The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung
Google

***SOSP'03,* October 19–22, 2003, New York, USA**

Hyeon-Gyu Lee, and Yeong-Jae Woo

Memory & Storage Architecture Lab.

School of Computer Science and Engineering

Seoul National University

Outline

- **Design overview**
  - Assumptions
  - Interface
  - Architecture
  - Single master
  - Chunk size
  - Metadata
  - Consistency model
- System interactions
- Master operation
- Fault tolerance
- Measurements
- Conclusion

Assumptions

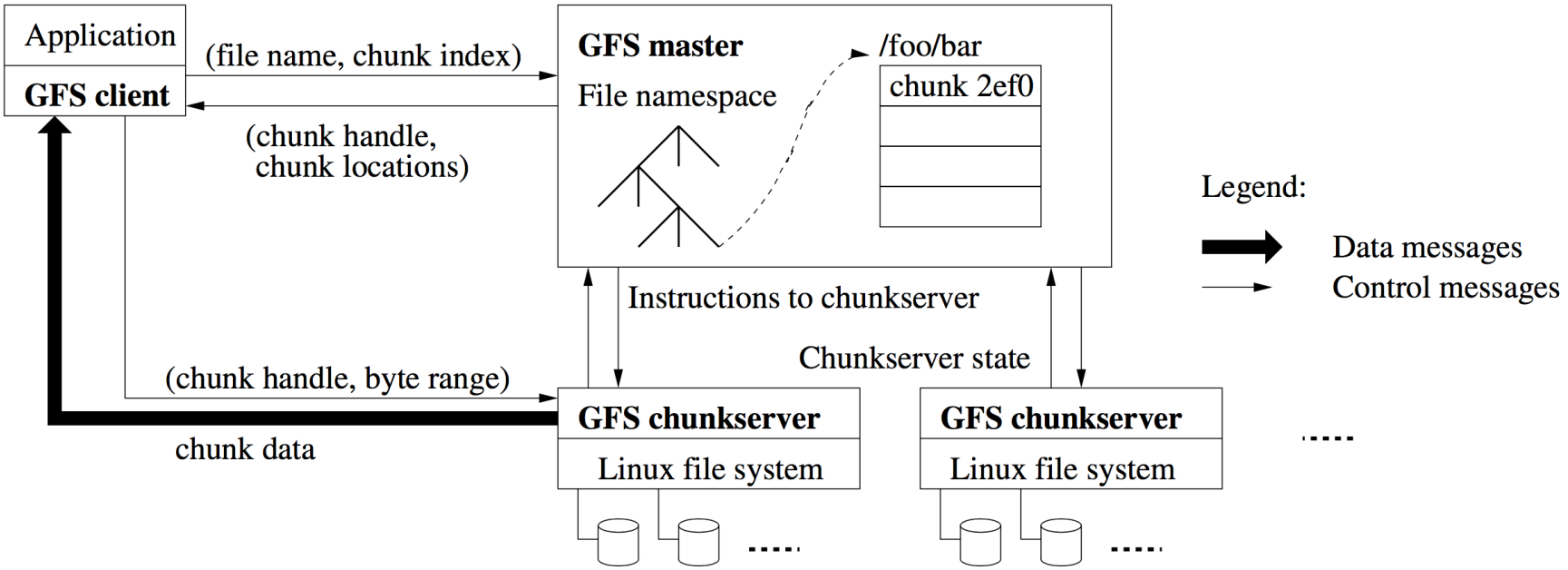
- **The system is built from many inexpensive commodity components**
- **The system stores a modest number of large files**
  - A few million files, each typically 100 MB or larger
- **The workloads primarily consist of**
  - Large sequential reads and small random reads
  - Large sequential writes that append data to files
- **The system must efficiently implement semantics for clients that concurrently append to the same file**
- **High sustained bandwidth is more important than low latency**

Interface

- **Files are organized in directories and identified by pathnames**
- **File operations**
  - Create, delete, open, close, read, and write
  - Snapshot
    - Creates a copy of a file or a directory tree
  - Record append
    - Allows multiple clients to append data to the same file concurrently while guaranteeing the atomicity

Architecture

- A GFS cluster consists of a single master and multiple chunkservers
- Files are divided into fixed-size chunks
- Master maintains all file system metadata
- Neither the client nor the chunkserver caches file data

