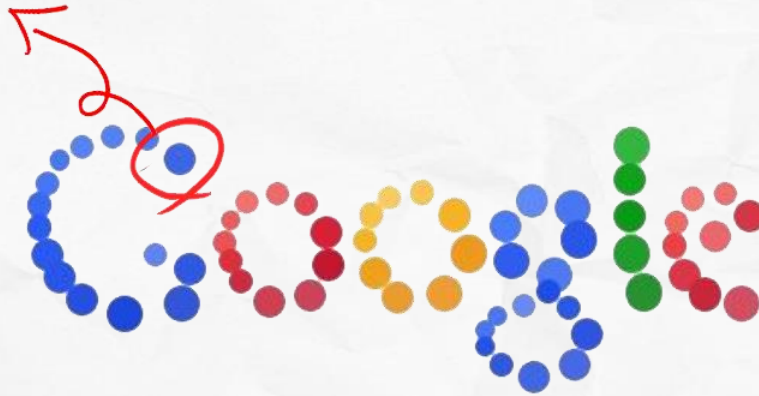


64mb




The Google File System

– Distributed File System



INDEX

0 1	Motivation
0 2	Background
0 3	Methods
0 4	Result
0 5	Conclusion



Motivation _ why gfs?

“to meet the rapidly growing demands of Google’s data processing needs.”

-> performance, scalability, reliability ↑

Motivation

- component failures(os bugs, human errors, and the failures of disks,) are the **norm**
- files are **huge** by traditional standards .
- overwrite, random write < **append new data,**
- **co designing the application and fs API** increase flexibility
Ex. relaxed consistency model, atomic append operation

Assumption (Background)

- many, inexpensive commodity components that often fail
- stores few million files (100MB or larger)
- many large, sequential writes that append data to files
- implement a processing method for clients accessing the same file concurrently
- High sustained bandwidth is more important than low latency.

