COMP5349– Cloud Computing

Week 3: Google Distributed File System and HDFS

A/Prof Uwe Roehm
School of Information Technologies

Outline

- **MapReduce/Hadoop brief Intro**
- GFS Overall Architecture
- Scalability and Consistency Strategies
- High Availability Strategy
- Measurements
- HDFS

The presentation is based on:
❑ Sanjay Ghemawat, Howard Gobioff and Shun-Tak Leung, *The Google File System*. In Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03),2003
❑ Yahoo Hadoop Tutorial Module 2: The Hadoop Distributed File System, accessible from [http://developer.yahoo.com/hadoop/tutorial/module2.html]

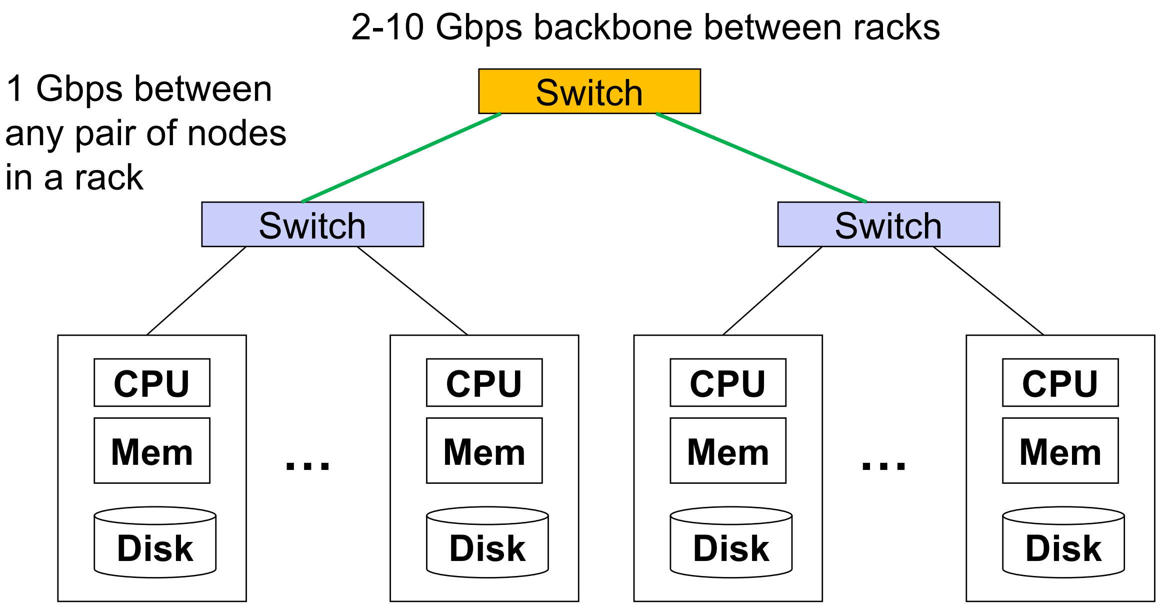
Dealing with Big Data

- “Big Data are high-**volume**, high-**velocity**, and/or high-**variety** information assets that require **new forms of processing** to enable enhanced decision making, insight discovery and process optimization.
  - [Gartner 2012 report]
- Some big data numbers
  - Facebook data in 2010 [http://www.facebook.com/note.php?note_id=409881258919]
    - 500 million active users
    - 100 billion hits per day
    - 50 billion photos
    - 2 trillion objects cached, with hundreds of millions of requests per second
    - 130TB of logs every day
  - Facebook garnered more than 1 trillion page views per month in June and July (2011), according to data from DoubleClick.

    [http://mashable.com/2011/08/24/facebook-1-trillion-pageviews/]
    - 23,148 views per minute
- Planet scale computing

Cluster of Commodity Linux Nodes

Typically a two-level architecture


2-10 Gbps backbone between racks
1 Gbps between any pair of nodes in a rack
Switch
Switch
Switch
CPU
Mem
Disk
…

Each rack contains 16-64 nodes

Anand Rajaraman and Jeff Ullman Map-Reduce lecture, CS345A, Winter 2009: Data Mining, Standford

New Software Stack

- Stable storage (a new type of distributed file system)
  - GFS/HDFS (Week 3)
- Higher-level programming framework
  - MapReduce/Hadoop (Week 4, 5, 6)
    - Basic framework
    - Data analytic extension
  - “An new abstraction … to express computation … but hides the messy details of parallelization, fault-tolerance, data distribution and load balancing in a library”
    - Jeffrey Dean and Sanjay Ghemawat, *MapReduce: Simplified Data Processing on Large Clusters*. In OSDI'04.
  - Spark, Flink (Week 7, 8)
- Flexible Structure Storage ('NoSQL')

How Big are the Hadoop Clusters: Facebook

- Facebook
  - "We use Hadoop to store copies of internal log and dimension data sources and use it as a source for reporting/analytics and machine learning."
  - "Currently we have 2 major clusters:
    - A 1100-machine cluster with 8800 cores and about 12 PB raw storage.
    - A 300-machine cluster with 2400 cores and about 3 PB raw storage.
    - Each (commodity) node has 8 cores and 12 TB of storage.
    - We are heavy users of both **streaming** as well as the **Java APIs**. We have built a higher level data warehousing framework using these features called **Hive"**

[http://wiki.apache.org/hadoop/PoweredBy]

How Big are the Hadoop Clusters: Yahoo!

