The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung

SOSP '03

CSC 456 Operating Systems
Seminar Presentation (11/11/2010)

Elif Eyigöz, Walter Lasecki

Outline

- **Background**
- Architecture
  - Master
  - Chunkserver
  - Write/Read Algorithm
  - Namespace management / Locking
- Reliability
  - Replication in Chunkservers
  - Replication in Master
- Conclusion
  - Comparison with AFS and NFS

Background – Introduction

Google

- Search engine: Huge workload
- Applications: Lots of data being processed
- Extreme scale: 100 TB of storage across 1000s of disks on over a thousand machines, accessed by hundreds of clients

AFS, NFS versus GFS

Read/write, Cache, Replication, Consistency, Faults

Background – Motivational Facts

- Extreme scale:
  - Thousands of 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
- Goal: Large, distributed, highly fault tolerant file system

Background – Design Observations

- Component failures are normal
  - Fault tolerance and automatic recovery are needed

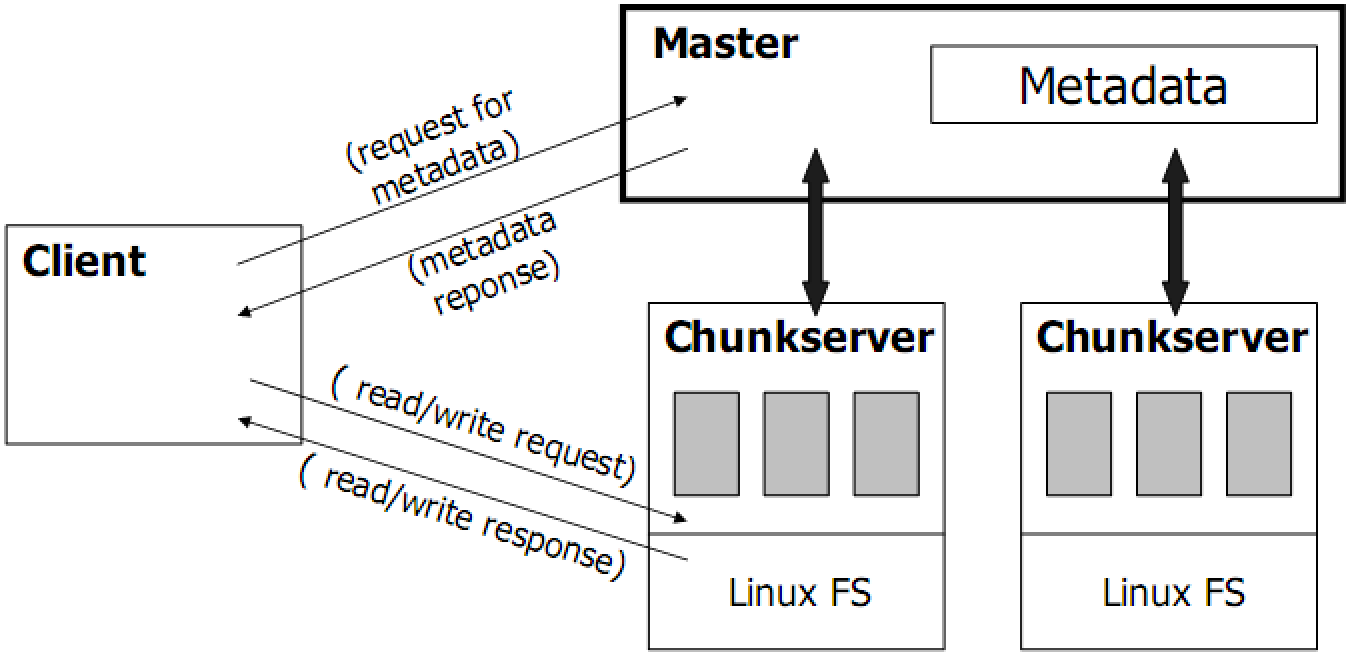
- Huge files are common (Multi GB)
  - I/O and block size assumptions

- Record appends are more prevalent than random writes
  - Appending is the focus of performance optimization and atomicity guarantees

Outline

- Background
- **Architecture**
  - Master
  - Chunkserver
  - Write/Read Algorithm
  - Namespace management / Locking
- Reliability
  - Replication in Chunkservers
  - Replication in Master
- Conclusion
  - Comparison with AFS and NFS

Architecture



The Role of the Master

Metadata Server : Maintain all file system metadata

- File namespace
- File to chunk mapping
- Chunk location information
- Keep all the metadata in the master's memory (FAST)

Location of chunks and their replicas:

- Master does not keep a persistent record (no disk I/O)
- Operations log for persistence and recovery
- Poll it from chunkservers (master as monitor, no extra cost)

Master as metadata server

Monitor

- *Heartbeat* messages to detect the state of each chunkserver
- Communicate with chunkservers periodically

Centralized Controller

- System-wide activities
- Manage chunk replication in the whole system

Master

- Single master
  - Simplify the design of the system
  - Control chunk placement using global knowledge
  - Potential bottleneck?

