CS x55: Distributed Systems

[The Google File System]

So Long, and Thanks for All the Fish

Outside of CS, there are few who care
For how we conjure things in software
Of stripe sets and hash buckets
And how networks route packets

Or wonder why accesses to memory are blazingly quick
But those to disks, so eternally slow
Or ponder what makes MapReduce tick
Here's to the pursuit of being in the know!

Shrideep Pallickara
Computer Science
Colorado State University

Frequently asked questions from the previous class survey

- Does GFS have the same non-sequential reads property as BitTorrent?
- Are all 64-bit IDs associated with the chunks stored in one file?

Topics covered in this lecture

- The Google File System

Replication

Reasons why chunk replicas are created

- Chunk creation
- Re-replication
- Rebalancing

Chunk replica creation

- Place replicas on chunk servers with below average **disk space** utilization
- **Limit** the number of **recent creations** on a chunk server
  - Predictor of imminent heavy traffic
- Spread replicas **across racks**

Re-replicate chunks when replication level drops

- How **far** is it from replication goal?
- Preference for chunks of **live** files
- Boost priority of chunks **blocking client progress**

Rebalancing replicas

- **Examine** current replica distribution and **move** replicas
  - Better disk space
  - Load balancing
- **Removal** of existing replicas
  - Chunk servers with below-average disk space

Incorporating a new chunk server

- **Do not swamp** new server with lots of chunks
  - Concomitant traffic will bog down the machine
- **Gradually** fill up new server with chunks

So, so you think you can tell
Heaven from hell?
Blue skies from pain?
Can you tell a green field
From a cold steel rail?
A smile from a veil?
Do you think you can tell?

Wish You Were Here; Gilmour/Waters; Pink Floyd

CREATING SNAPSHOTS

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

- Master **revoke**s outstanding leases for any chunks of the file (source) to be snapshot
- **Log** the operation to disk
- Update in-memory state
  - Duplicate metadata of the source file
- Newly created file points to the same chunks as the source

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

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
- No inode needs to be protected from modification

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`
- Used to **prevent** operations during snapshots

Locks are used to prevent operations during snapshots

- For e.g. cannot create `/home/user/foo`
  - While `/home/user` is being snapshotted to `/save/user`
- Read locks on `/home` and `/save`
  - **Read lock prevents a directory from being deleted**
- Write lock on **`/home/user`** and `/save/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`

CONSISTENCY IN GFS

In GFS the state of file region after mutation depends on …

- **TYPE** of the mutation
- **SUCCESS/FAILURE** of the mutation
- Whether there were **CONCURRENT** mutations

GFS has a relaxed consistency model

- **Consistent**: See the same data
  - On all replicas
- **Defined**
  - Clients see mutation writes in its **entirety**

File state region after a mutation

	Write	Record Append
Serial success	**defined**	**defined** interspersed with *inconsistent*
Concurrent success	Consistent but *undefined*	
Failure	Inconsistent	

Implications for applications

- Rely on **appends** instead of overwrites
- Checkpoint
- Write **records** that are
  - Self-validating
  - Self-identifying

Deletion of Files & Garbage Collection

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

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 removed from namespace
  - *In memory metadata* is also removed
  - **Severs links** to all its chunks!

Garbage collection: When Master scans its chunk namespace

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

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 free to delete its replica of such chunks

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

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

Occurs BEFORE any client can write to chunk

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

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

Data Integrity

Data Integrity

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

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

Inefficiencies

The master server is a single point of failure

- Master server **restart** takes several seconds
- **Shadow servers** exist
  - Can handle reads of files
    - In place of the master
  - But not writes
- Requires a **massive main memory**

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

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!**

Security model

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

The contents of this slide-set are based on the following references

- Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung: The Google file system. Proceedings of SOSP 2003: 29-43.