- Yahoo!
  - More than 100,000 CPUs in >40,000 computers running Hadoop
  - Our biggest cluster: 4500 nodes (2*4cpu boxes w 4*1TB disk & 16GB RAM)
    - Used to support research for Ad Systems and Web Search
    - Also used to do scaling tests to support development of Hadoop on larger clusters
    - >60% of Hadoop Jobs within Yahoo are **Pig** jobs.

[http://wiki.apache.org/hadoop/PoweredBy]

Outline

- **MapReduce/Hadoop brief Intro**
- **GFS Overall Architecture**
- **Scalability and Consistency Strategies**
- **High Availability Strategy**
- **Measurements**
- **HDFS**

Distributed File System Basics

- A distributed file systems allow clients to access files on remote servers "transparently".
- Example: We are using at least two popular ones in SIT
  - Your home directory is mounted on all Linux servers using **NFS**
    - Physically located in soitlabhomes01.it.usyd.edu.au
    - Mounted on all Linux servers: ucpu1.ug.it.usyd.edu.au, ucpu2.ug.it.usyd.edu.au, soit-ucpu-pro-1.ucc.usyd.edu.au, etc…
  - Your home directory is mapped as U drive in Windows using **Samba**
- Constraints
  - No sufficient mechanism in terms of reliability and availability
    - Replica, migration, etc
    - You may have encounter login hiccups like your U drive is not mapped when you login to one of the lab machine…
    - There was a power issue in the server room during the semester break and all lab machines cannot be logged in

Google File System (GFS) - Design Constraints -

[Ghemawat/Gobioff/Leung, SOSP03]

- Component failures are the norm
  - 1000s of components
  - Bugs, human errors, failures of memory, disk, connectors, networking, and power supplies
  - Monitoring, error detection, fault tolerance, automatic recovery
- Files are huge by traditional standards
  - Multi-GB files are common
  - Billions of objects
- Workload features (Co-design of application and the file system)
  - Most files are mutated by appending new data rather than overwriting existing data.
  - Files are often read sequentially rather than randomly.

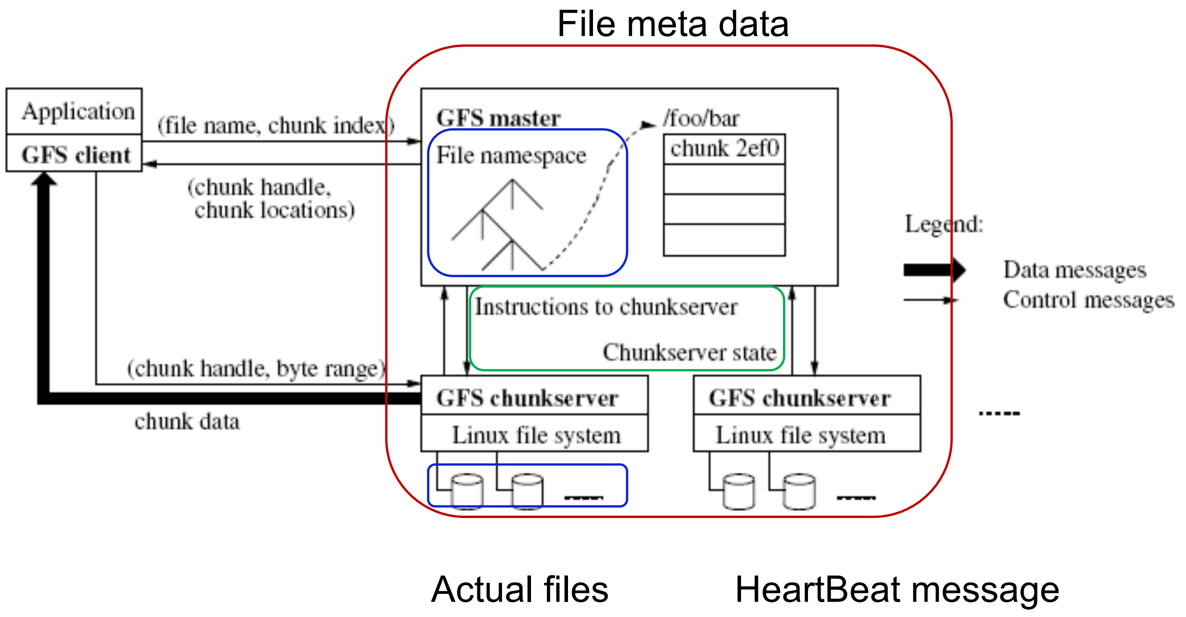
Master/Slave Architecture

- A GFS cluster
  - A **single** *master*
    - Maintains all **metadata**
      - Name space, access control, file-to-chunk mappings, garbage collection, chunk migration
    - Periodically communicates with chunkservers in *HeartBeat* messages.
  - **Multiple** *chunkservers* per master
    - Store actual file data
  - Running on commodity Linux machines

Master/Slave Architecture (cont'd)