architecture

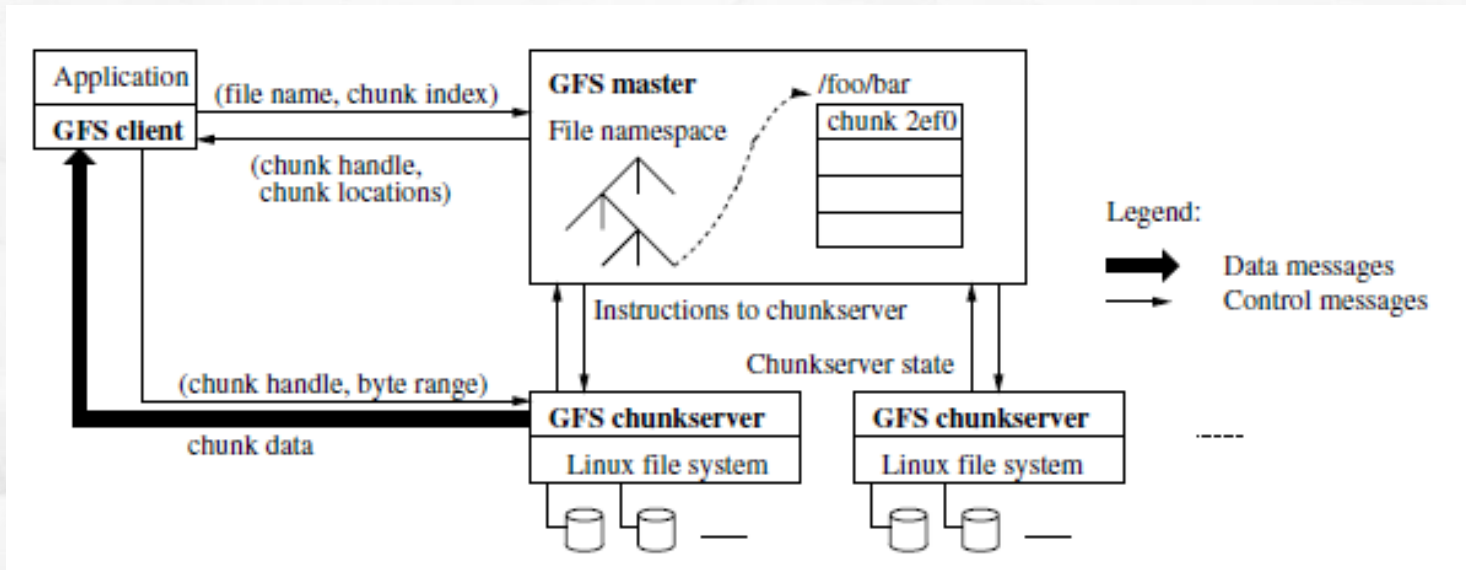
- interface

not posix but familiar file system interface api

have file operation : read, write,,,, + **snapshot, record append**

architecture

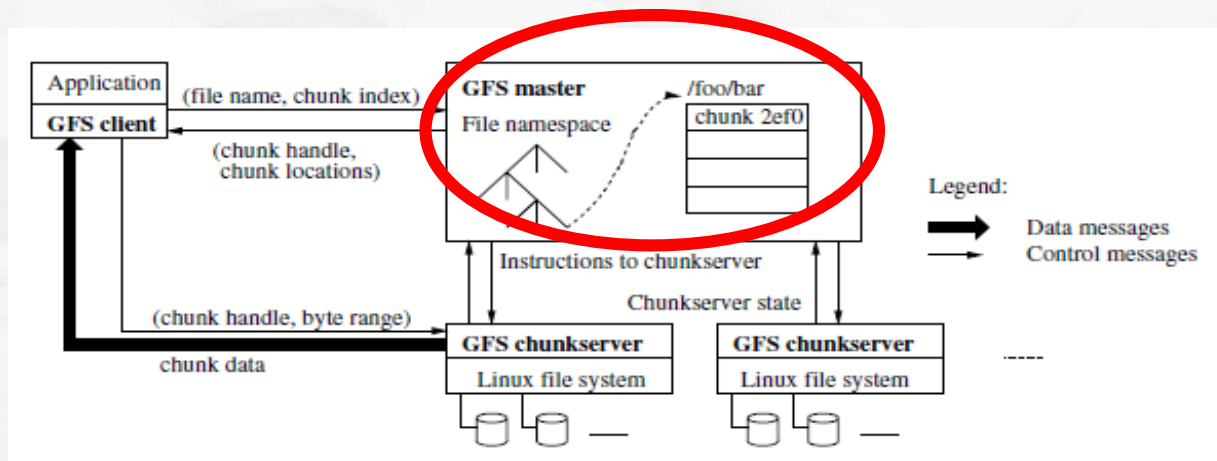
- single master + multiple chunk servers + clients
- files are divided into fix sized chunk (64 MB)
- chunkserver store chunks



Architecture_master

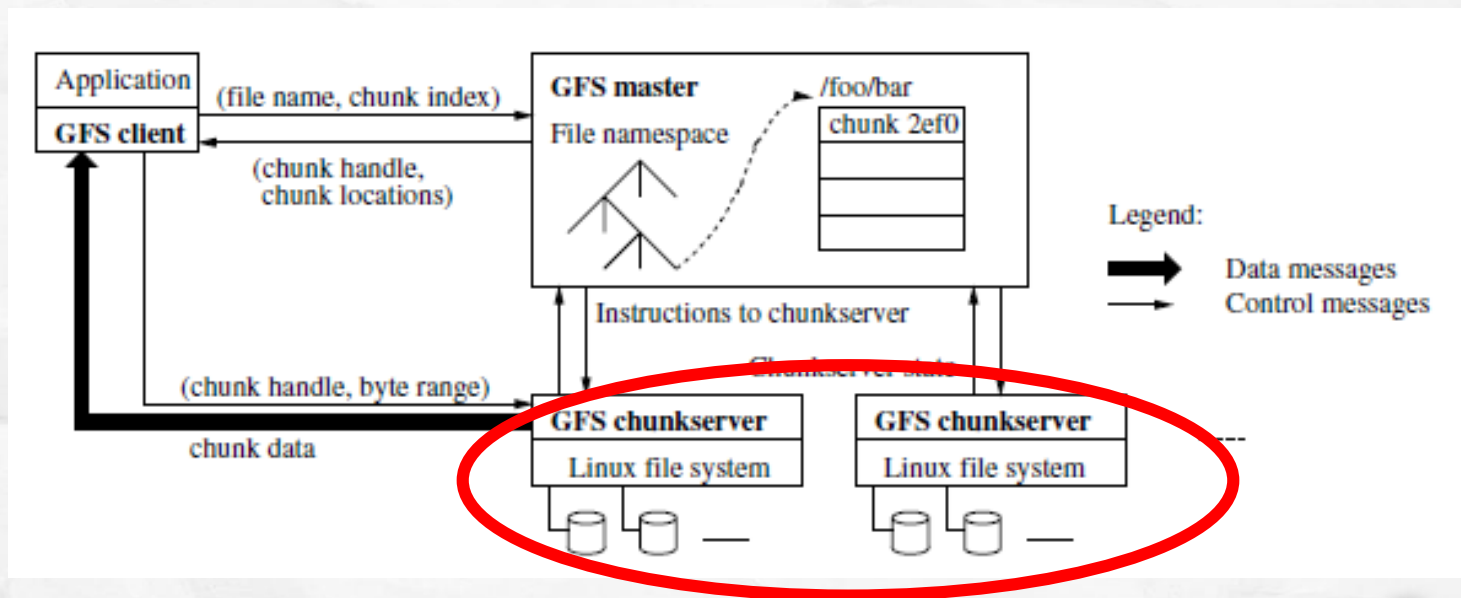
have all file system **metadata**

- metadata? namespace, access control info, mapping from file – chunks, location of chunks
- **manage system-wide activities** – chunk lease management, garbage collection of orphaned chunk, chunk migration between chunkserver
- **communicate with chunkserver** by **sending heartbeat message periodically** (for instructions, state monitoring)



