.
2. GFS client translates the request from (filename, byte range) -> (filename, chunk index), and sends it to master.
3. Master responds with chunk handle and replica locations (i.e. chunkservers where the replicas are stored).
4. Client picks a location and sends the (chunk handle, byte range) request to that location.
5. Chunkserver sends requested data to the client.
6. Client forwards the data to the application.

Read Algorithm (*Example)*



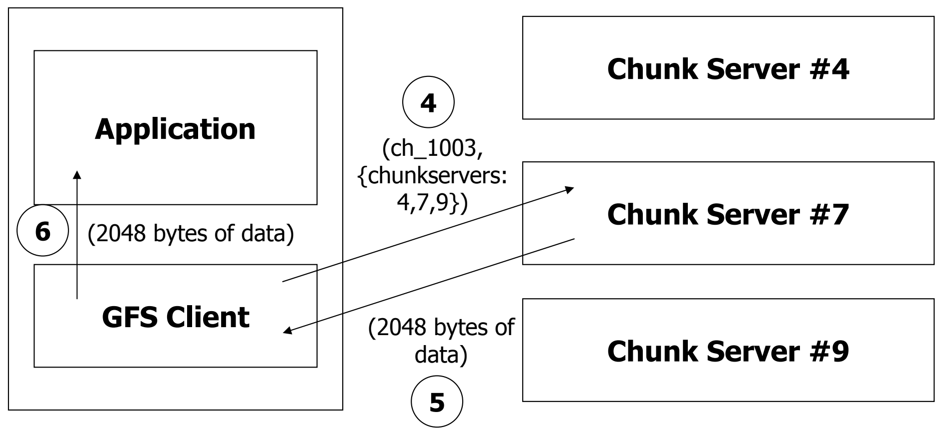
Read Algorithm (*Example)*

Calculating chunk index from byte range:

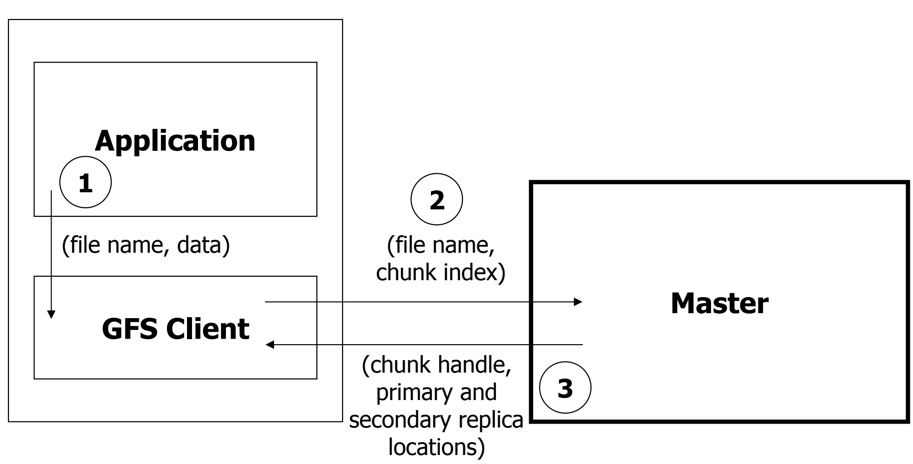
(Assumption: File position is 201,359,161 bytes)

- Chunk size = 64 MB.
- 64 MB = 1024 *1024 * 64 bytes = 67,108,864 bytes.
- 201,359,161 bytes = 67,108,864 * 2 + 32,569 bytes.
- So, client translates 2048 byte range -> chunk index 3.