- A file
  - Divided into fixed-sized *chunks*
    - Default chunk size is 64 MB
    - Labeled with 64-bit unique global IDs (chunk handle)
    - Stored at *chunkservers*
    - Chunk is replicated on multiple chunkservers (by default 3 replicas)
      - files that needs frequent access will have a high replication factor
- GFS clients
  - Consult master for **metadata**
  - Access data from **chunk servers**

Architecture diagram



[Ghemawat/Gobioff/Leung, SOSP03]

Single-Master Design

- Benefits:
  - Simple, master can make sophisticated chunk placement and replication decisions using **global** knowledge
- Possible disadvantage:
  - Single point of failure (availability)
  - Bottleneck (scalability)
- Solution
  - Replicate master states on multiple machines; Chubby is used to point a master among several shadow masters
  - Fast recovery;
  - Clients use master only for metadata, not reading/writing actual file

Metadata

- Three types:
  - File and chunk namespaces
  - Mapping from files to chunks
  - Locations of chunk replicas
- All metadata is in memory.
  - Large chunk size (64MB) ensures small meta data size
  - First two are kept persistent in an *operations log* for recovery.
  - Third is obtained by querying chunkservers at startup and periodically thereafter.
    - Chunkservers may come and go or have partial disk failure

Operation Log and Master Recovery

- Metadata updates are logged
  - Create new directory, new files
  - Log replicated on remote machines
- Take global snapshots (checkpoints) to truncate logs
  - Memory mapped (no serialization/de-serialization)
  - Checkpoints can be created while updates arrive
- Recovery
  - Latest checkpoint + subsequent log files

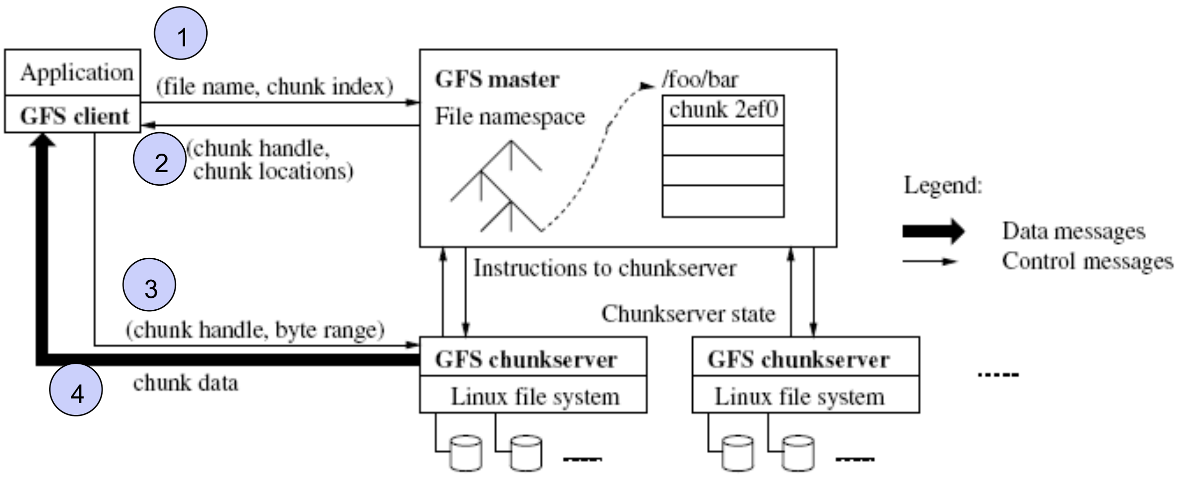
Outline

- MapReduce/Hadoop brief Intro
- GFS Overall Architecture
- **Scalability and Consistency Strategies**
- High Availability Strategy
- Measurements
- HDFS

Scalability

- Single Master is the only possible bottleneck
  - Minimize Master storage requirement for each file
  - Minimize Master involvement in read/write operation
    - Number of messages
    - Size of messages
- Design
  - Master is only involved at the beginning of Read/Write operation
  - Only limited chunk information is transferred
  - Actual data movement does not involve master
- A single master is able to server thousands of chunk servers

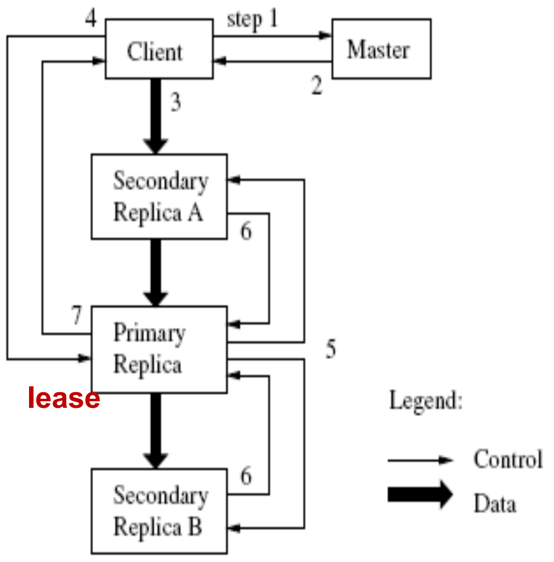
GFS Read



- Client translates file name and byte offset to chunk index.
- Sends request to master.
- Master replies with chunk handle and location of replicas.
- Client caches this info.
- Client sends request to a close replica, specifying chunk handle and byte range.
- The chunk server sends chunk data to the client

[Ghemawat/Gobioff/Leung, SOSP03]

GFS Write



1. The client asks the master which chunkserver holds the current lease for the chunk and the locations of the other replicas.