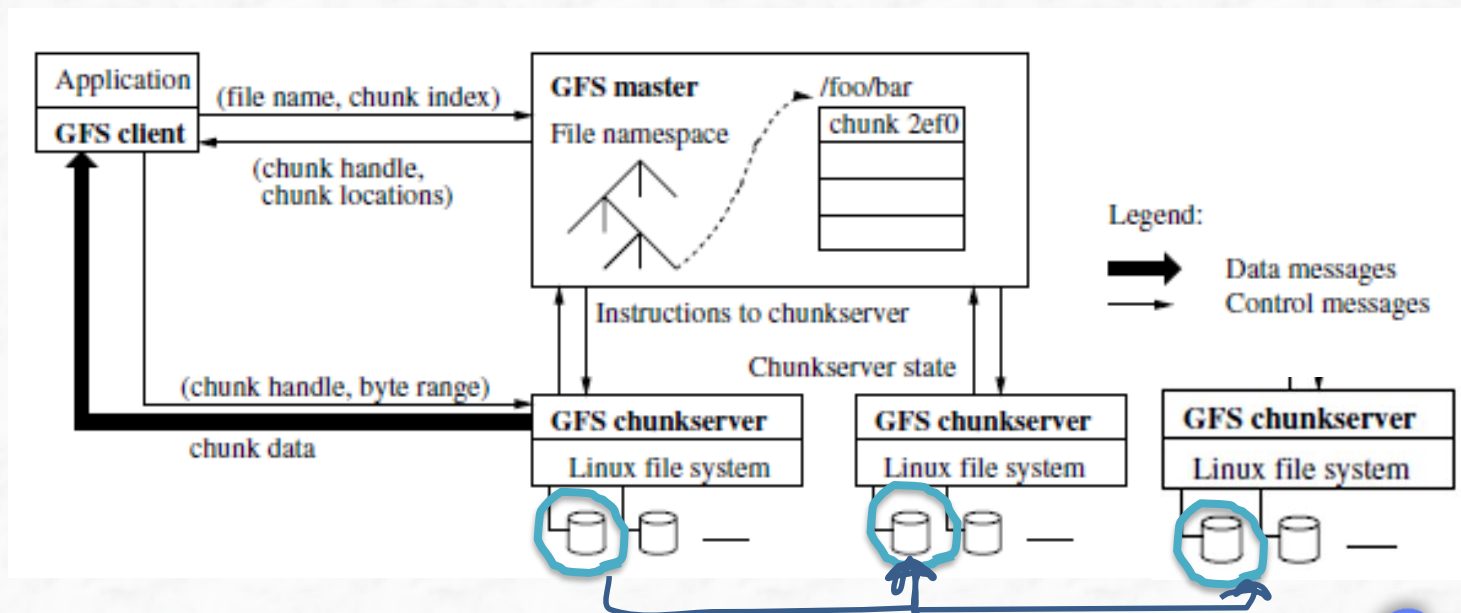
Architecture_chunkserver

- chunkserver **store chunks** on local disks as linux files
- each chunk is replicated on multiple chunkservers (normally,3)