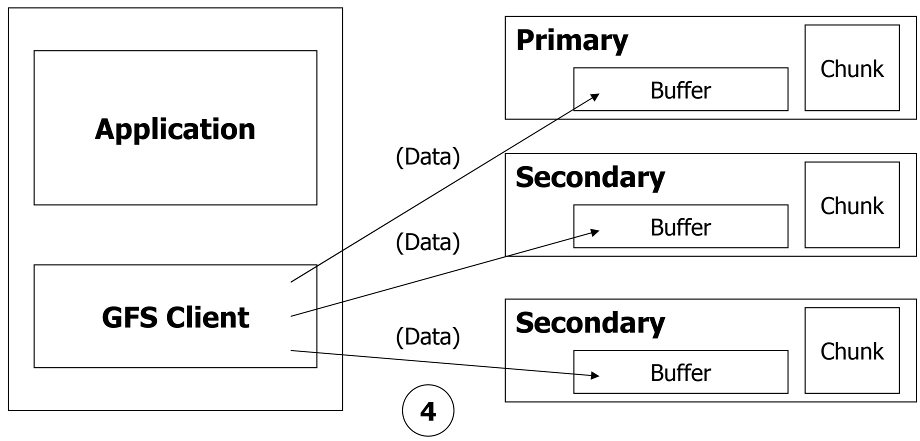
Read Algorithm (*Example)*



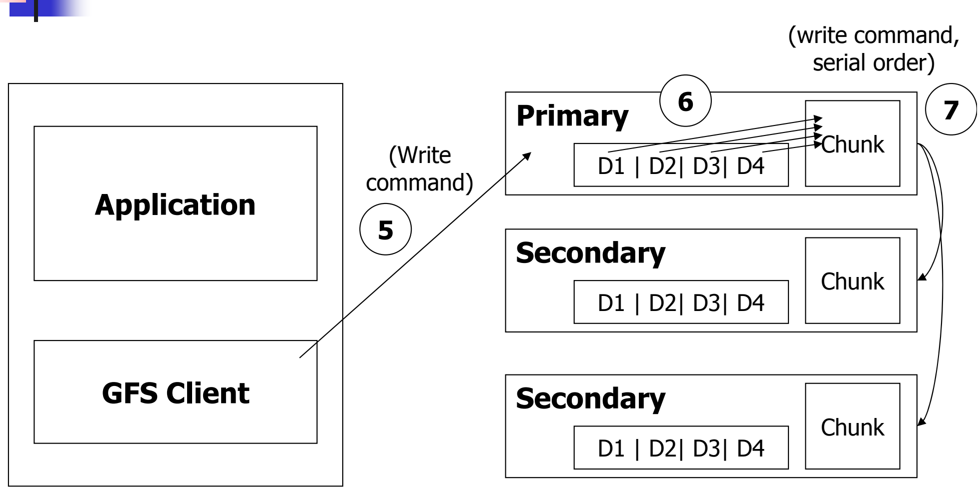
Write Algorithm



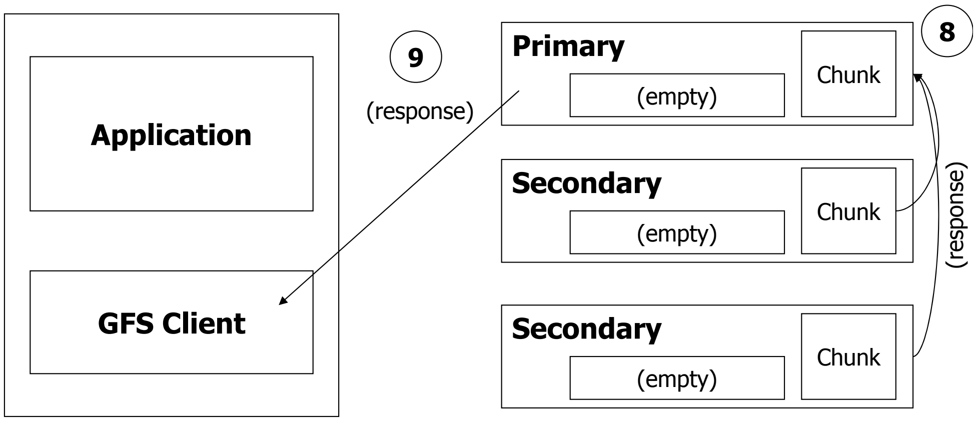
Write Algorithm



Write Algorithm



Write Algorithm



Write Algorithm

1. Application originates write request.
2. GFS client translates request from (filename, data) -> (filename, chunk index), and sends it to master.
3. Master responds with chunk handle and (primary + secondary) replica locations.
4. Client pushes write data to all locations. Data is stored in chunkservers' internal buffers.
5. Client sends write command to primary.

Write Algorithm

6. Primary determines serial order for data instances stored in its buffer and writes the instances in that order to the chunk.
7. Primary sends serial order to the secondaries and tells them to perform the write.
8. Secondaries respond to the primary.
9. Primary responds back to client.

Note: If write fails at one of chunkservers, client is informed and retries the write.

Record Append Algorithm

Important operation at Google:

- Merging results from multiple machines in one file.
- Using file as producer - consumer queue.

1. Application originates record append request.
2. GFS client translates request and sends it to master.
3. Master responds with chunk handle and (primary + secondary) replica locations.
4. Client pushes write data to all locations.