   Grant a new lease if no one holds one
2. The master replies with the identity of the primary and the locations of the other (*secondary*) replicas

   Cached in the client
3. The client pushes the data to all the replicas
4. Once all the replicas have acknowledged receiving the data, the client sends a write request to the primary. The primary assigns consecutive serial numbers to all the mutations it receives, possibly from multiple clients, which provides the necessary serialization.
5. The Primary forwards the write request to all secondary replicas.
6. Secondaries signal completion.
7. Primary replies to client. Errors handled by retrying.

[Ghemawat/Gobioff/Leung, SOSP03]

Write Operation's Data Flow

- Data flow is decoupled from control flow
- Data is pushed from the client to a chain of chunkservers
  - Each machine selects the “closest” machine to forward the data
    - Client pushes data to chunkserver **S1** that has **replica A** and tells it there are two more chunkservers: **S2** and **S3** in the chain
    - **S1** pushes the data to **S2**, which is closer than **S3** and tells it **S3** is in the chain. **S2** has primary replica.
    - **S2** pushes the data to **S3** and tells it you are the last one. **S3** has **replica B**
- The network topology is simple enough that “distances” can be accurately estimated from IP addresses
- Data is pushed in a pipelined fashion
  - Each trunk server starts forwarding immediately after it receives some data

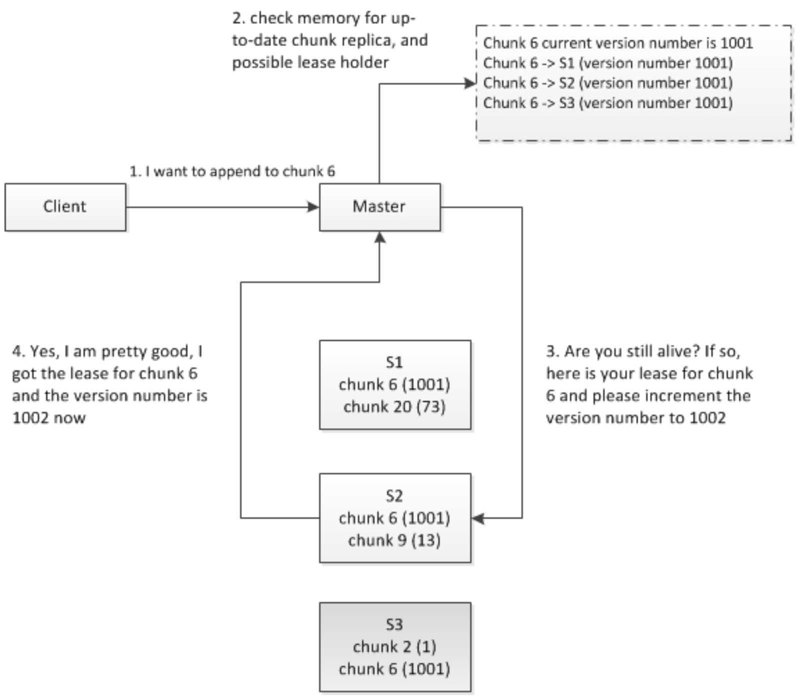
Lease and Mutation Order

- A mutation is an operation that changes the contents or metadata of a chunk
- The master grants a chunk lease to a replica
- The replica holding the lease becomes the primary replica; it determines the order of updates to all replicas
- Lease
  - 60 second timeouts
  - Can be extended indefinitely
  - Extension request are piggybacked on heartbeat messages
  - After an old lease expires, the master can grant new leases
- Any replica may become primary to coordinate the mutation process

