

CS535 BIG DATA

PART 1. BATCH COMPUTING MODELS FOR BIG DATA ANALYTICS
1. DISTRIBUTED MODEL FOR SCALABLE
BATCH COMPUTING
– DISTRIBUTED FILE SYSTEMS

Sangmi Lee Pallickara
Computer Science, Colorado State University
<http://www.cs.colostate.edu/~cs535>

9/22/2016 CS535 Big Data - Fall 2016 W5.A.1

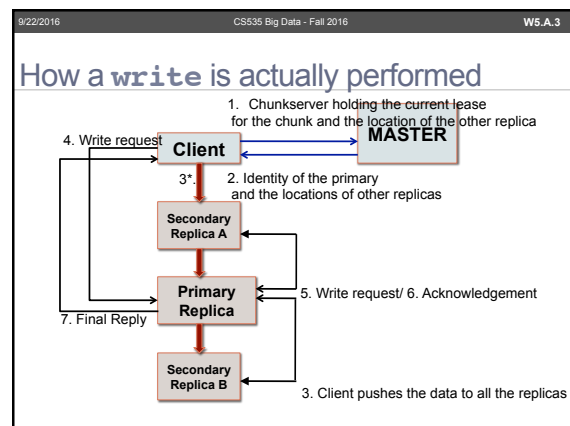
FAQs

- Questions about PA1
 - Send an email to cs535@cs.colostate.edu
- FAQ for PA1 page is available at:
 - http://www.cs.colostate.edu/~cs535/FAQ_PA1.html
- Wikibomb
 - Please see the description of PA1
- Term Project
 - Google computing cluster credit is available
 - Optional

9/22/2016 CS535 Big Data - Fall 2016 W5.A.2

Objectives

- Distributed File System
 - GFS, GFS2 (Colossus)



9/22/2016 CS535 Big Data - Fall 2016 W5.A.4

Data flow is decoupled from the control flow to utilize network efficiently

- Utilize each machine's network bandwidth
- Avoid network bottlenecks
- Avoid high-latency links
- **Leverage** network topology
 - Estimate distances from IP addresses
- Pipeline the data transfer
 - Once a chunkserver receives some data, it starts forwarding immediately.
 - For transferring B bytes to R replicas
 - Ideal elapsed time will be $\approx B/T + RL$ where:
 - T is the network throughput
 - L is latency to transfer bytes between two machines

9/22/2016 CS535 Big Data - Fall 2016 W5.A.5

Distributed File Systems

Google File System

9/22/2016 CS535 Big Data - Fall 2016 W5.A.6

Managing Mutations: Handling writes and appends to a file

9/22/2016 CS535 Big Data - Fall 2016 W5.A.7

Append: Record sizes and fragmentation

- Size is restricted to $\frac{1}{4}$ the chunk size
 - Maximum size
- Minimizes worst-case fragmentation
 - Internal fragmentation in each chunk ...

9/22/2016 CS535 Big Data - Fall 2016 W5.A.8

Inconsistent Regions

User will re-try to store Data 3

9/22/2016 CS535 Big Data - Fall 2016 W5.A.9

What if record append fails at one of the replicas

- Client **must retry** the operation
- Replicas of same chunk may contain
 - Different data
 - Duplicates** of the same record
 - In whole or in part
- Replicas of chunks are **not bit-wise identical!**
 - In most systems, replicas are identical

9/22/2016 CS535 Big Data - Fall 2016 W5.A.10

GFS **only guarantees** that the data will be written **at least once** as an atomic unit

- For an operation to return **success**
 - Data must be **written at the same offset** on **all** the replicas
- After the write, all replicas are **as long as** the end of the record
 - Any future record will be assigned a higher offset or a different chunk

9/22/2016 CS535 Big Data - Fall 2016 W5.A.11

GFS client code implements the file system API

- Communications with master and chunk servers done **transparently**
 - On behalf of apps that read or write data
- Interact with master for **metadata**
- Data-bearing** communications directly to chunk servers

9/22/2016 CS535 Big Data - Fall 2016 W5.A.12

Creating Snapshots

9/22/2016 CS535 Big Data - Fall 2016 W5.A.13

Snapshots

- Copying file or directory tree almost instantaneously
- Minimizing any interruptions of ongoing mutations
- Providing checkpoint
 - Users can commit later
 - Rollback