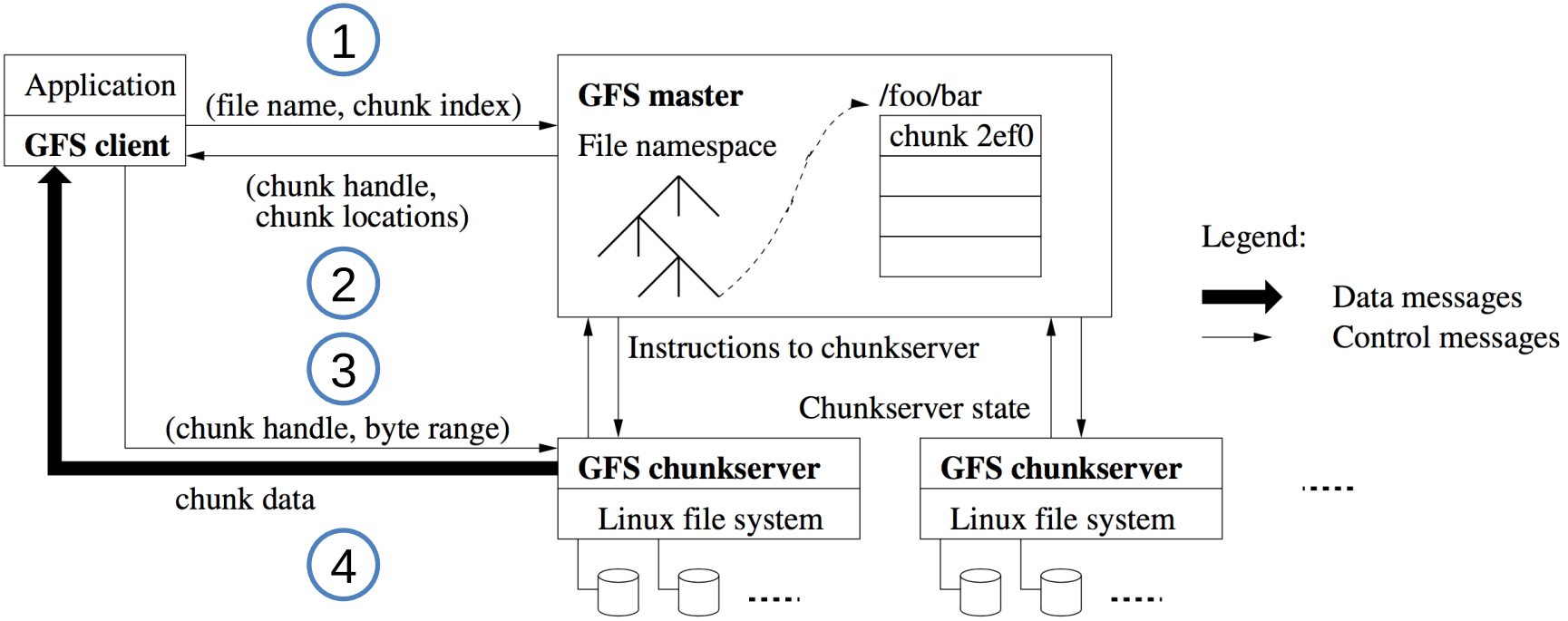


Architecture

- **A GFS cluster consists of a single master and multiple chunkservers**
- **Files are divided into fixed-size chunks**
  - Each chunk is identified by an immutable and globally unique 64 bit chunk handle
- **Master maintains all file system metadata**
  - Namespace, access control information, mapping from files to chunks, and current locations of chunks
  - Master periodically communicates with each chunkserver in HeartBeat messages
- **Neither the client nor the chunkserver caches file data**

Single master

- **Master makes chunk placement and replication decisions**
- **Client asks the master which chunkserver it should contact**
  - Client caches this information and interacts with the chunkserver directly



Chunk size

- **Large chunk size (64 MB)**
  - Pros
    - Reduces clients' need to interact with the master
    - Reduces network overhead since client is likely to perform many operations on a large chunk
    - Reduces the size of the metadata stored on the master
  - Cons
    - If many clients are accessing the same small file, chunkservers may become hot spots
      - Ex) an executable single-chunk file
      - Sol) Stores such files with a higher replication factor

Metadata

- **Metadata types (stored in memory)**
  - File and chunk namespaces
  - Mapping from files to chunks
  - Locations of each chunk's replicas

