The Google File System

Written by
Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung
***presented at* SOSP 2003**

presented by
Thomas Reidemeister

Motivation for the Google File System

Google Search Engine Facts:

- **Google attempts to index the entire Internet**
- **Analytical backend has to process huge amounts of data for indexing and data mining**

Implications:

- **Need huge, distributed, available, and dependable file system.**
- **Google's problems differ from traditional file system design constraints (e.g. by workloads, by size of files, …)**

Outline

- Overview of Design Constraints and Decisions
- Centralized Architecture
- Operations: Consistency, Reading, Writing, Append, Snapshot
- Master Operation
- Measures to Attain Fault Tolerance
- Evaluation: Micro Benchmark
- Conclusion

Design Constraints and Decisions

Constraints:

- **Component failures are the norm**
- **Support for huge files by traditional file system standards**
- **Support for large streaming reads**
- **File append is dominant operation**
- **Throughput is favored over latency**
- **Custom file system API**
- **Use commodity hardware**

Decisions:

- **Use simple centralized architecture**
- **Achieve dependability through replication**
- **Files are large fixed-sized chunks**
- **File content is not cached**
- **Extend familiar FS APIs with special operations**

Are these issues really novel?

Centralized Architecture

GFS uses a central master that maintains the meta data and stores the data on replicated chunk servers.

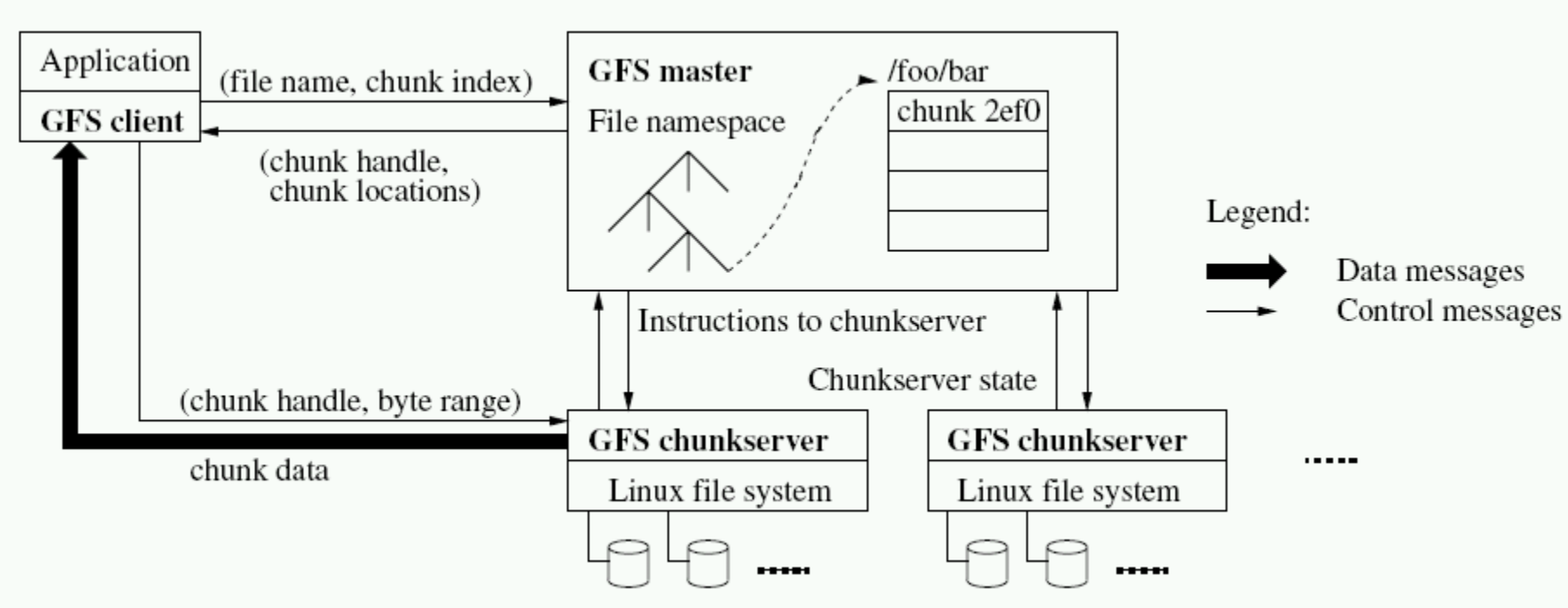


*Image taken from: Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google File System, SOSP2003