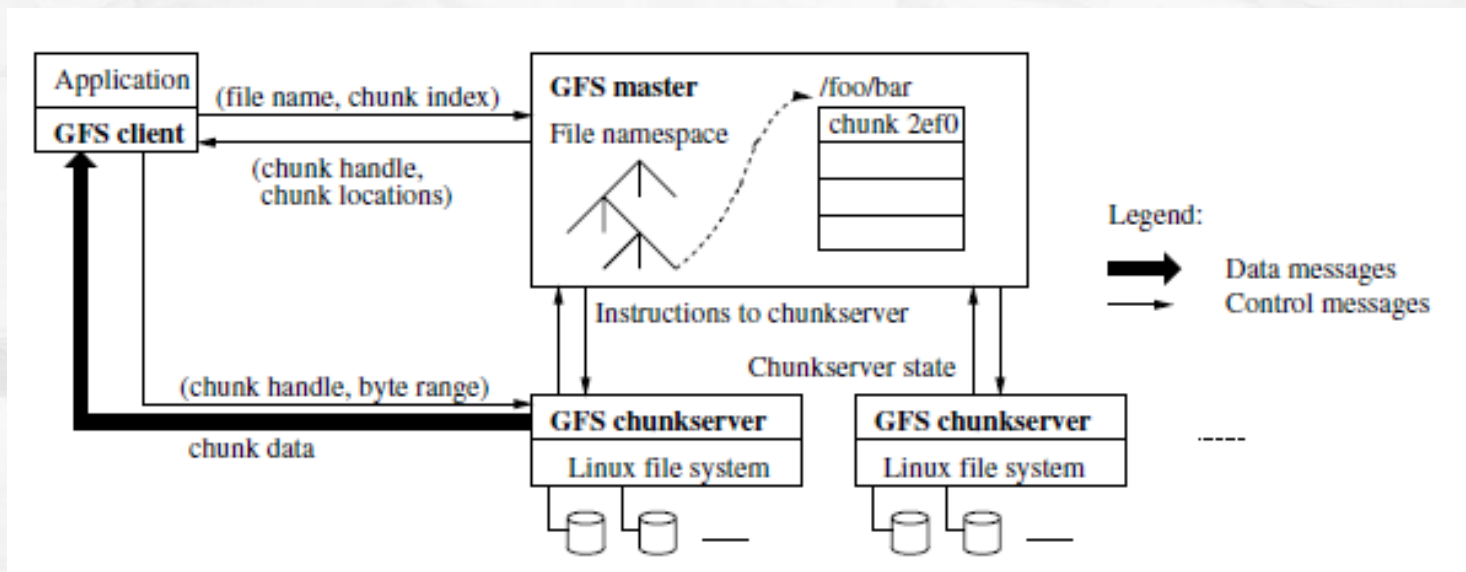
Architecture_chunkserver

- chunkserver **store chunks** on local disks as linux files
- each chunk is replicated on multiple chunkservers (normally,3)