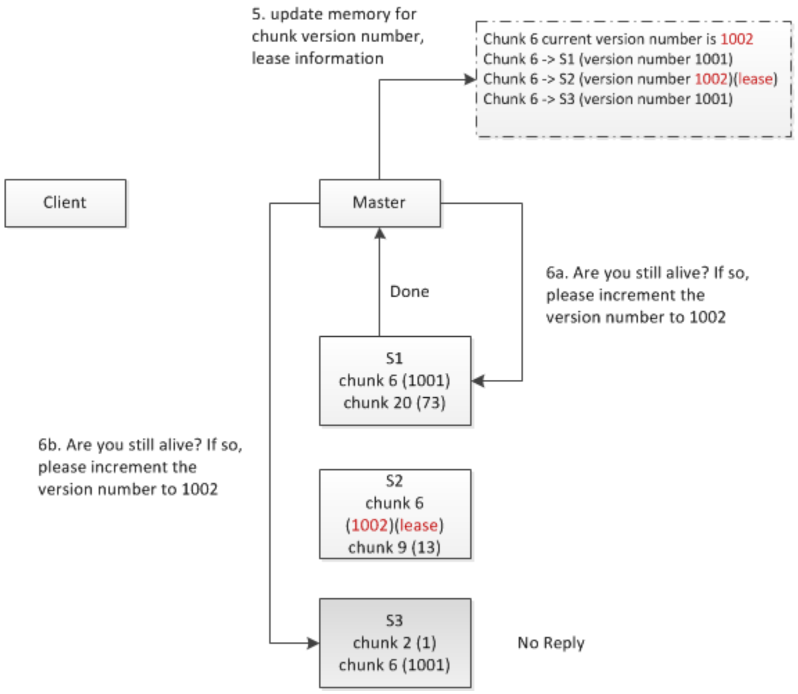
How Consistency is Maintained

- Applying mutations to a chunk in the same order on all its replicas
  - Managed by primary replicas
- Using chunk version numbers to detect any replica that has become stale because it has missed mutations while its chunkserver was down
  - Whenever the master grants a new lease on a chunk, it increases the chunk version number and informs the up-to-date replicas. They all persist this information.
  - If a replica is not available at the time, its version number won't be advanced, which is an indication of stale replica. **Why?**
  - A restarted chunk server needs to report to master about its chunks and version numbers.
  - During read, when client asks for trunk information, master will simply ignore stale replica, it won't be returned to client
  - Master removes stale replicas in regular garbage collection

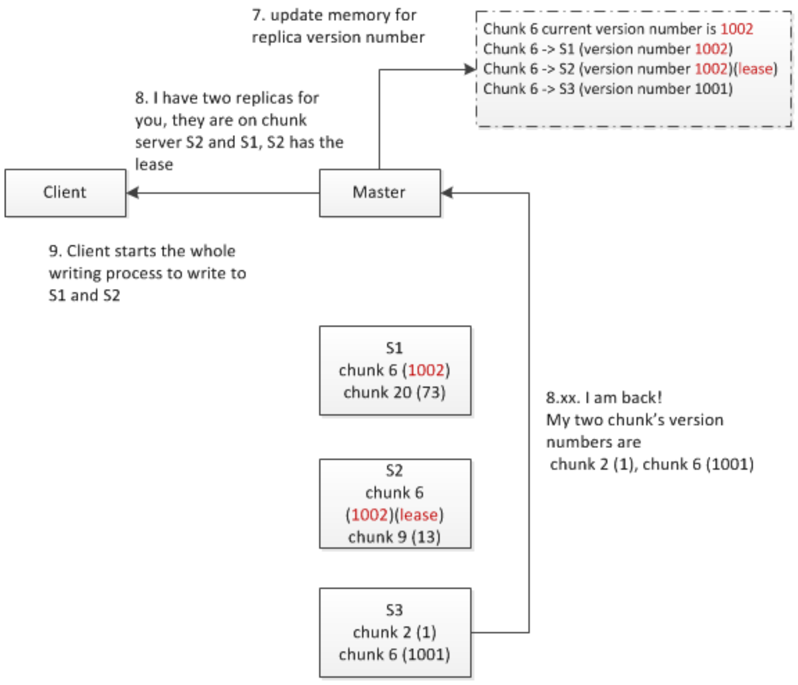
Chunk Version Number Mechanism(1)


2. check memory for up-to-date chunk replica, and possible lease holder
Chunk 6 current version number is 1001
Chunk 6 -> S1 (version number 1001)
Chunk 6 -> S2 (version number 1001)
Chunk 6 -> S3 (version number 1001)
1. I want to append to chunk 6
Client
Master
4. Yes, I am pretty good, I got the lease for chunk 6 and the version number is 1002 now
3. Are you still alive? If so, here is your lease for chunk 6 and please increment the version number to 1002
S1
chunk 6 (1001)
chunk 20 (73)
S2
chunk 6 (1001)
chunk 9 (13)
S3
chunk 2 (1)
chunk 6 (1001)

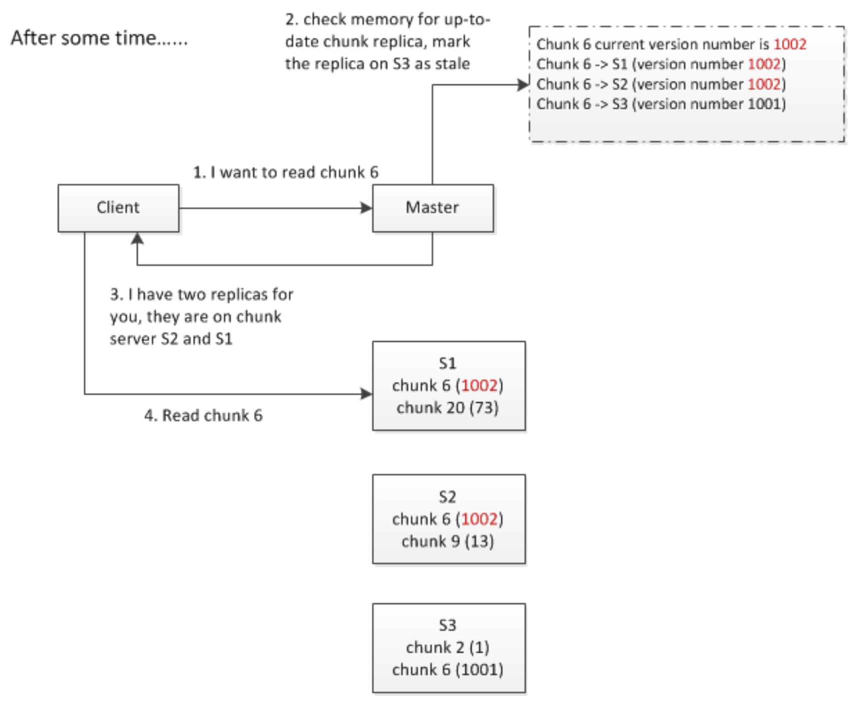
Chunk Version Number Mechanism(2)


