Cloud Storage

Barry Denby

Griffith College Dublin

March 11, 2021

Cloud Storage

- Clouds provide mass storage as well as mass computation power
- Mobile devices of limited power and storage will opt to store some data on the cloud
- However, because there are potentially millions of users and vast amounts of data there is a need for specialist storage systems in the cloud
- There must also be effective access to replication and backups
  - The previous two objectives conflict
- A lot of this lecture is based on Cloud Computing: Theory and Practice

Cloud Storage: Why is data a problem?

- Data is vital to each and every application that runs on the cloud
- Applications are easy to replace whereas data is not
- To put this into context there are numbers on the amount of data human activity generates in a single year
  - Global data volumes at the end of 2009 exceeded 800EB (Exabytes, Giga $\rightarrow$ Tera $\rightarrow$ Peta $\rightarrow$ Exa)
  - At the end of 2013 internet video was expected to hit 18EB a month

Cloud Storage: Why is data a problem?

- Storage has not kept pace with CPU and memory developments
- While storage space has rapidly increased it has not kept pace with Moore's law
- This also applies to I/O transfer mechanisms
  - All cloud applications suffer as a result of this
  - Any storage mechanism must devise methods to use parallel I/O to overcome these limitations
- Storage systems face pressure because data generation has increased exponentially

Cloud Storage: How does this affect storage design?

- Due to the need to have multiple backups we must have space for at least two to three copies of all data
- Because of the size of generated data we need large amounts of storage therefore more disks
- More disks means a higher chance of disk failure
- Must diagnose faults, hide them, and rebuild when faults have been corrected

Cloud Storage: How does this affect storage design?

- Due to having multiple disks and the potential for data to be greater than the size of a single disk
- We need to use RAID arrays to create a large storage space on a single node
- We also need parallel file systems to divide files between multiple nodes
- We must also be able to use parallel read/write to overcome the limitations of I/O links

Storage Models

- A storage model describes the layout of a datastructure in a physical storage device.
- While a data model captures the logical aspects of the data that is to be stored.
- There are generally two models of data storage that are commonly used: cell storage and journal storage.

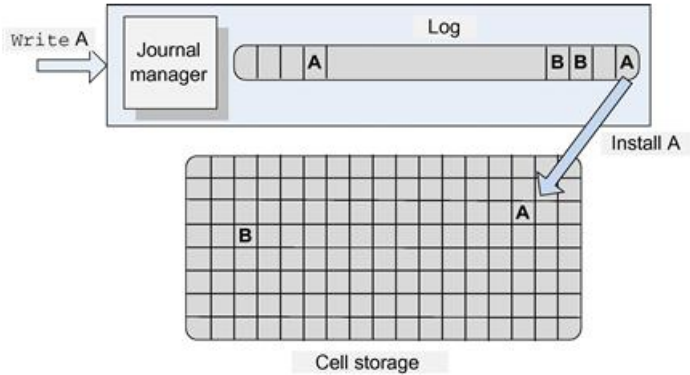
Cell Storage

- Cell storage assumes that your storage consists of blocks of the exact same size
  - And that all objects fit into once cell exactly
  - Bigger objects will be divided into multiple cells
- Cell storage assumes that your storage consists of blocks of the exact same size
- Two properties that are desirable in this kind of storage is coherence and atomicity
  - Coherence states that a read value should equal the value of the last write on a cell
  - Atomicity states that a read or write operation cannot be interrupted and will finish before the next operation begins

Journal Storage

- Journal storage is cell storage but with a manager object on top
  - All read and write operations must go through the manager first
  - Journal storage maintains an entire history for each each and every cell in storage
  - Users do not have direct access to any cell
  - Journal manager will log all write operations before they are committed to storage
  - If the storage fails or suffers a power cut or bad shutdown then the journal can be used to finish all write operations that were not committed to disk
- All or nothing operations
  - An all or nothing operation will only be committed if it is complete. if it is incomplete nothing gets committed

Journal Storage



Transaction Processing

- Transaction processing systems are based on cell or journal storage
- Transactions consist of multiple actions that should be completed in one go
  - There is a possibility that a transaction may fail before completion
  - Transaction systems must ensure that should a transaction fail it will be completed on recovery
  - Must guarantee correctness: the result should be the same regardless of the order the actions were received
  - Most transaction systems will use all or nothing operations to guarantee correctness

File Systems

- File systems are ordered collections of directories and files. However in the cloud due to the scale of data involved we need network centric filesystems
- However, there are a number of requirements that a cloud file system must satisfy
  - For large distributed file systems there must be a manager/coordinator to maintain the consistency of files and ensure they are written correctly
  - Cloud applications must not interface with the file system directly
  - Should enforce data integrity