  → Be kept persistent by logging to an operation log

  → Rebuild by asking each chunkserver
- **Master periodically scans its entire state in background**
  - For chunk garbage collection, re-replication, and chunk migration
- **Size of metadata is not a problem**
  - Master maintains < 64 B of metadata for each 64 MB chunk
  - File namespace data requires < 64 B per file name
- **Operation log contains a record of metadata changes**
  - Also provides a logical time line that defines the order of concurrent operations
  - Changes are not visible to clients until metadata changes are made persistent
  - Master recovers its state by replaying the operation log from last checkpoint

Consistency model

- **GFS has a relaxed consistency model**
- **Guarantees by GFS**
  - File namespace mutations are atomic and are handled in master with locking
  - File region state after mutation

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

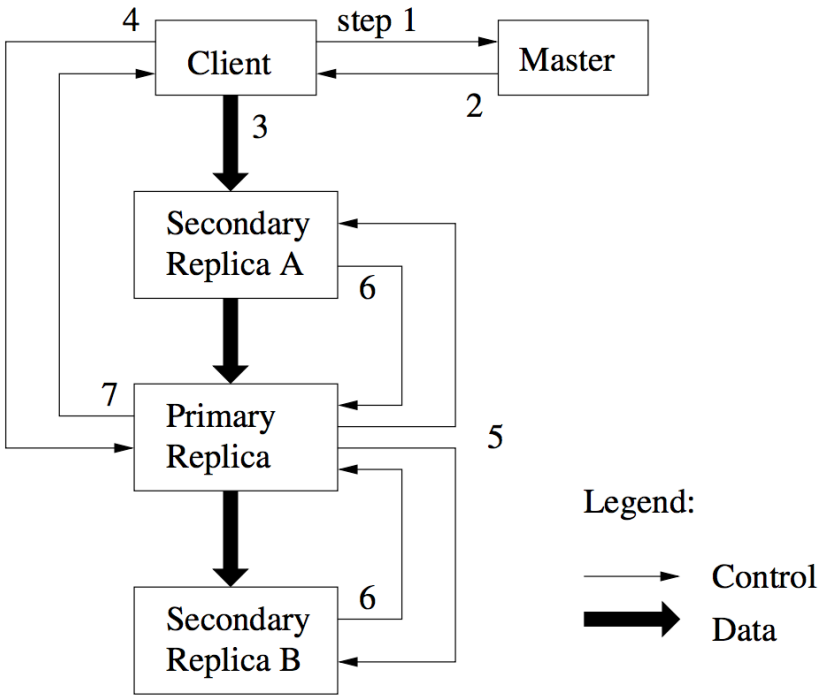
- **Implications for applications**
  - Applications deal with the relaxed consistency model by using appends, checkpointing, and writing self-validating, self-identifying records

Outline

- Design overview
- **System interactions**
  - Leases and mutation order
  - Data flow
  - Atomic record appends
  - Snapshot
- Master operation
- Fault tolerance
- Measurements
- Conclusion

Leases and mutation order

- **Mutation is performed at all chunk's replicas using leases to maintain a consistent mutation order across replicas**
  - Master grants a chunk lease to one of the replicas, called primary
  - Primary picks a serial order for all mutations to the chunk



Data flow

- **GFS decouples the data flow from the control flow**
- **Each machine forwards the data to the "closest" machine in the network topology that has not received it**
  - Distances can be estimated from IP addresses
- **GFS minimizes the latency by pipelining the data transfer over TCP connections**

Atomic record appends

- **GFS appends the data to the file at least once atomically at an offset of GFS's choosing and returns that offset to the client**
- **Algorithm of atomic record append**
  - Client pushes the data to all replicas of the last chunk of the file
  - Client sends the append request to the primary
  - Primary checks to see if data causes chunk to exceed the maximum size
    - If so, pads chunk to the maximum size and tells secondaries to do same, then tells client to retry on the next chunk
    - If the record fits, primary appends data to its replica, tells secondaries to write at the exact offset where it has

Snapshot

- **Snapshot operation makes a copy of a file or a directory tree almost instantaneously**
- **GFS uses copy-on-write technique to implement snapshots**
  - Revokes any outstanding leases on the chunks to snapshot
  - Logs the snapshot operation to disk and duplicates the metadata
  - If a client wants to write, creates a new chunk and tells secondaries to do same

Outline

- Design overview
- System interactions
- **Master operation**
  - Namespace management and locking
  - Replicas
  - Garbage collection
- Fault tolerance
- Measurements
- Conclusion

Namespace management and locking

- **GFS allows multiple operations to be active and uses locks to ensure serialization**
- **Namespace is represented as a lookup table mapping full pathnames to metadata**
  - Represented as a tree in memory with prefix compression
- **Each node has an associated read-write lock and each master operation acquires locks before it runs**
- **Locks are acquired in a total order to prevent deadlock**
  - Ordered by level in the namespace tree and lexicographically in the same level