- Avoiding bottleneck
  - Clients do no read/write data through master
  - Clients cache metadata (e.g., chunk handles, locations)
  - Client typically asks for multiple chunks in the same request
  - Master can also include the information for chunks immediately following those requested

Caching

- Metadata
  - Cached on client after being retrieved from the Master
  - Only kept for a period of time to prevent stale data

- File caching
  - Clients never cache file data
  - Chunkservers never cache file data (Linux's buffer will for local files)
  - File working sets too large
  - Simplifies cache coherence

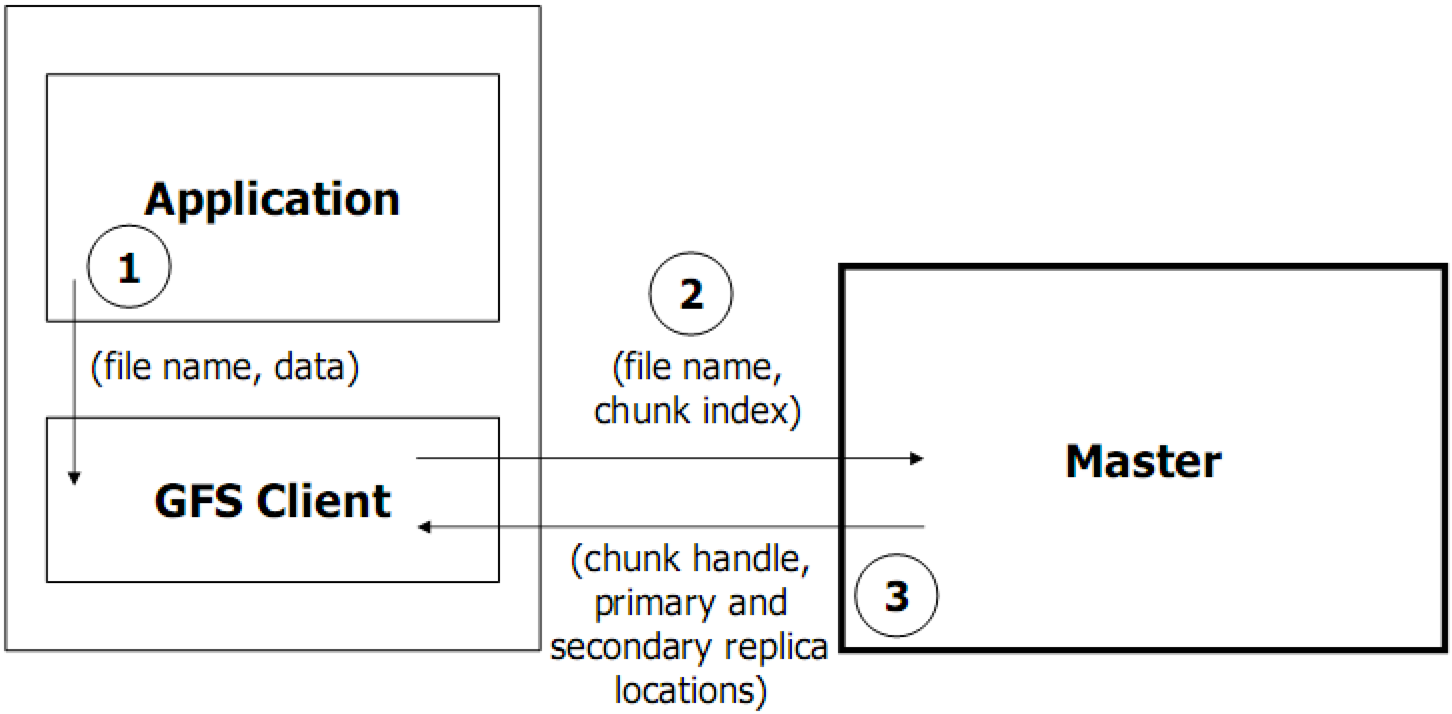
Chunkserver

- Large chunk size
- 64 MB: much larger than typical file system blocks
- Advantages:
- reduces network overhead via reduced client-server interaction
- reduces network overhead
- reduces the size of metadata stored on master
- Disadvantages:
- internal fragmentation (solution: lazy space allocation)
- large chunks can become hot spots:

  (solutions: higher replication factor, staggering application start time, client-to-client communication)