5. update memory for chunk version number, lease information
Chunk 6 current version number is 1002
Chunk 6 -> S1 (version number 1001)
Chunk 6 -> S2 (version number 1002)(lease)
Chunk 6 -> S3 (version number 1001)
Client
Master
Done
6a. Are you still alive? If so, please increment the version number to 1002
S1 chunk 6 (1001) chunk 20 (73)
6b. Are you still alive? If so, please increment the version number to 1002
S2 chunk 6 (1002)(lease) chunk 9 (13)
S3 chunk 2 (1) chunk 6 (1001)
No Reply

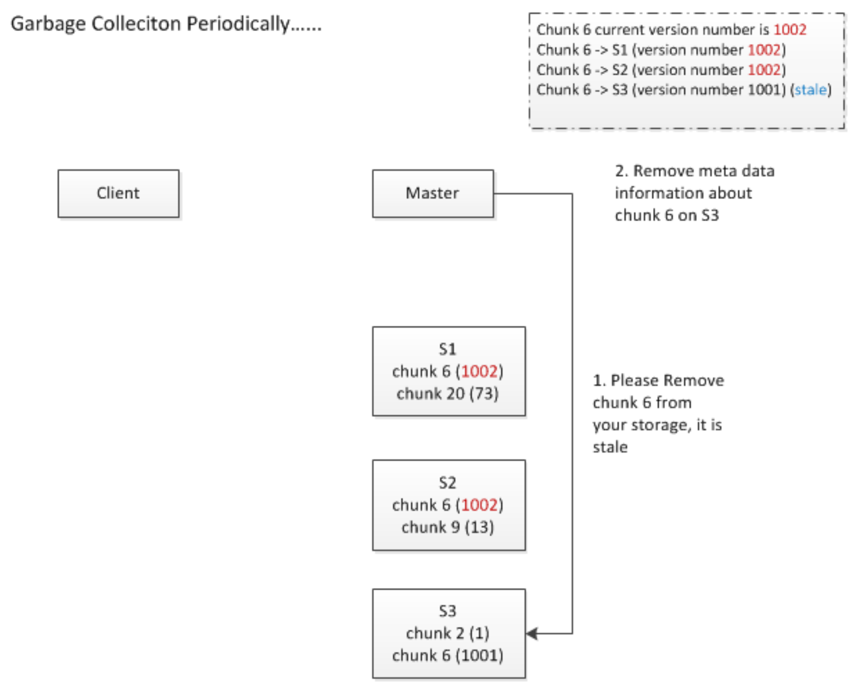
Chunk Version Number Mechanism(3)


7. update memory for replica version number
Chunk 6 current version number is 1002
Chunk 6 -> S1 (version number 1002)
Chunk 6 -> S2 (version number 1002)(lease)
Chunk 6 -> S3 (version number 1001)
8. I have two replicas for you, they are on chunk server S2 and S1, S2 has the lease
Client
Master
9. Client starts the whole writing process to write to S1 and S2
S1 chunk 6 (1002) chunk 20 (73)
8.xx. I am back! My two chunk's version numbers are chunk 2 (1), chunk 6 (1001)
S2 chunk 6 (1002)(lease) chunk 9 (13)
S3 chunk 2 (1) chunk 6 (1001)

Chunk Version Number Mechanism(4)


After some time......
2. check memory for up-to-date chunk replica, mark the replica on S3 as stale
Chunk 6 current version number is 1002
Chunk 6 -> S1 (version number 1002)
Chunk 6 -> S2 (version number 1002)
Chunk 6 -> S3 (version number 1001)
1. I want to read chunk 6
Client
Master
3. I have two replicas for you, they are on chunk server S2 and S1
4. Read chunk 6
S1
chunk 6 (1002)
chunk 20 (73)
S2
chunk 6 (1002)
chunk 9 (13)
S3
chunk 2 (1)
chunk 6 (1001)

Chunk Version Number Mechanism(5)


Garbage Colleciton Periodically......
Chunk 6 current version number is 1002
Chunk 6 -> S1 (version number 1002)
Chunk 6 -> S2 (version number 1002)
Chunk 6 -> S3 (version number 1001) (stale)
Client
Master
2. Remove meta data information about chunk 6 on S3
1. Please Remove chunk 6 from your storage, it is stale
S1
chunk 6 (1002)
chunk 20 (73)
S2
chunk 6 (1002)
chunk 9 (13)
S3
chunk 2 (1)
chunk 6 (1001)

Outline

- MapReduce/Hadoop brief Intro
- GFS Overall Architecture
- Scalability and Consistency Strategies
- **High Availability Strategy**
- Measurements
- HDFS

High Availability (HA) Strategies

- Any server may be down/unavailable at a given time
- HA of the overall system relies on two simple strategies
  - fast recovery
    - restore states quickly when a server (master or chunk) is back to life
  - replication
    - storing extra copies of data

Fast Recovery

- Small meta data
- Chunkservers maintain
  - Checksums for 64kb blocks of data for **data integrity check**
  - Replica/chunk version number for **data freshness check**
- Master maintain
  - Data information (owner, permission, mapping, etc)
  - Memory image and short operation log enable fast recovery
- It takes only a few seconds to read this metadata from disk before the server is able to answer queries.
- Master may wait a little longer to receive all chunk location information.