Centralized Architecture: GFS Master

Master server maintains all meta data, including:

- **Namespaces and operations thereon**
- **Access control information**
- **Chunk index: filename to chunk mapping**
- **Chunk replication and maintenance**

All metadata is kept in memory

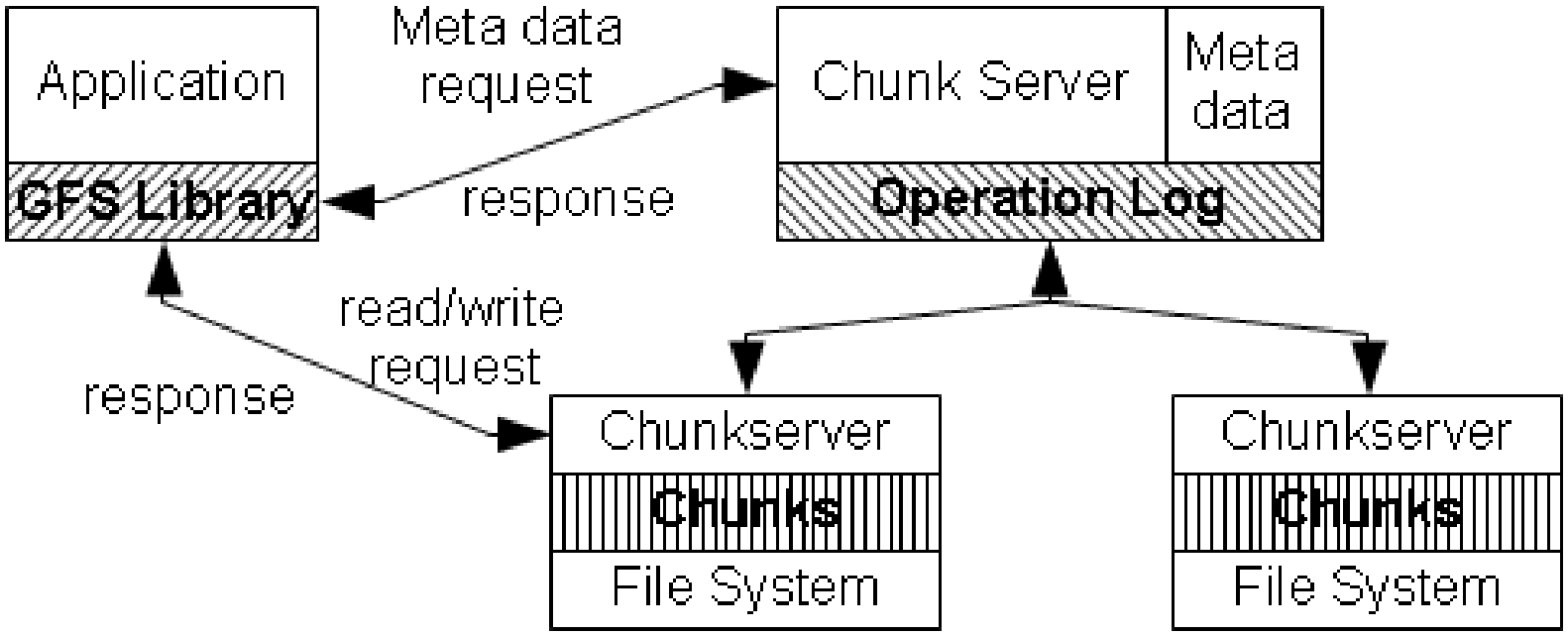
- **c.a 64 bytes per 64MB chunk**
- **Fast access**

Operations are logged to attain dependability

- **Log is replicated**

Centralized Architecture: Operation

GFS clients only communicate with master to obtain metadata, read/writes are done at the chunk servers.