Record Append Algorithm

5. Primary checks if record fits in specified chunk.
6. If record does not fit, then the primary:
   - pads the chunk,
   - tells secondaries to do the same,
   - and informs the client.
   - Client then retries the append with the next chunk.
7. If record fits, then the primary:
   - appends the record,
   - tells secondaries to do the same,
   - receives responses from secondaries,
   - and sends final response to the client.

Observations

- Clients can read in parallel.
- Clients can write in parallel.
- Clients can append records in parallel.

Overview

- Design Motivations
- Architecture
- **Algorithms:**
  - **Read**
  - **Write**
  - **Record Append**
- Fault-Tolerance
- Performance Results

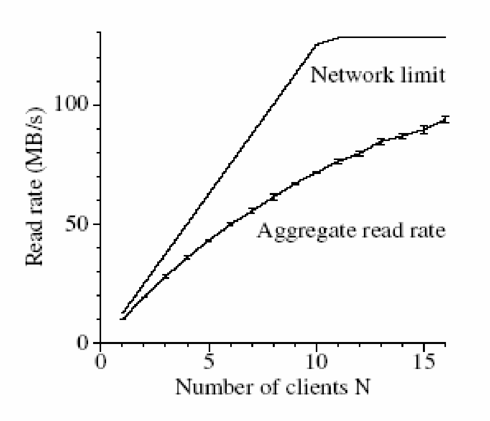
Fault Tolerance

- Fast Recovery: master and chunkservers are designed to restart and restore state in a few seconds.
- Chunk Replication: across multiple machines, across multiple racks.
- Master Mechanisms:
  - Log of all changes made to metadata.
  - Periodic checkpoints of the log.
  - Log and checkpoints replicated on multiple machines.
  - Master state is replicated on multiple machines.
  - "Shadow" masters for reading data if "real" master is down.
- Data integrity:
  - Each chunk has an associated checksum.

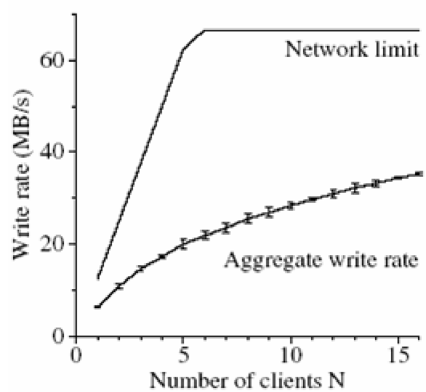
Performance *(Test Cluster)*

- Performance measured on cluster with:
  - 1 master
  - 16 chunkservers
  - 16 clients
- Server machines connected to central switch by 100 Mbps Ethernet.
- Same for client machines.
- Switches connected with 1 Gbps link.

Performance *(Test Cluster)*



Performance *(Test Cluster)*



Performance *(Real-world Cluster)*

- Cluster A:
  - Used for research and development.
  - Used by over a hundred engineers.
  - Typical task initiated by user and runs for a few hours.
  - Task reads MB's-TB's of data, transforms/analyzes the data, and writes results back.
- Cluster B:
  - Used for production data processing.
  - Typical task runs much longer than a Cluster A task.
  - Continuously generates and processes multi-TB data sets.
  - Human users rarely involved.
- Clusters had been running for about a week when measurements were taken.

Performance *(Real-world Cluster)*

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

Performance *(Real-world Cluster)*

- Many computers at each cluster (227, 342!)
- On average, cluster B file size is triple cluster A file size.
- Metadata at chunkservers:
  - Chunk checksums.
  - Chunk Version numbers.
- Metadata at master is small (48, 60 MB) -> master recovers from crash within seconds.

Performance *(Real-world Cluster)*

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 *(Real-world Cluster)*

- Many more reads than writes.
- Both clusters were in the middle of heavy read activity.
- Cluster B was in the middle of a burst of write activity.
- In both clusters, master was receiving 200-500 operations per second -> master is not a bottleneck.

Performance *(Real-world Cluster)*

Experiment in recovery time:

- One chunkserver in Cluster B killed.
- Chunkserver has 15,000 chunks containing 600 GB of data.
- Limits imposed:
    - Cluster can only perform 91 concurrent clonings.
    - Each clone operation can consume at most 6.25 MB/s.
- Took 23.2 minutes to restore all the chunks.
- This is 440 MB/s.

Conclusion

- Design Motivations
- Architecture
- Algorithms:
- **Fault-Tolerance**
- **Performance Results**