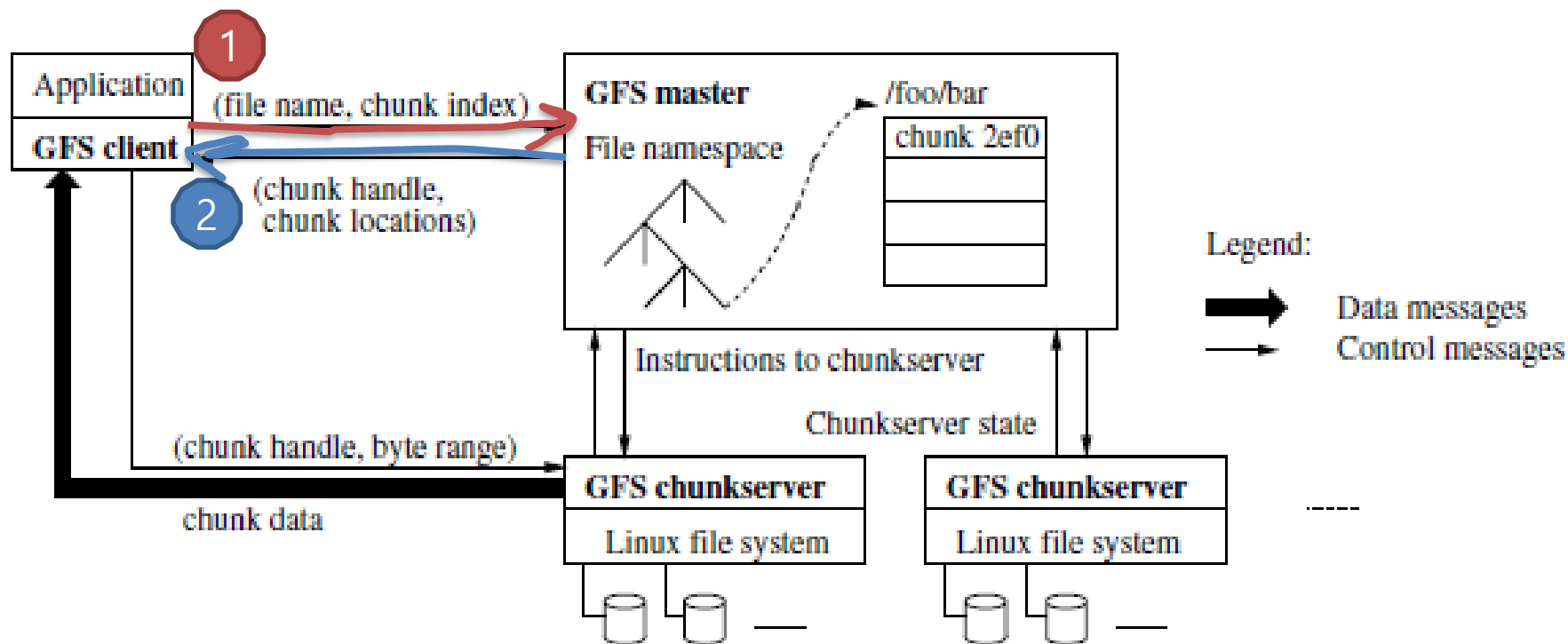


Architecture_client

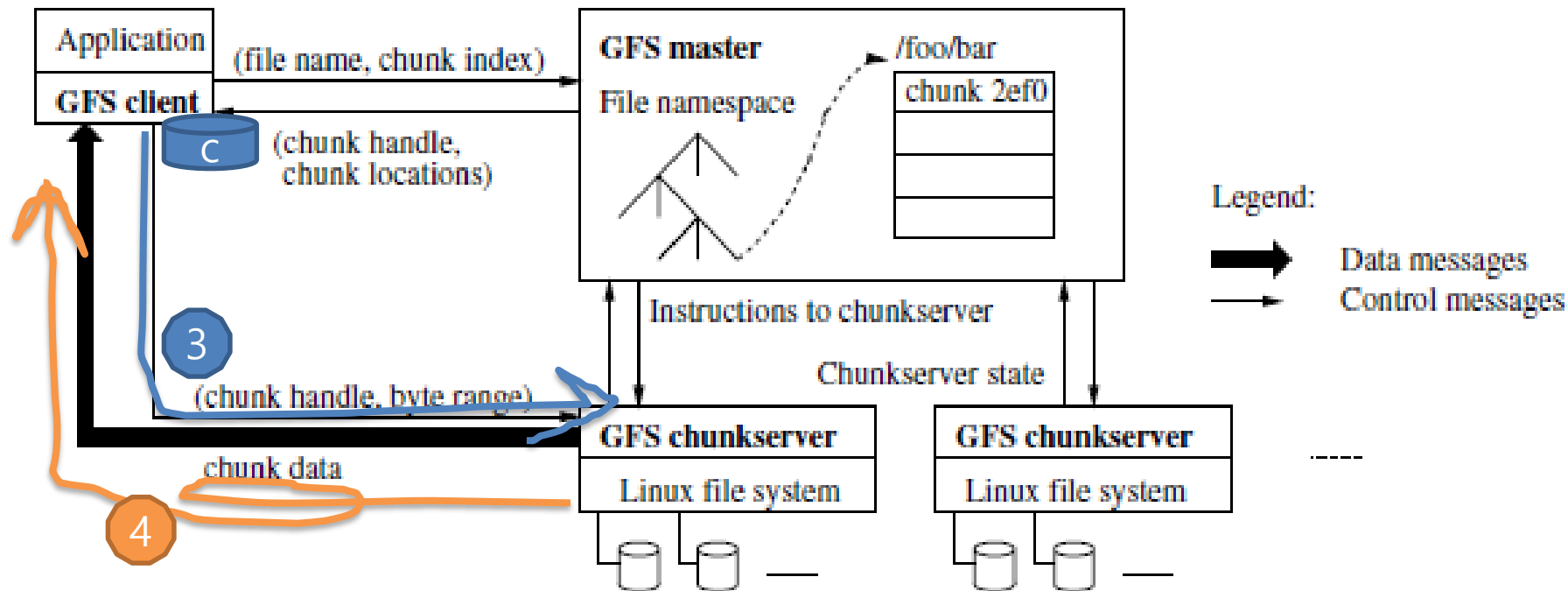
- client code linked into each applications
- Communicate with the **master** for metadata operations
- Communicate with the **chunkservers** for Read/Write operations



operation_read



operation_read



large chunk size

64MB

advantages :

- Reduces the **number of times the client communicates with the master.**
- Reduce the size of the **metadata.** (can keep in memory)
- Reduces **network overhead** by keeping persistent TCP connections to the chunkserver over an extended period of time

Disadvantages:

- **a small file** consists of a small number of chunks. if many clients are accessing the same file, those chunks may become **hot spots.**

metadata

3 types of metadata

1. file and chunk namespace
2. mapping from files to chunks
3. locations of each chunks' replicas

1,2,3 -> memory

12-> operation log in master disk

3-> do not store persistently. instead, ask each chunkserver about each chunks at master startup and whenever chunkserver joins the cluster

Log -> the logical time line that defines the order of concurrent operations kept on GFS master's local disk

master checkpoint its state whenever the log grows beyond a certain size

consistency model

	Write	Record Append
Serial success	<i>defined</i>	<i>defined interspersed with inconsistent</i>
Concurrent successes	<i>consistent but undefined</i>	
Failure	<i>inconsistent</i>	

Table 1: File Region State After Mutation

- changes to namespace are **atomic**
 - done by single master
 - master uses log to define global total order of operations
- The state of a file region after a data mutation depends on the **type of mutation, whether it succeeds or fails, and whether there are concurrent mutations.**
- consistent** : All clients see the same file, regardless of which replicas they read from.
 - defined** : consistent + all clients will see the entire mutation
 - undefined but consistent** : after Concurrent successful mutations. all clients see the same data, but it may not reflect what any one mutation has written.
 - inconsistent (also undefined)** : A failed mutation. different client sees different data at different times.

consistency model

	Write	Record Append
Serial success	<i>defined</i>	<i>defined interspersed with inconsistent</i>
Concurrent successes	<i>consistent but undefined</i>	
Failure	<i>inconsistent</i>	

Table 1: File Region State After Mutation

write : data written at an application-specified file offset

record append : data appended atomically at least once even in the presence of concurrent mutations, **but at an offset of GFS' s choosing**(GFS may insert padding or records duplicates in between)

-> since a failure at any replica makes the client try the write again there might be some duplicate data. Therefore GFS does not guarantee against duplicates but in anyway **the write will be carried out at least once**

system interactions

Leases and mutation order

- each mutation is performed at all the chunk's replicas
- leases used to maintain a consistent mutation order across replicas
 - master grants a chunk lease for 60s to one for the replicas -> primary replica
 - The primary replica picks a serial order for all mutations to the chunk
 - all replicas follow this order when applying mutations
- the primary can request extensions to master, grants are piggybacked on the HeartBeat messages.