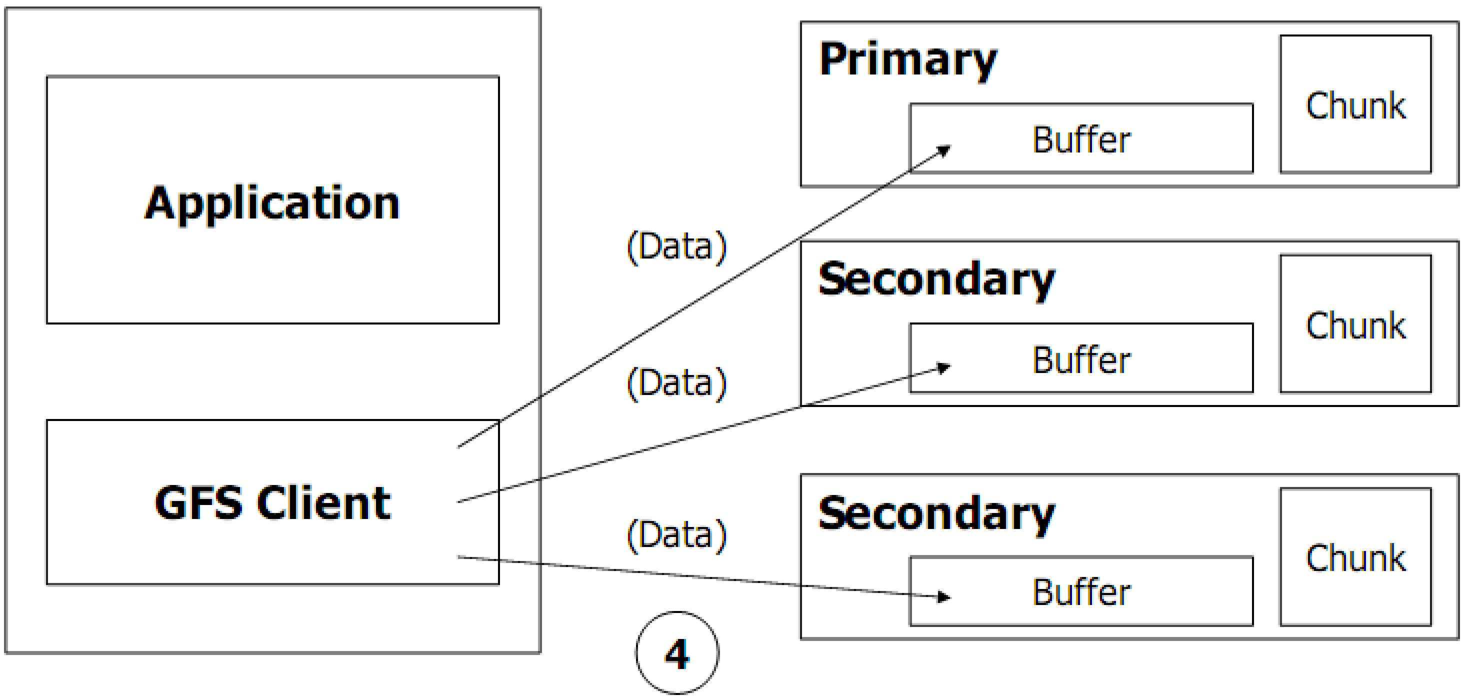
Write/Read Algorithm

- Write Algorithm



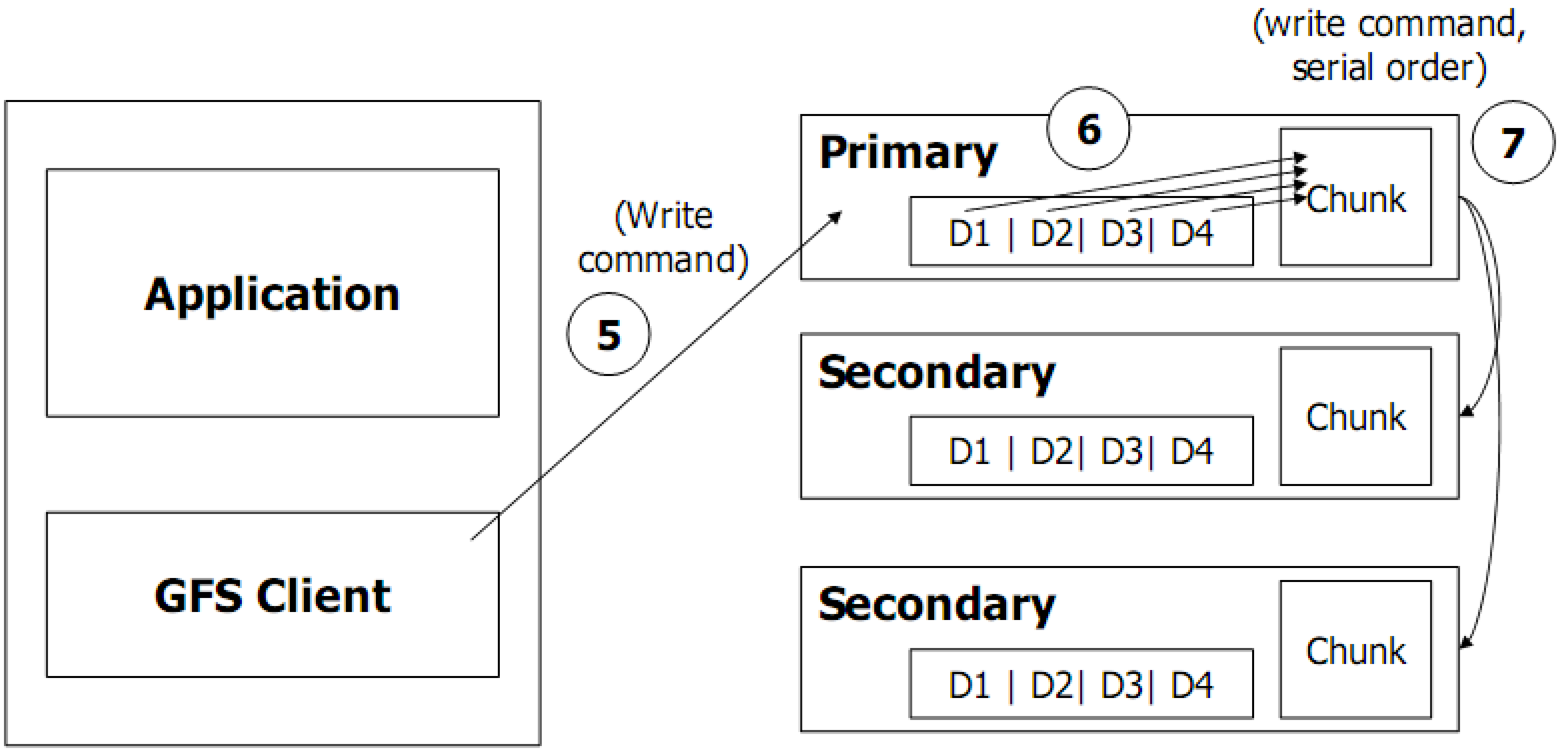
Write/Read Algorithm

- Write Algorithm



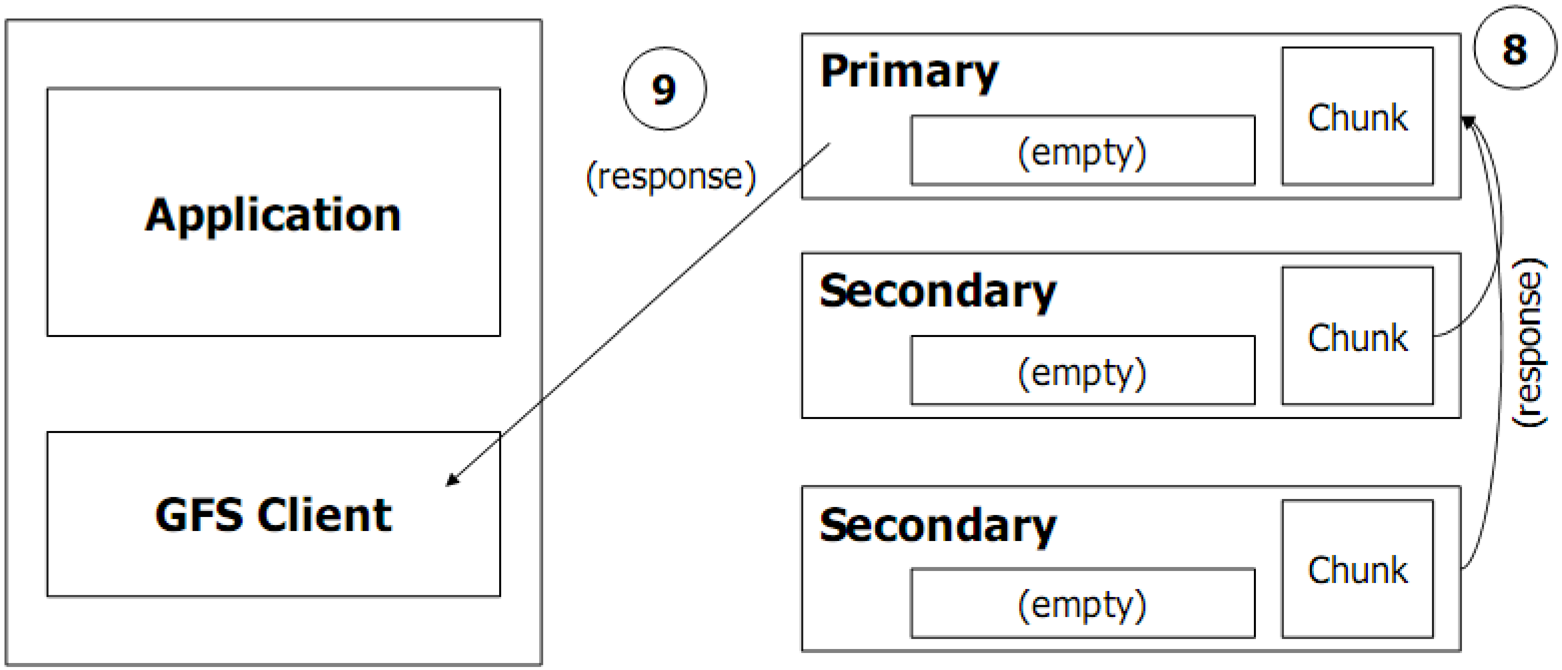
Write/Read Algorithm

- Write Algorithm



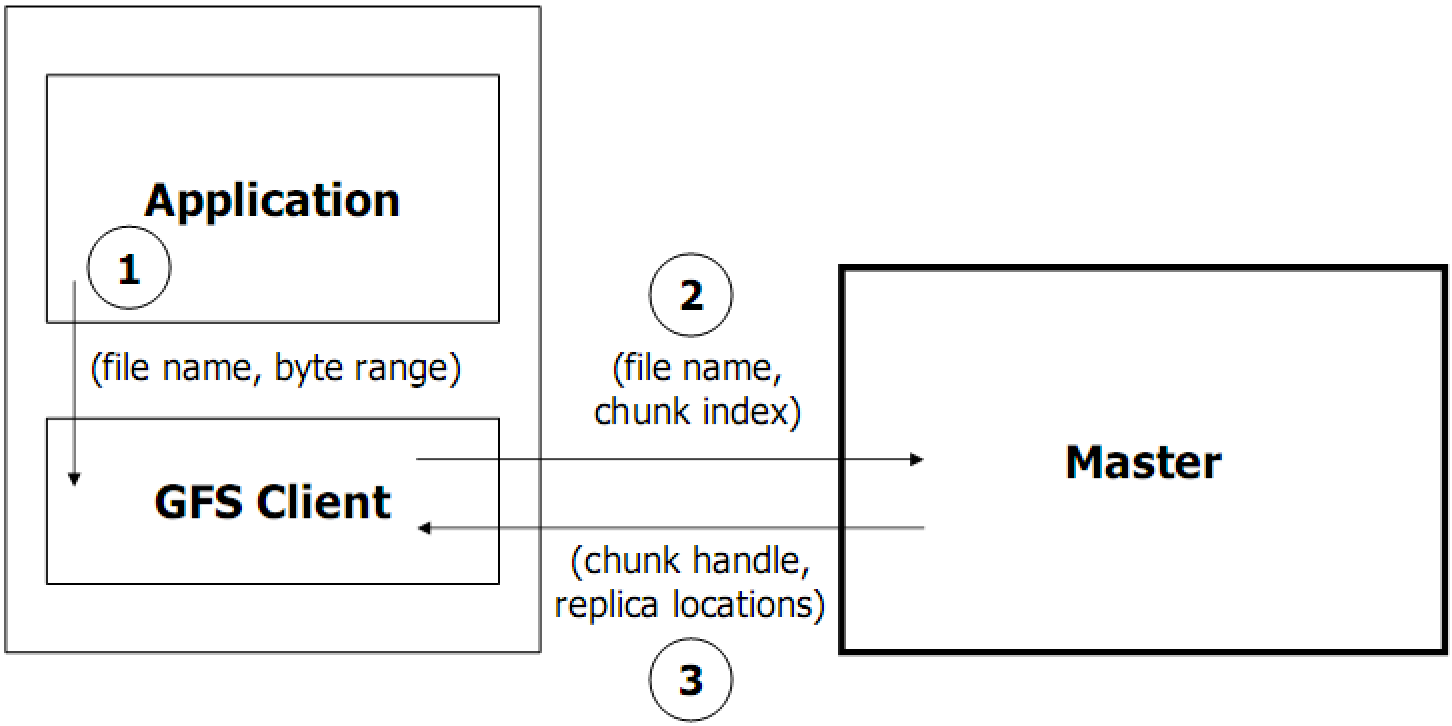
Write/Read Algorithm

- Write Algorithm