File Systems

- However, there are a number of requirements that a cloud file system must satisfy
  - Should manage data access
  - Enable concurrency control
  - Support failure recovery
  - Low latency transfers
  - Must be scalable
  - Highly available
  - Replication must guarantee consistency of data

File Systems

- Traditional web applications use relational databases
- However, these do not scale well in cloud applications
- Fine grained locking of single database rows or cells means applications spend much time acquiring and releasing locks
- Thus the move towards simpler key-value pair databases with coarse grained locking (NoSQL Databases)

Parallel File System

- A parallel file system uses multiple I/O operations concurrently to improve data transfer rates
- Most cloud applications will use a parallel system in the background to serve data to requests as quickly as possible
- Parallel file systems allow multiple clients to read and write concurrently to the same file
- Usually build on a SAN to maximise throughput as Ethernet does not have the necessary bandwidth for this

Google File System

- Google's approach to the cloud storage problem
- GFS is designed to use inexpensive commodity components to build thousands of storage systems linked together
- Because of the number of nodes reliability and scalability are the main concerns

Google File System: Design criteria

- Scalability and Reliability are critical to the system
- Expected to take file sizes in the GB to TB range
- The most common write operation will be file appending
- Most common read operation is sequential read
- Data will be processed in bulk instead of small amounts

Google File System: Design Decisions

- Files are segmented in large chunks
- Appending should be an atomic operation
- Favour a high bandwidth network over a low latency one
- Caching is forbidden on the client as it complicates consistency

Google File System: Design Decisions

- Ensure consistency by channelling critical file operations through a master node
- Master node should have very little involvement in file access operations
  - avoids contention and ensures scalability
- Support efficient checkpointing and fast recovery
- Support efficient garbage collection mechanisms
  - necessary for freeing unused space on the file system

Google File System

- Files are divided into large chunks in the MB to GB range
- These chunks will be replicated and stored on multiple nodes in the system.
- Each file is made of one or more chunks

Google File System: Chunk Location

- Location of all chunks is stored at the master node
- This list is updated at system startup
- Or whenever a new chunk server joins the file system
- All requests for chunks will go through the master node

Google File System: Chunk Consistency

- The master node maintains a log of all metadata changes in case of system failure
- All changes in GFS are considered atomic
- Thus any change will not be visible to a client until the change is completed and sufficiently replicated

Google File System: Chunk Locations

- The master knows the locations of all chunks in the system
  - Thus all reads and writes must go through the masters
  - Simplifies the maintenance and updates of all chunks in the system
  - The list gets updated at system startup or when a new chunk server joins the system
- Elements of Journalling
  - An operations log is maintained by all nodes in the system
  - This allows the master to recover in case of a failure
  - Any changes to the file system are not made visible to clients until the master has made multiple persistent copies

Google File System: Checkpointing

- Master takes periodic snapshots of the system
- Snapshots are of system in a known good state
- Thus logs will only be replayed from the last successful snapshot
- Recovering from the last snapshot is much quicker than recovering from the beginning of the log

Google File System: Chunk Servers

- Each chunk server is a slave in GFS
- Anytime a client requests a file the master will communicate with the appropriate chunk server and will grant the client access to that chunk
- From then on the client and chunk server communicate directly
- Enables offloading of read and write operations from the master
- Reduces contention, and enhances scalability

Google File System: Chunk Servers

- File creation operations are normally handled by the master
- Reads, Writes, and Appends handled by chunk servers as the occur frequently
- Master will allocated a primary chunk server for each operation (delegated master)
- Along with a set of secondary chunk servers (delegated slaves) that have copies of chunks or files
- Client interacts with the primary chunk server
- Primary enforces changes while secondaries replicate

Google File System: File and Chunk Deletion

- Periodically chunk servers are required to check in with the master node
- Required to update master with list of chunks it contains
- If chunks or files have been deleted from the system the master notifies chunk server of these orphaned chunks
- Chunk server deleted these chunks and removes them from the system

Google File System: Chunk Integrity

- Each chunk server maintains checksums for locally stored chunks
- Needed to ensure the integrity of data
- Can be compared with other checksums from replicas to ensure consistency
- Should there be a difference in checksums the majority checksum is considered correct