9/22/2016 CS535 Big Data - Fall 2016 W5.A.14

Snapshots allow you to make a copy of a file very fast

1. Master **revokes** outstanding leases for any chunks of the file (source) to be snapshot
2. **Log** the operation to disk
3. Update in-memory state
 - Duplicate metadata of the source file
4. Newly created file points to the "**same chunks**" as the source

9/22/2016 CS535 Big Data - Fall 2016 W5.A.15

When a client wants to write to a chunk C after the snapshot operation

- Master sees "the **reference count** to C > 1"
- Pick **new chunk-handle C'**
- Ask chunk-server with current replica of C
 - Create new chunk C'
 - Data is **copied locally, not over the network**
- From this point, chunk handling of C' is no different

9/22/2016 CS535 Big Data - Fall 2016 W5.A.16

GFS does not have a per-directory structure that lists files in the directory

- Name spaces represented as a **lookup** table
 - Maps **full pathnames** to metadata
- Each node has an associated read/write lock
- File creation does not require a lock **on the directory structure**
 - No **inode** needs to be protected from modification

9/22/2016 CS535 Big Data - Fall 2016 W5.A.17

Each master operation acquires a set of locks before it runs

- If operation involves **/d1/d2/.../dn/leaf**
 - Acquire read locks on directory names
 - /d1, /d1/d2, ..., /d1/d2/.../dn
 - Read or write lock on full pathname
 - /d1/d2/.../dn/leaf
- **Read lock** prevents a directory from **being deleted, renamed, or snapshotted**
- **Write lock** on file names serialize attempts to create a file with the same twice

9/22/2016 CS535 Big Data - Fall 2016 W5.A.18

Locks are used to prevent operations during snapshots

- How do we prevent creating `/home/user/foo`
 - While `/home/user` is being snapshotted to `/save/user` ?
- `/home/user` is being snapshotted to `/save/user`
 - Read lock on `/home` and `/save`
 - Write lock on `/home/user` and `/save/user`
- To create file
 - Read lock on `/home` and `/home/user`
 - Write lock on `/home/user/foo`
- The two operations will be serialized
 - because they try to obtain `/home/user`
- File creation does not require write lock on parent directory ... there is no "directory"
 - Read locks on `/home` and `/home/user`
 - Write lock on `/home/user/foo`

9/22/2016 CS535 Big Data - Fall 2016 W5.A.19

Deletion of Files and Garbage Collection

9/22/2016 CS535 Big Data - Fall 2016 W5.A.20

Garbage collection in GFS

- After a file is deleted, GFS does not reclaim space immediately
- Done **lazily** during garbage collection at
 - File and chunk levels

9/22/2016 CS535 Big Data - Fall 2016 W5.A.21

Master logs a file's deletion immediately

- File is **renamed to a hidden name**
 - Includes deletion timestamp
- Master scans the file system namespace
 - Delete if hidden file existed for more than 3 days
- When file is removed from namespace
 - *In memory metadata* is also removed
 - **Severs links** to all its chunks!

9/22/2016 CS535 Big Data - Fall 2016 W5.A.22

Garbage collection: When Master scans its chunk namespace

- Identifies **orphaned chunks**
 - Not reachable from any file
- Erase metadata for these chunks

9/22/2016 CS535 Big Data - Fall 2016 W5.A.23

The role of heart-beats in garbage collection

- Chunk server reports subset of chunks it currently has
- Master replies with identity **of chunks no longer present**
 - Chunk server is now free to delete its replica of such chunks

9/22/2016 CS535 Big Data - Fall 2016 W5.A.24

Stale chunks and issues

- If a chunk server fails
 - AND misses mutations to the chunk
 - The chunk replica becomes **stale**
- Working with a **stale replica** causes problems with:
 - Correctness
 - Consistency

9/22/2016 CS535 Big Data - Fall 2016 W5.A.25

Aiding the detection of stale chunks

- Master maintains a **chunk version number** for each chunk
 - Distinguish between stale and up-to-date chunks
- When master grants a new lease on chunk
 - Increase version number
 - Inform replicas
 - Record new version persistently