Writes and appends also referred to as <u>mutations</u>

Operations: Overview

Metadata operations are executed at the master.

File operations that involve chunk server are:

- **Reading**
- **Writing**
- **Appending**
- **Snapshot**

Operations: Consistency Overview

Guarantees for meta data:

- **Namespace mutations are atomic**

Data consistency model:

- **"Consistent": All clients see the same data**
- **"Defined": All clients see what the mutation has written (on a per-chunk basis)**
- **Risk of stale reads and interleaved chunk writes; attempt to signal errors to client**

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

*Image taken from: Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google File System, SOSP2003

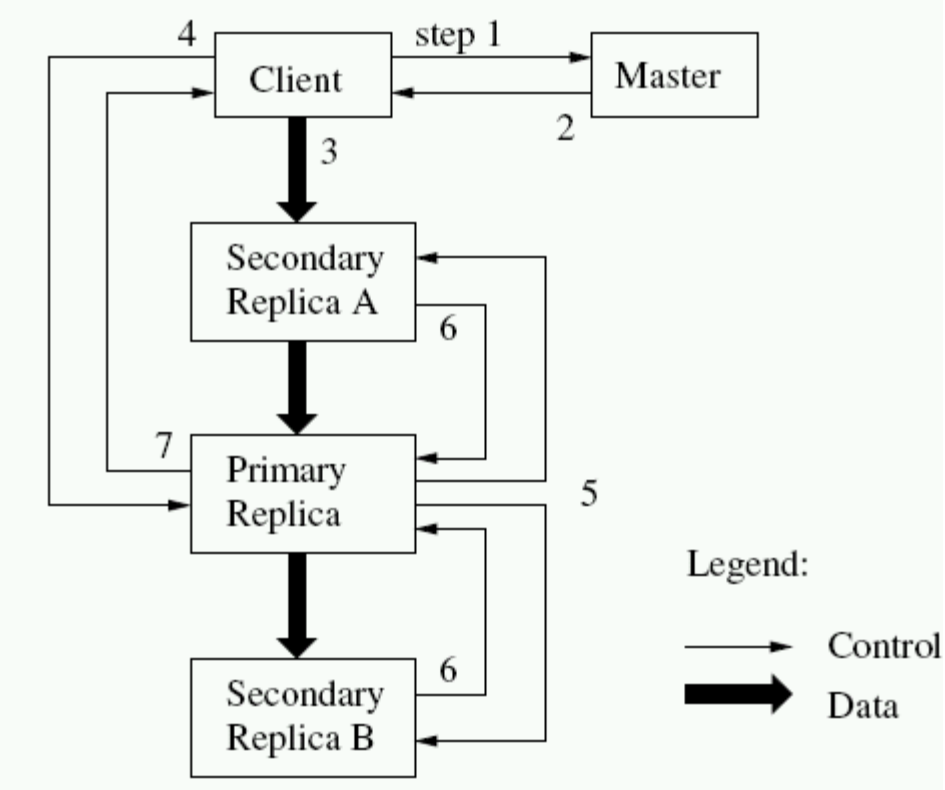
Operations: Reading

1. Application invokes read (filename, offset)
2. Client library translates offset to chunk index and sends it to master (filename, chunk index)
3. Master responds with handle (chunk, replica locations)
4. Client library selects location and sends (handle, offset) to it
5. Chunk server responds with requested data
6. Data is forwarded to application

Operations: Chunk Mutations

- **Master elects primary for each chunk among replicas**
- **Primary holds lease for at least 60s that is updated through keep-alive requests**
- **Primary coordinates chunk mutations**
- **However:**
  - **Client library sends all data to all replicas**
  - **Client library retries failed mutations**