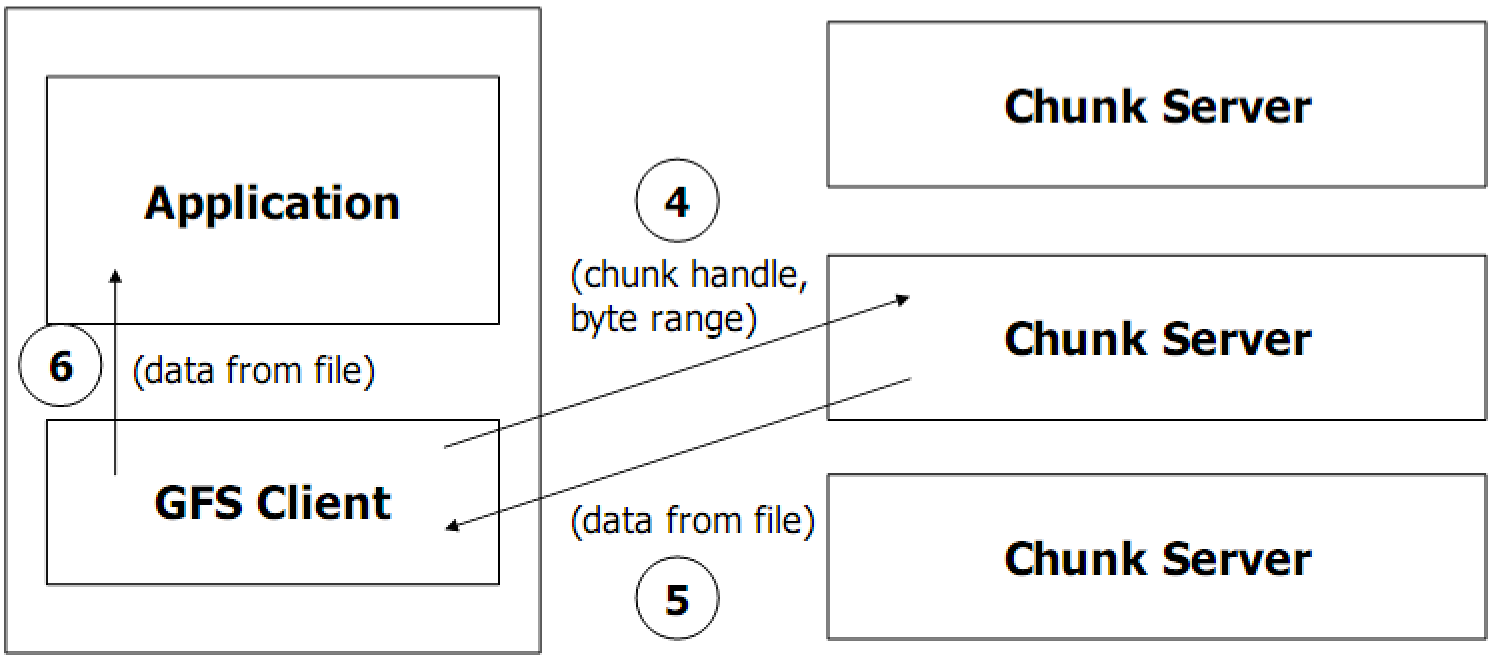
Write/Read Algorithm

- Read Algorithm



Write/Read Algorithm

- Read Algorithm



Namespace Management

- Namespace
  - How file system represent and manage the hierarchy of files and directories

User's perspective	File system's perspective

`/dir`
`/dir/file1`
`/dir/file2`
 	
`dir` is a directory
`file1` is a file
`file2` is a file
	?

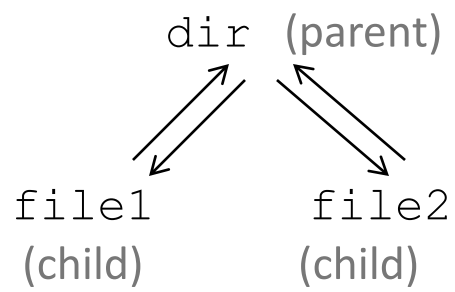
Namespace Management

- Traditional file systems
  - Per-directory data structure that lists all files in the directory
  - Hierarchy: based on *<u>parent-child tree</u>*

`/dir`
`/dir/file1`
`/dir/file2`

`dir` is a directory
`file1` is a file
`file2` is a file

User's perspective



File system's perspective

Namespace Management

- Google File System
- No such per-directory data structure
- Lookup table mapping full pathname to metadata
- Hierarchy: based on *full pathname*

User's perspective	File system's perspective

`/dir`
`/dir/file1`
`/dir/file2`
 	
`dir` is a directory
`file1` is a file
`file2` is a file
	`/dir`
	`/dir/file1`
	`/dir/file2`