Occurs BEFORE any client can write to chunk

9/22/2016 CS535 Big Data - Fall 2016 W5.A.26

If a replica is unavailable its **version number** will not be advanced

- When a chunk server restarts, it reports to the Master with the following:
 - Set of Chunks
 - Corresponding version numbers
- Used to **detect** stale replicas
- **Remove stale replicas** in regular garbage collection

9/22/2016 CS535 Big Data - Fall 2016 W5.A.27

Additional safeguards against stale replicas

- Include chunk version number
 - When **client requests** chunk information
 - Client/Chunk server verify version to make sure things are up-to-date
 - During cloning operations
 - Clone the most up-to-date chunk
- Clients and chunk servers expected to **verify** versioning information

9/22/2016 CS535 Big Data - Fall 2016 W5.A.28

Data Integrity

9/22/2016 CS535 Big Data - Fall 2016 W5.A.29

Data Integrity

- **Impractical** to detect chunk corruptions **across replicas**
 - Not bitwise identical in any case!
- Detection of corruption should be **self-contained**

9/22/2016 CS535 Big Data - Fall 2016 W5.A.30

Data Integrity

- Break chunks into **64 KB** data blocks
- Compute 32-bit checksum for block
 - Keep in chunk server memory
 - Store persistently, **separate** from the data
- Verify checksums of data blocks that **overlap** read range

9/22/2016 CS535 Big Data - Fall 2016 W5.A.31

Inefficiencies

9/22/2016 CS535 Big Data - Fall 2016 W5.A.32

The master server is a single point of failure

- Master server **restart** takes several seconds
 - Complete recovery takes several minutes
- **Shadow servers** exist
 - Can handle reads of files
 - In place of the master
- Requires a **massive main memory**

9/22/2016 CS535 Big Data - Fall 2016 W5.A.33

The system is optimized for large files

- But **not for** a very large number of very **small files**
- Primary operation on files
 - Long, sequential reads/writes
 - Large number of **random overwrites** will **clog** things up quite a bit

9/22/2016 CS535 Big Data - Fall 2016 W5.A.34

Consistency Issues: GFS expects clients to resolve inconsistencies

- File chunks may have **gaps or duplicates** of some records
 - The client has to be able to deal with this
- Imagine doing this for a scientific application
 - Portions of a massive array are corrupted
 - Clients would have to detect this
 - Detection is possible of course, but **onerous!**
- **HDFS handles gaps and duplicates**

9/22/2016 CS535 Big Data - Fall 2016 W5.A.35

Security model

- **Originally None**
 - Operation is expected to be in a trusted environment

9/22/2016 CS535 Big Data - Fall 2016 W5.A.36

Distributed File Systems

Colossus: Google File System II

9/22/2016 CS535 Big Data - Fall 2016 W5.A.37

Storage Software: Colossus (GFS2)

- Next-generation cluster-level file system
- Automatically sharded metadata layer
 - Distributed Masters (64MB block size → 1MB)
 - Data typically written using Reed-Solomon (1.5x)
 - Client-driven replication, encoding and replication
 - Metadata space has enabled availability
- Why Reed-Solomon?
 - Cost
 - Especially with cross cluster replication
 - More flexible cost vs. availability choices
- Google File System II: Dawn of the Multiplying Master Nodes, http://www.theregister.co.uk/2009/08/12/google_file_system_part_deux/?page=1

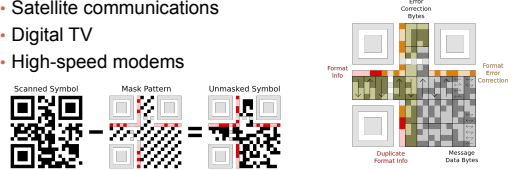
9/22/2016 CS535 Big Data - Fall 2016 W5.A.38

Reed-Solomon Codes

9/22/2016 CS535 Big Data - Fall 2016 W5.A.39

Reed-Solomon Codes

- Block-based error correcting codes
 - Digital communication and storage
- Storage devices (including tape, CD, DVD, barcodes, etc)
- Wireless or mobile communications
- Satellite communications
- Digital TV
- High-speed modems



SOURCE: https://en.wikiversity.org/wiki/Reed-Solomon_codes_for_coders