master may sometimes try to revoke a lease before it expires
(snap shot, rename)

system interactions

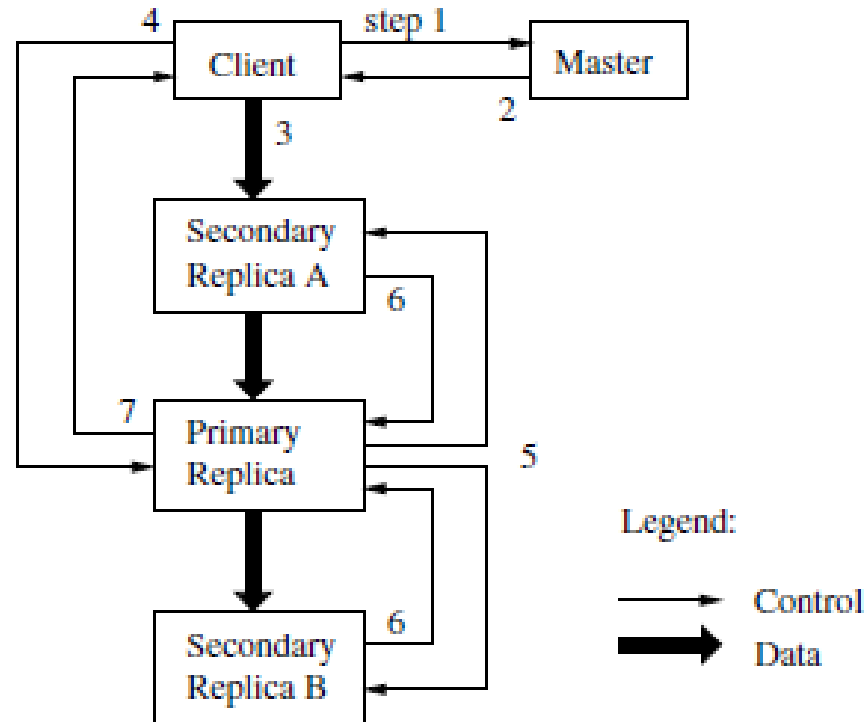
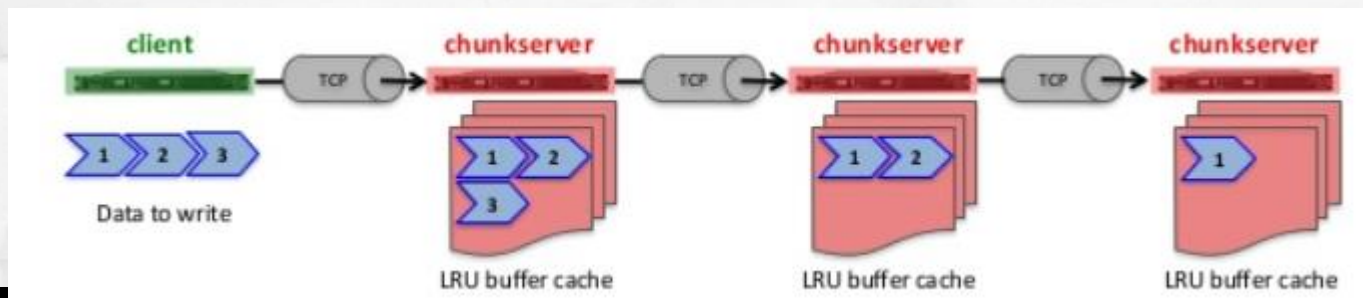


Figure 2: Write Control and Data Flow

system interactions

data flow

- Data is pushed linearly along a carefully picked chain of chunkservers in a TCP pipelined fashion
- Once a chunkserver receives some data, it starts forwarding immediately to the next chunkserver
- Each machine forwards the data to the closest machine in the network topology that has not received it



system interactions

snapshot operation: makes a copy of a file or a directory tree almost instantaneously

- Master revokes any outstanding leases on the chunks in the files to snapshot
- Master logs the snapshot operation to disk
- Master duplicate the metadata for the source file or directory tree: newly created snapshot files point to the same chunk as source files
- First time a client wants to write to a chunk C , it sends a request to the master to find current lease holder. The master notices that the reference count for chunk C is greater than one. It defers replying to the client request and instead picks a new chunk handle C' and ask each chunkserver that has a current replica of C to create a new chunk called C'

system interactions

namespace management and locking

- Read-lock : /dir1
 - Read-lock : /dir1/dir2
 - Read-lock : /dir1/dir2/dir3
 - Read-lock or Write-lock : /dir1/dir2/dir3/leaf
-
- File creation doesn't require write-lock on parent directory: read-lock on the name sufficient to protect the parent directory from deletion, rename, or snapshot
 - Locks are acquired in a consistent total order to prevent deadlock

creation, re-replication, balancing

- Balancing : Disk space utilization is below average
- Creation : Limit the number of recent files in each chunk. Because newer files have a high probability of significant write traffic and are only read-only at a later time.
- Re-replication : Distributing data to the rack.
The master replicates if the number of replicas available is less than the default.
Priority exists when replicating again
→ More replicas are damaged first. Chunk server distance

Garbage Collection

After a file is deleted, GFS does not immediately reclaim the available physical storage.