Locking Mechanisms

- Locks are necessary in a distributed file system to ensure consistency of files
- Also used to enforce atomicity of operations
- There are many forms a lock can take
- Advisory Locks: Assumes all process obey the rules (no Byzantine processes) and will not circumvent the locking system, thus no protection
- Mandatory Locks: If a process does not hold the lock then it will not be granted access, stricter but ensures consistency

Locking Mechanisms

- Fine Grained Locks: locks guarding small size data
  - Generally used on small size data that needs quick updates
  - If a process requires many fine grained locks a significant amount of time is spent acquiring and releasing locks
- Coarse Grained Locks: locks guarding large size data
  - While less time is taken in acquiring and releasing locks
  - It is possible other processes may be waiting a long time should one process hold a lock for long

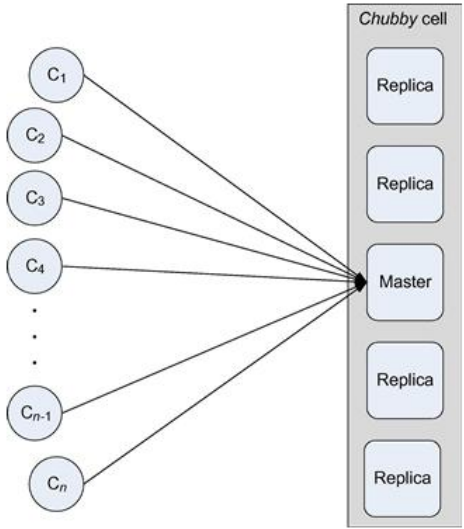
Locking Mechanisms

- The choice of locking system is important in a distributed file system
  - Fine grained locks will permit more threads and processes to access shared data however, this imposes more lock/unlock operations on the lock server
  - If the lock server fails they there is a larger number of applications that will be affected by the loss of the server
  - Mandatory locks may enforce consistency but they will create additional work and checks on the lock server
  - This would suggest that coarse grained, advisory locks are the choice for GFS as it must scale to thousands of nodes and processes

Google File System: Chubby Lock Server

- Lock server that manages locks in GFS
- As this is a critical component there are at least 5 servers in a group
- Should a chubby server fail, a pause will show in the system while a new chubby server is elected as master
- Upon completion all processes resume their work
- Uses coarse grained advisory locks
- Layout shown on next slide

Google File System: Chubby Lock Server



Google File System: Chubby Lock Server

- Generally there will be 5 servers implementing the chubby service
- One master with 4 replicas. A group of 5 servers here is called a chubby cell
- The idea is that should one of the lock servers fail there are at least four other copies of lock state information
  - thus should a single server go down the lock information is still present thus avoiding deadlock
- All lock requests are sent to the master node
- If the master grants or rescinds a lock from a client this information is replicated before the operation is completed

Google File System: Chubby Lock Server

- The 5 servers are placed a large physical distance apart to reduce the probability of one server failure causing another to fail
- When the master fails a consensus algorithm is run to determine the new master (election by simple majority)
- The new master also gets a master lease
- This prevents reelections for a given time period or until the master node fails

Google File System: Chubby Lock Server

- Each client maintains a session with the lock server
- Chubby demands that the session is maintained at all times. If not the locks associated with that session are automatically rescinded
- Clients will use RPCs to communicate with the chubby server
- Read commands will issue immediately as they do not modify data

Google File System: Chubby Lock Server

- Write commands will stall until master chubby server receives a majority response from slave chubby servers
- To ensure that the lock state information has been recorded in case of master node failure

Google File System: Chubby Notification Events

- Clients can register for certain notification events
  - File Content Modification: As soon as a file is modified an application may request an immediate read to update itself based on new file content
  - Lock Acquirement: Clients need to know they have acquired their locks before writing a file
  - Master Failure/Master Election: If the master fails the clients should stop issuing requests until a new master has been elected

Google File System: Chubby Locks

- Uses coarse grained locks on the file and directory level
- Only one client is permitted to hold a lock at any time
- However, any reading process is permitted to read the content of the file regardless of lock state

Google File System: Chubby Implementation Details

- Each replica maintains an entire copy of the all or nothing atomicity log
  - In the event of a failure all-or-nothing actions that are partially committed to the log will be undone
  - In the event of a failure all-or-nothing actions that have been fully committed to the log but not to disk will be completed before client access is granted

Google File System: Chubby Implementation Details

- Chubby has two separate network connections
  - One for clients, and one for replicas
  - The replica network enables fast communications between all replicas
  - Helps ensure consistency among all replicas due to low latency links
  - Also aids reliability as if one network goes down the other is still available for communications
- Diagram of chubby on last slide

Google File System: Chubby Diagram