Replicas

- **GFS spreads chunk replicas across racks**
  - This ensures that some replicas will survive even if an entire rack is damaged
  - This can exploit the aggregate bandwidth of multiple racks

- **Replicas are created for chunk creation, re-replication, and rebalancing**
  - Places new replicas on chunkservers with below-average disk space utilization
  - Re-replicates a chunk when the number of available replicas falls below a user-specified goal
  - Rebalances replicas periodically for better disk space and load balancing

Garbage collection

- **GFS reclaims the storage for deleted files lazily via garbage collection at the file and chunk levels**
  - File level
    - When a file is deleted, master logs the deletion and renames the file to a hidden name that includes the deletion timestamp
    - During the master's scan, hidden files are removed if they have existed for more than 3 days
  - Chunk level
    - Each chunkserver reports a subset of chunks it has and master identifies and replies with orphaned chunks
    - Each chunkserver deletes orphaned chunks

Outline

- Design overview
- System interactions
- Master operation
- **Fault tolerance**
  - High availability
  - Data integrity
- Measurements
- Conclusion

High availability

- **Fast recovery**
  - Master and chunkserver are designed to restore their state and start in seconds
- **Chunk replication**
  - Each chunk is replicated on multiple chunkservers on different racks
  - Users can specify different replication levels
    - The default is 3
- **Master replication**
  - Operation logs and checkpoints are replicated on multiple machines
  - One master process is in charge of all mutations and background activities
  - Shadow masters provide read-only access to the file system when the primary master is down

Data integrity

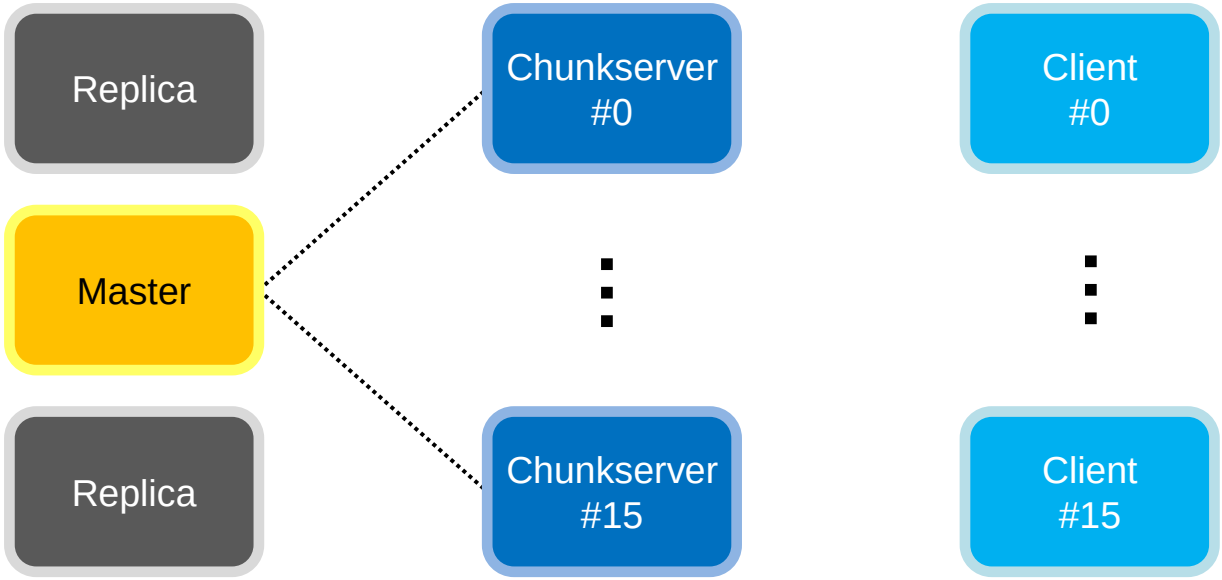
- **Each chunkserver uses checksumming to detect data corruption**
  - A chunk is broken up into 64 KB blocks
  - Each block has a corresponding 32 bit checksum
- **Chunkserver verifies the checksum of data blocks for reads**
- **During idle period, chunkservers scan and verify the inactive chunks**

Outline

- Design overview
- System interactions
- Master operation
- Fault tolerance
- **Measurements**
  - Micro-benchmarks
  - Real world clusters
- Conclusion