– Mechanism:

- Master logs the deletion like others changes
- File renamed to a hidden name that include the deletion time-stamp
- During the master's regular scan of the file system namespace, it removes any such hidden files if they have existed for more than three days.
- When the hidden file is removed from the namespace, its in-memory metadata is erased.
- In heartbeat message regularly exchange with the master, chunkserver reports a subset of the chunks it has, master replies with the identity of chunks not present in its metadata chunkserver is free to delete those chunks

advantage :

Split required resources, prevent accidental deletion

Stale replica detection

For each chunk, the master maintains a chunk version number to distinguish between up-to-date and stale replicas.

- Whenever master grants a new lease on a chunk, master increases the chunk version number and informs up-to-date replicas
- Master detect that chunkserver has a stale replica when the chunkserver
- restarts and reports its set of chunks and associated version numbers
- Master removes stale replica in its regular garbage collection



Fault Tolerance and Diagnosis

High Availability

fast Recovery

Both the master and the chunkserver are designed to re-store their state and start in seconds no matter how they terminated.

chunk Replication

each chunk is replicated on multiple chunkservers on different racks.

Fault Tolerance and Diagnosis

High Availability

Master Replication

Its operation log and checkpoints are replicated on multiple machines.

A mutation to the state is considered committed only after its log record has been flushed to disk locally and on all master replicas.

When it fails, it can restart almost instantly.

Moreover, "shadow" masters provide read-only access to the file system even when the primary master is down.

Fault Tolerance and Diagnosis

data integrity

- Each chunkserver uses checksumming to detect corruption of stored data.
- A chunk is broken up into 64 KB blocks. Each has a corresponding 32 bit checksum.
- the chunkserver verifies the checksum of data blocks that overlap the read range before returning any data to the requester, whether a client or another chunkserver.

not match the recorded checksum:

read from other replicas

master will clone the chunk from another replica

Fault Tolerance and Diagnosis

diagnostic Tools

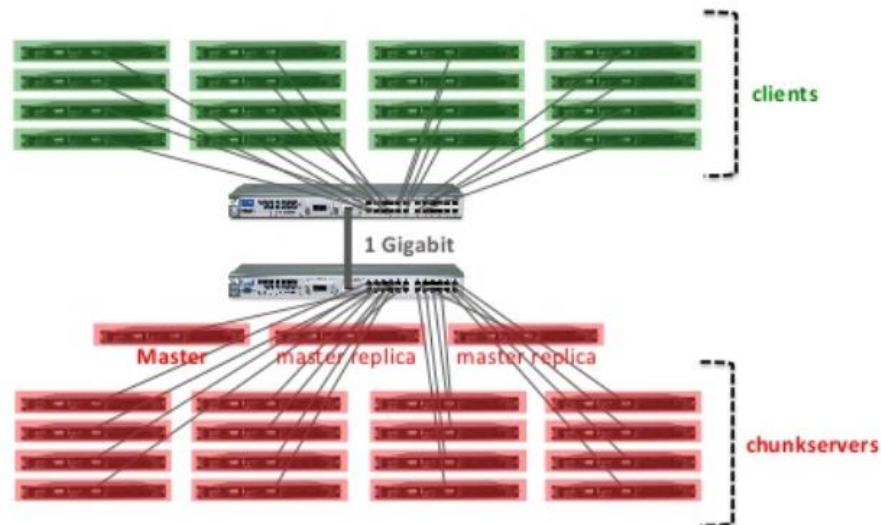
- logging has helped immeasurably in problem isolation, debugging, and performance analysis.
- The most recent events are also kept in memory and available for continuous monitoring.

Mesurements

Measurements (2003)

- **Micro-benchmarks: GFS CLUSTER**

- 1 **master**, 2 **master replicas**, 16 **chunkservers** with 16 **clients**
- Dual 1.4 GHz PIII processors, 2GB RAM, 2x80GB 5400 rpm disks, FastEthernet NIC connected to one HP 2524 Ethernet switch 24 ports 10/100 + Gigabit uplink

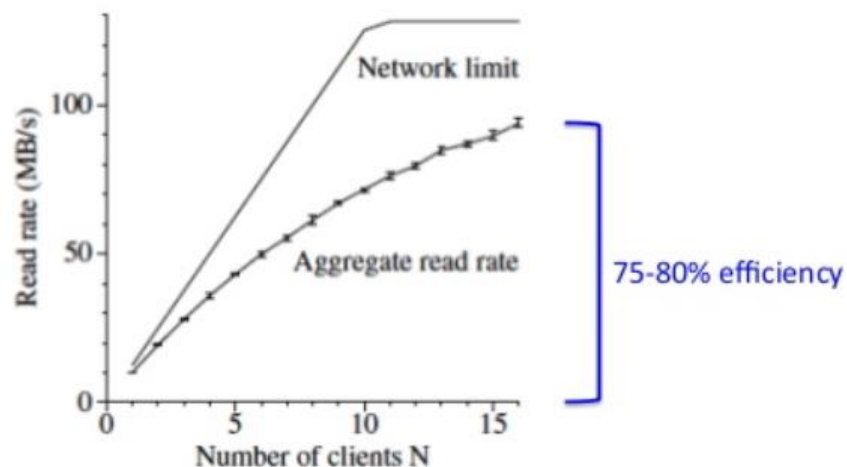


Measurements

Measurements (2003)