Operations: Writing

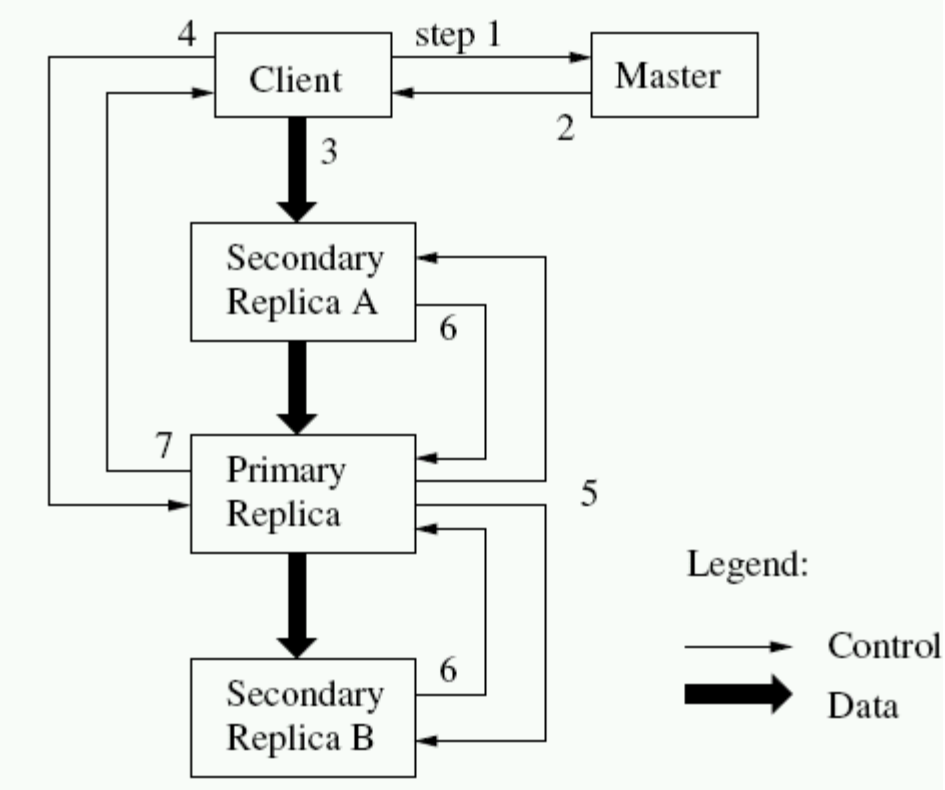


1. Client sends write request (filename, chunk index) to master
2. Master responds with chunk handle (chunk, replica locations)
3. Client sends data to all replicas; kept in memory
4. Client sends write to primary
5. Primary versions and performs write and informs replicas
6. Secondary replicas respond to primary
7. Primary reports to client

Hint: In the event of failures, client is informed and retries

*Image taken from: Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google File System, SOSP2003

Operations: Append



Appends are handled analogous to write

- **Special case (at 5): If data to append does not fit…**
- **Primary replica pads chunk and instructs secondary copies to do likewise**
- **Client is asked to request append with next chunk**

Hint: In the event of failures, client is informed and retries

*Image taken from: Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google File System, SOSP2003

Operations: Snapshot

GFS supports copy-on-write snapshots that allows check-pointing the state of directory trees.

1. **Master revokes leases from chunk primary**
2. **Master logs updates while performing snapshot**
3. **Master applies log to copy of meta data**
4. **On update a copy of the chunk is modified at the chunk server**

Master Operation

Performs meta data operations and coordinates system wide maintenance. Particular features are:

- **Namespace management and locking**
- **Replica placement**
- **Chunk creation, re-replication and rebalancing**
- **Garbage collection of deleted chunks**
- **Stale replica deletion**

Measures to Attain Fault Tolerance

Replication:

- **Multiple replicas per chunk (default 3)**
- **Intelligent chunk placement across racks**
- **Masters are shadowed**

Data integrity verification:

- **Use of checksums for data integrity (32 bit for each 64kb block)**

Monitoring:

- **Keep-alive messages**
- **Log all RPC client requests (i.e. excluding the transmitted data)**

Evaluation: Setup

Design is evaluated with micro benchmark and using real-world clusters.