Master Replication

- Master operation log and checkpoints are replicated on multiple machine.
- If the master fails, monitoring infrastructure outside GFS starts a new master process elsewhere (Chubby lock service which utilizes Paxos algorithms to elect a new master among a group of living shadow masters).
- Shadow masters provides read-only access when the primary master is down.

Chunk replica

- Chunk replica is created within the cluster
- Master is responsible for chunk replica creation and placement
  - Decide where to put replica
  - Decide if a chunk needs new replica
  - Decide if a replica needs to migrate to a new location

Chunk Replica Placement

- Goal
  - Maximize data reliability and availability
  - Maximize network bandwidth
- Need to spread chunk replicas across machines and racks
  - Read can exploit the aggregate bandwidth of multiple racks (read only needs to contact one replica)
  - Write has to flow through multiple racks (write has to be pushed to all replicas)

Creating, Re-replication, Rebalancing

- Replicas created for three reasons:
  - Chunk creation, Re-replication, Load balancing
- Creation location consideration
  - Balance disk utilization
  - Balance creation events
    - After a creation, lots of traffic, especially write
  - Spread replicas across racks
- Re-replication
  - Occurs when number of replicas falls below a user-specified number
    - Replica corrupted, chunkserver down, replication number increased.
  - Replicas prioritized, then rate-limited (to reduce impact on client traffic).
  - Placement heuristics similar to that for creation.
- Rebalancing
  - Periodically examines distribution and moves replicas around.

Outline

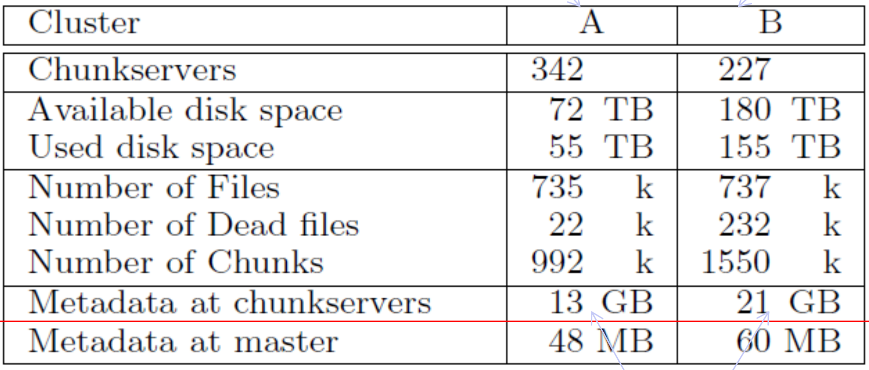
- Distributed File System basics
- GFS Overall Architecture
- Scalability and Consistency Strategies
- High Availability Strategy
- **Measurements**
- HDFS

Real world cluster observation

Replication factor is 3

Research cluster

Production cluster

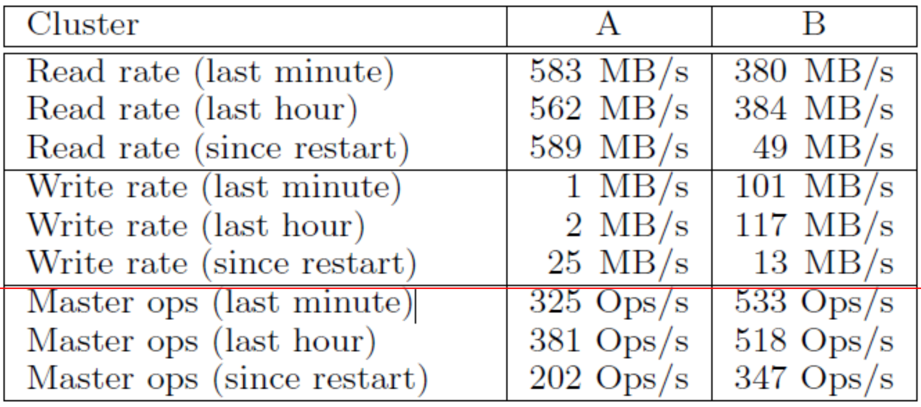


Cluster	A	B
Chunkservers	342	227
Available disk space	72 TB	180 TB
Used disk space	55 TB	155 TB
Number of Files	735 k	737 k
Number of Dead files	22 k	232 k
Number of Chunks	992 k	1550 k
Metadata at chunkservers	13 GB	21 GB
Metadata at master	48 MB	60 MB

Very small meta data on master

Individual chunkserver meta data is also small

Read rate, write rate and master load



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

Master is not a bottleneck.