Namespace Management

- What's the advantage of the namespace management in Google File System?
- Locking
  - Transaction operation
  - Deadlock prevention

Locking – Transactional Operation

- Traditional file systems
- Locking on the certain data structure (e.g., mutex)
- Example: Create new file under a directory
- e.g., `create /dir/file3`

```
lock (dir->mutex)
create file3
add file3 to dir
unlock (dir->mutex)
```

- Only one creation in a certain directory at one time
  - Reason: lock the directory
  - Prevent from creating files with the same name simultaneously
  - Add new file into the directory

Locking – Transactional Operation

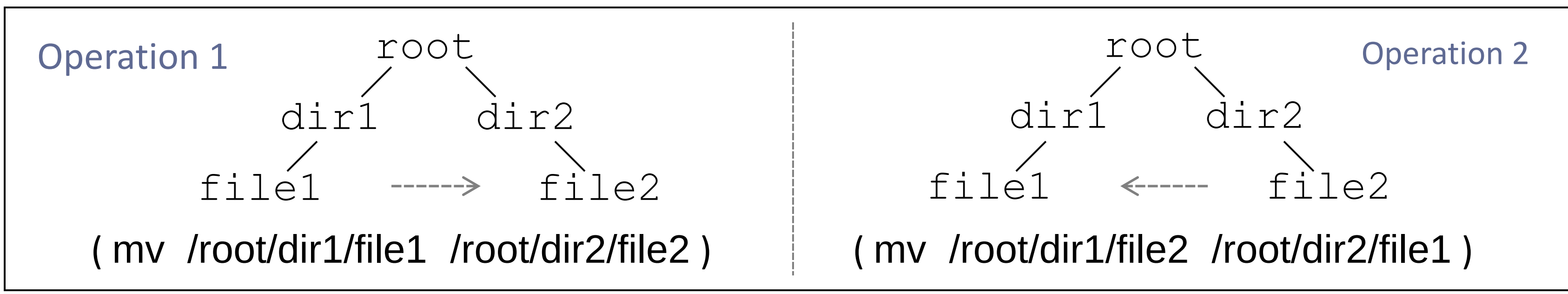
- Google File System
- Locking on the pathname
- Example: Create new file under a directory
- e.g., create `/dir/file3, /dir/file4/,` ……

`read_lock (/dir)`
`write_lock (/dir/file3)`
`create file3`
`unlock (/dir/file3)`
`unlock (/dir)`
	`read_lock (/dir)`
	`write_lock (/dir/file4)`
	`create file4`
	`unlock (/dir/file4)`
	`unlock (/dir)`
		…

- Allow concurrent mutations in the same directory
  - Key: using <u>read lock</u> for `dir`. Why it can?
  - By locking pathname, it can lock the new file before it is created → Prevent from creating files with the same name simultaneously
  - No per-directory data structure → do not need to add new file into the directory

Locking – Deadlock Prevention

- Traditional file systems (Linux)
- Hierarchy: based on ***parent-child tree***
- Ordering for resources : parent, child
- Ordering for requests : lock parent first, then lock child
- Problem: e.g., `rename` operation



- First step of `rename` : lock `dir1` and `dir2`
- But it cannot decide which one should be locked first
- Linux only allows one rename operation one time in one FS

Locking – Deadlock Prevention

- Google file system
- Hierarchy: based on ***full pathname***
- Based on consistent total order
- First ordered by ***level*** in the namespace tree
- ***Lexicographically*** within the same level
- Example: `rename` operation

Operation 1 : `mv /root/dir1/file1 /root/dir2/file2`
Operation 2 : `mv /root/dir1/file2 /root/dir2/file1`

- First step of `rename` : lock `dir1` and `dir2`
- Level order : `dir1` and `dir2` are in the same level, so then
- Lexicographical order : Lock `dir1` first