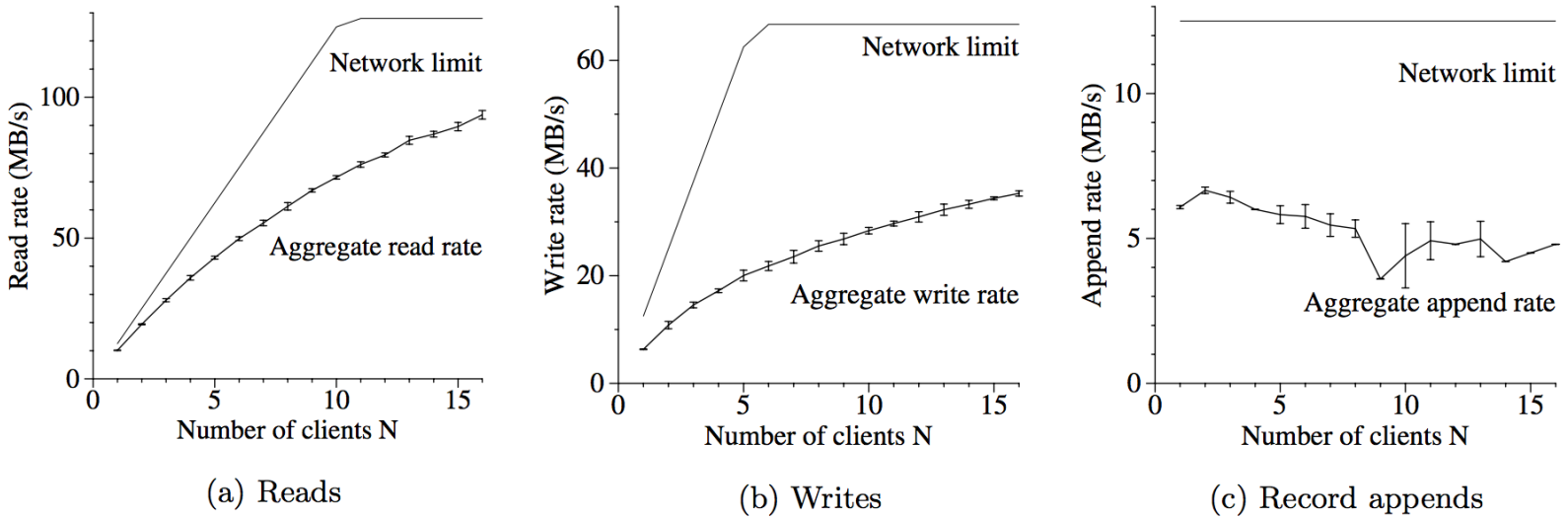
Micro-benchmarks

- **Test environment**
  - GFS cluster consists of 1 master with 2 replicas, 16 chunkservers, and 16 clients
  - Each machine
    - Dual 1.4 GHz PIII processor
    - 2 GB memory
    - 2 * 80 GB 5400 RPM HDD
    - 100 Mbps full-duplex Ethernet



Micro-benchmarks

- **Reads**
  - Each client reads a randomly selected 4 MB * 256 from a 320 GB file
- **Writes**
  - Each client writes 1 GB data to a new file in a series of 1 MB writes
  - Each write involves 3 different replicas
- **Record appends**
  - All clients append simultaneously to a single file



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

Read world clusters

- **Test environment**
  - Cluster A
    - Used for research and development
    - Reads MBs ~ TBs data and writes the results back
  - Cluster B
    - Used for production data processing
    - Generates and processes TBs data

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

Read world clusters

- **Performance metrics for two GFS clusters**
  - Cluster B was in the middle of a burst write activity
  - The read rates were much higher than the write rates

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

- **Recovery time**
  - Killed a single chunkserver of cluster B
    - Which had 15 K chunks containing 600 GB data
  - All chunks were restored in 23.3 minutes

Outline

- Design overview
- System interactions
- Master operation
- Fault tolerance
- Measurements
- **Conclusion**

Conclusion

- **GFS treats component failures as the norm rather than the exception**
- **GFS optimizes for huge files that are appended to and then read**
- **GFS provides fault tolerance by replication and fast recovery**
  - Replication to tolerate against chunkserver failures
  - Checksumming to detect data corruption
- **GFS delivers high throughput to many concurrent clients**
  - By separating file system control from data transfer
- **Master involvement in common operation is minimized**
  - By a large chunk size
  - By chunk leases
- **GFS is widely used within Google as the storage platform**