Recovery Time

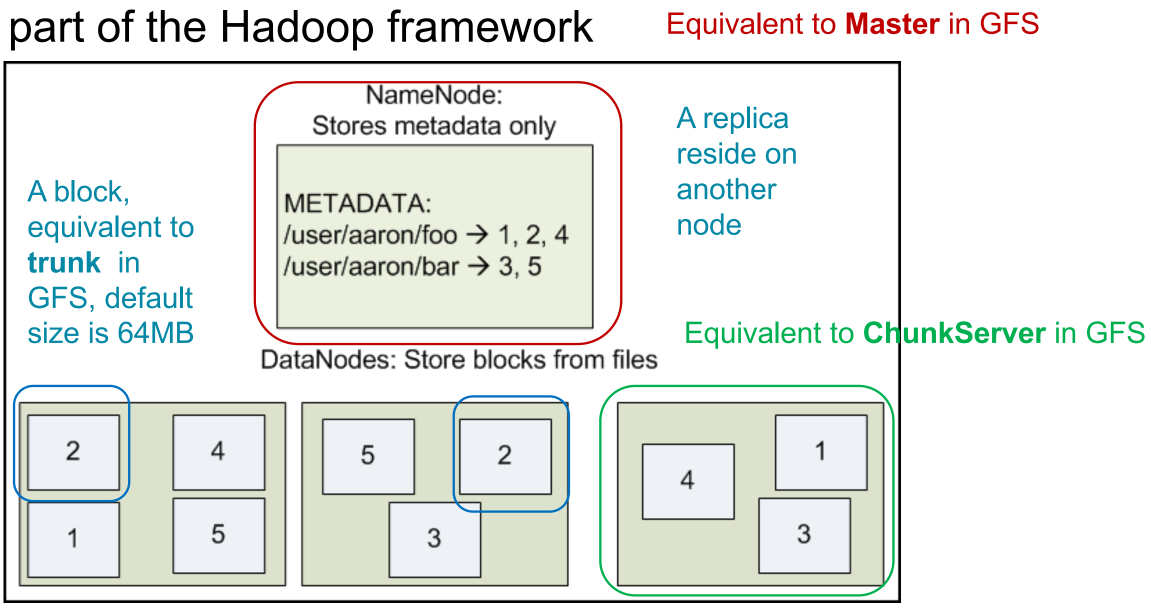
- Recovery Time (in Cluster B)
  - A single chunkserver was killed.
    - The chunkserver had 15,000 chunks containing 600GB of data.
    - The cluster needs to clone those chunks on other servers
    - A maximum of 91 concurrent clone operations are allowed
    - All chunks were restored in 23.2 minutes.
  - Two chunkservers were killed.
    - The chunkservers had 32,000 chunks, roughly 1320GB of data
    - 266 chunks had no replicas.
    - These chunks were cloned to have at least 2 replicas within 2 minutes.

Outline

- Distributed File System basics
- GFS Overall Architecture
- Scalability and Consistency Strategies
- High Availability Strategy
- Measurements
- **HDFS**

HDFS

- **Hadoop Distributed File System** is the open source implementation of GFS
- It is part of the Hadoop framework


Equivalent to **Master** in GFS
NameNode: Stores metadata only
METADATA:
/user/aaron/foo → 1, 2, 4
/user/aaron/bar → 3, 5
A block, equivalent to **trunk** in GFS, default size is 64MB
A replica reside on another node
Equivalent to **ChunkServer** in GFS
DataNodes: Store blocks from files
2
4
1
5
5
2
3
4
1
3

http://developer.yahoo.com/hadoop/tutorial/module2.html

GFS vs. HDFS

GFS	HDFS
Naming	
Master	NameNode
Chunk Server	DataNode
Chunk	Block
Functionality	
Support random write and record append. In practical workload, the number of random write is very small	Support Write-Once-Read-Many workload
HA strategy requires an external monitoring services, such as Chubby lock service for Master recovery	HA configuration is only available in the latest version Secondary name node does part of the job GFS master is supposed to do. It is not a shadow master

Our Hadoop Cluster

- http://soit-hdp-pro-1.ucc.usyd.edu.au:50070

Interacting with HDFS

- HDFS runs in a **separate namespace**, isolated from the contents of the local files
- HDFS shell commands are similar to standard Linux command
  - hdfs dfs –ls *dirName*
  - hdfs dfs –mkdir *dirName*
  - hdfs dfs –rm *file*
  - *etc..*
- Most of the time we use HDFS together with MapReduce
  - Moving computation to data
  - By default, one block/chunk represents one computing unit

Summary

- **MapReduce/Hadoop**
  - Basic terminologies
  - Hadoop/MapReduce clusters in large companies
- **Introduction to GFS/HDFS**
  - Master/Slave architecture
  - Special consideration for single master design (scalability)
    - Minimize master data/operation/contact
    - Manage master election in an external system
  - Consistency strategy
    - Replica primary control write order
    - Version number control replica freshness
  - HA strategies
    - Fast Recovery
    - Replication

Next Week

- Topic
  - MapReduce Framework
- Reading
  - **Jeffrey Dean and Sanjay Ghemawat, Map Reduce: Simplified Data Processing on Large Clusters, Commun. ACM 51, 1 (January 2008), 107-113**