Outline

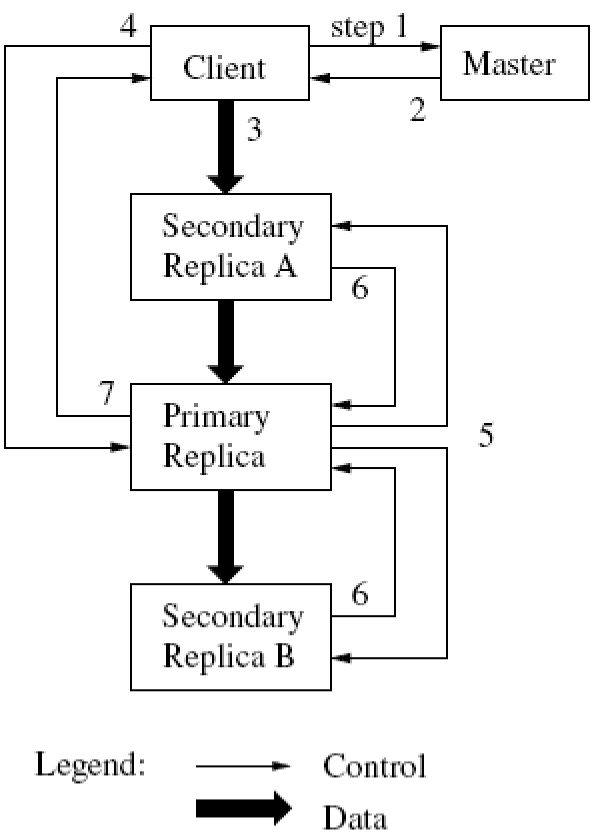
- Background
- Architecture
  - Master
  - Chunkserver
  - Write/Read Algorithm
  - Namespace management / Locking
- **Reliability**
  - Replication in Chunkservers
  - Replication in Master
- Conclusion
  - Comparison with AFS and NFS

Replication in Chunkservers

- Replication
  - Create redundancy
  - Each chunk is replicated on multiple chunkservers
  - By default, store three replicas
- Replication of chunks
  - Each mutation is performed at all the chunk’s replicas
  - Split the replication into two flows
    - Data flow : replicate the data
    - Control flow : replicate the write requests and their order

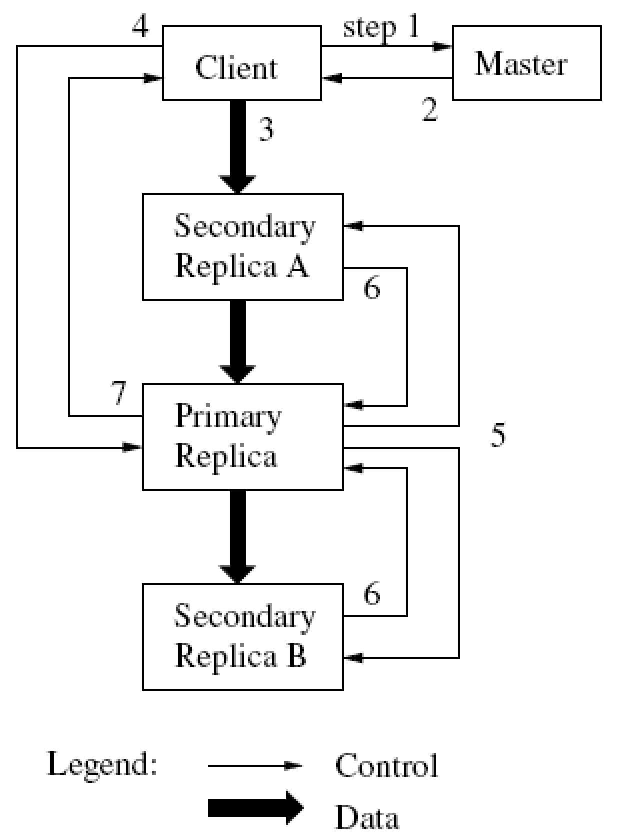
Replication in Chunkservers

- Data flow (Step 3)
- Client pushes data to all replicas
- Data is pushed linearly along a chain of chunkservers in a pipelined fashion.
- e.g., Client → Sec. A → Primary → Sec. B
- Once a chunkserver receives some data, it starts forwarding immediately



Replication in Chunkservers

- Control flow
  - (Step 4) After data flow, client send write request to the primary
  - The primary assigns consecutive serial number to all the write requests it received
  - (Step 5) The primary forwards the write requests and the serial number order to all secondary replicas



Replication in Master

- Replication of master state
  - Log of all changes made to metadata
  - Periodic checkpoints of the log
  - Log and checkpoints are replicated on multiple machines
  - If master fails, another replica would be activated as the new master
  - “shadow” masters provide read-only service
    - Not mirrors, may lag the master slightly
    - Enhance read throughput and availability for some files

Outline

- Background
- Architecture
  - Master
  - Chunkserver
  - Write/Read Algorithm
  - Namespace management / Locking
- Reliability
  - Replication in Chunkservers
  - Replication in Master
- **Conclusion**
  - Comparison with AFS and NFS

Conclusion

- AFS, NFS and GFS
  - Read/write, Cache, Replication, Consistency, Faults
- The design is driven by Google's application workloads and technological environment
- Advantages
- Know how to improve performance
- Design the file system only for Google
- Disadvantages
- May not be adopted in other system
- Single master may be still a potential bottleneck

References

- [1] The Google File System, SOSP ’03
- [2] Presentation slide of GFS in SOSP ’03
- [3] “Operating System Concepts”, 8th edition, 2009
- [4] Linux 2.6.20

Thank you!