- **Micro-benchmarks: READS**

- Each client read a randomly selected 4MB region 256 times (= 1 GB of data) from a 320MB file
- Aggregate chunkserver memory is 32GB, so 10% hit rate in Linux buffer cache expected

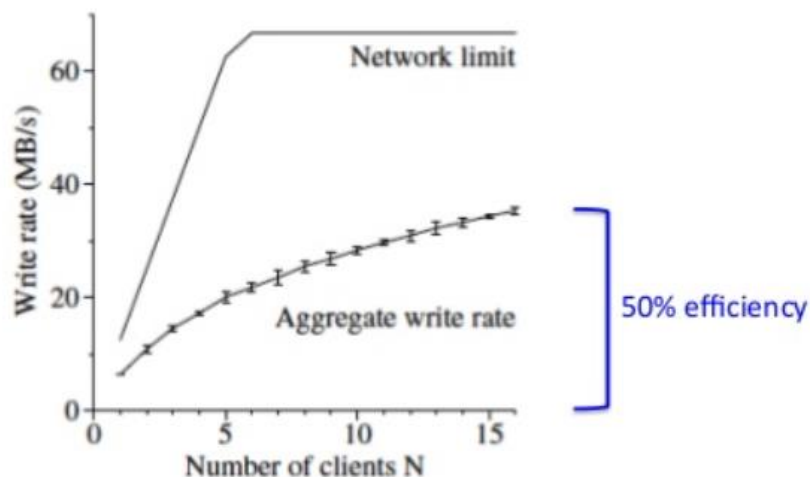


Mesurements

Measurements (2003)

- **Micro-benchmarks: WRITES**

- Each client writes 1 GB of data to a new file in a series of 1 MB writes
- Network stack does not interact very well with the pipelining scheme used for pushing data to the chunk replicas: network congestion is more likely for 16 writers than for 16 readers because each write involves 3 different replicas (→ see: write data flow)

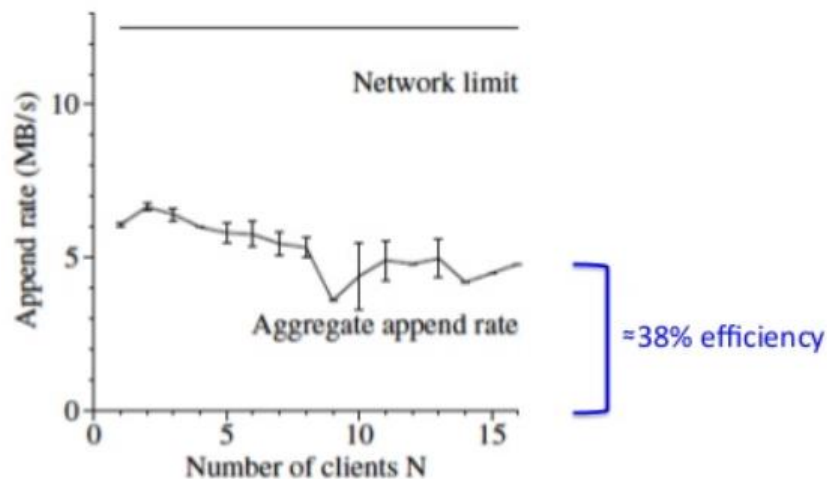


Mesurements

Measurements (2003)

- **Micro-benchmarks: RECORDS APPENDS**

- Each client appends simultaneously to a single file
- Performance is limited by the network bandwidth of the 3 chunkservers that store the last chunk of the file, independent of the number of clients



Mesurements

Measurements (2003)

- **Real world clusters: RECOVERY TIME**
 - **Kill 1 chunkserver in cluster B (Production)**
 - 15.000 chunks on it (= 600 GB of data)
 - Cluster limited to 91 concurrent chunk cloning (= 40% of 227 chunkservers)
 - Each clone operation consume at most 50 Mbps
 - **15.000 chunks restored in 23.2 minutes effective replication rate of 440 MB/s**
 - **Kill 2 chunkservers in cluster B (Production)**
 - 16.000 chunks on each (= 660 GB of data)
 - This double failure reduced 266 chunks to having a single replica... ☹
 - **These 266 chunks cloned at higher priority and were all restored to a least 2xreplication within 2 minutes**

conclusions

- **optimizing for huge files** that are mostly appended to and then read , and both extend and relax the standard file system interface to improve the overall system.
- system provides fault tolerance by constant monitoring, replicating crucial data, and **fast and automatic recovery**.
- **using check to detect data** corruption at the disk.
- system achieve this **by separating file system control**, which passes through the master, from data transfer, which passes directly between chunkservers and clients.

Q & A