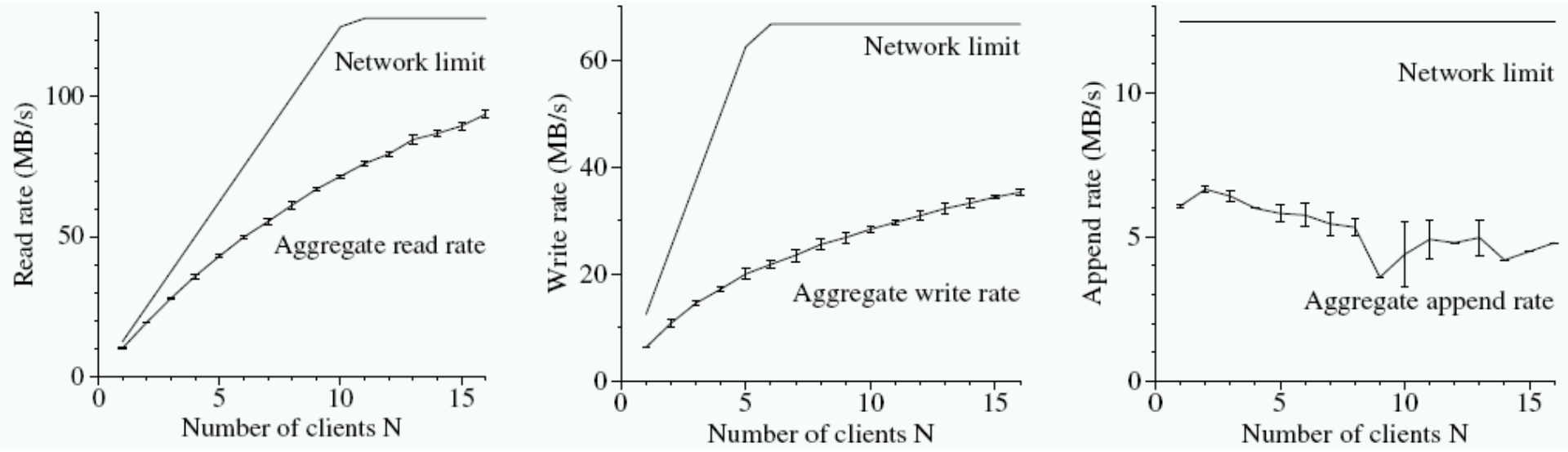
Micro benchmark:

- **One master with two replicas**
- **16 chunk servers**
- **16 clients**
- **100 Mbps Ethernet**

Real-world clusters:

- **Workloads of a research and a production clusters are presented in the paper *(not presented)***

Evaluation: Micro Benchmark



- **Reads achieve c.a. 75 % of network bandwidth limit**
- **Writes achieve c.a. 50 % of network bandwidth limit**
- **Concurrent appends lead to network congestion of chunk servers (experiment setup is unrealistic according to authors)**

*Image taken from: Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google File System, SOSP2003

Conclusion

- GFS leverages commodity hard- (and software)
- GFS is optimized for large reads and sequential appends
- Single master paradigm simplifies coordination
- Fault tolerance is achieved through replication, continuous monitoring and data integrity verification
- High throughput is achieved through:
  - Delegating mutations to chunk servers
  - Keeping the meta data in memory
- GFS guarantees serialized mutations and atomic meta data operations
- Risk of stale reads

Potential Discussion Points

- **Why not accessing partitions directly (i.e., in GFS chunks are actual files on a Linux FS)?**
- **What are possible failure modes for snapshot?**
- **Why is the append bottleneck “unrealistic” in the evaluation?**
- **Which artifacts of the GFS design could be handled by Chubby?**
- **Why not leveraging OS-level snapshots (i.e. LVM snapshots)?**