Goel, Ajay: Nu Id (001897443)

INFO 7250
Engineering Big-Data Systems
Summer Full 2019
Assignment 1

The Google File System:
It was designed to provide fault tolerance while running on inexpensive commodity hardware, and it delivers high aggregate performance to a large number of clients to meet performance, scalability, reliability, and availability.

Features re-examined for this system:

- The file system consists of hundreds or even thousands of storage machines built from inexpensive com, modify parts and is accessed by a comparable number of client machines. Problems like application bugs, operating system bugs, human errors, and the failures of disks, memory, connectors, networking, and power supplies occurs a lot.
- File size is very huge, multi GB files are common.
- Most files are mutated by appending new data rather than overwriting existing data which becomes the focus of performance optimization and atomicity guarantees while caching data blocks in the client loses its appeal.
- Co-designing the applications and the file system API benefits the overall system by increasing our flexibility.

Assumptions in the Google File System:

- The system is built from many inexpensive commodity components that often fail. It must constantly monitor itself and detect, tolerate, and recover promptly from component failures on a routine basis.
- The system stores a modest number of large files.
- The workloads primarily consist of two kinds of reads: large streaming reads and small random reads.
- The workloads also have many large, sequential writes that append data to files. Once written, files are seldom modified again.
- The system must efficiently implement well-defined semantics for multiple clients that concurrently append to the same file.

In GFS, Files are organized hierarchically in directories and identified by path names. GFS has snapshot and record append operations. Snapshot creates a copy of a file or a directory tree at low cost. It is useful for implementing multi-way merge results.

Architecture:

- A GFS cluster consists of a single master and multiple chunk servers and is accessed by multiple clients. Chunk Servers are immutable.
- Chunk servers store chunks on local disks as Linux files and read or write chunk ta specified by a chunk handle and byte range.
- The master maintains all file system metadata.
- It also controls system-wide activities such as chunk lease management, garbage collection of orphaned chunks, and chunk migration between chunk servers.
- Clients interact with the master for metadata operations, but all data-bearing communication goes directly to the chunk servers.
- Chunk servers need not cache file data because chunks are stored as local files and so Linux's buffer cache already keeps frequently accessed data in memory.

Why single master?

- It enables the master to make sophisticated chunk placement and replication decisions using global knowledge. However, we must minimize its involvement in reads and writes so that it does not become a bottleneck.

Chunk Size:

- Chosen size: 64 MB, it reduces clients' need to interact with the master because reads and writes on the same chunk require only one initial request to the master for chunk location information.
- client is more likely to perform many operations on a given chunk, it can reduce network overhead by keeping a persistent TCP connection to the chunk server over an extended period of time.

Meta Data:

- The master stores three major types of metadata:
    - the file and chunk name spaces, the mapping from files to chunks and the locations of each chunk's replicas.
      (logging mutations to an operation log stored on the master's local disk and replicated on remote machines to update the master state simply, reliably and without risking inconsistencies in the event of a master crash).

Periodic scanning is used to implement chunk garbage collection, re-replication in the presence of chunk server failures,
and chunk migration to balance load and disk space usage across chunk servers.

Chunk Locations:

- The master does not keep a persistent record of which chunk servers have a replica of a given chunk. At startup, master gather data of all the location from chunk servers and periodically thereafter.

Operation Logs:

- It contains a historical record of critical metadata changes.
- To avoid failure we replicate it on multiple remote machines and respond to a client operation only after flushing the corresponding log record to disk both locally and remotely.
- The master recovers its file system state by replaying the operation log. To minimize startup time, we must keep the log small. The master checkpoints its state whenever the log grows beyond a certain size so that it can recover by loading the latest checkpoint from local disk.

After a sequence of successful mutations, the mutated file region is guaranteed to be defined and contain the data written by the last mutation. GFS achieves this by

- applying mutations to a chunk sin the same order on all its replicas
- using chunk version numbers to detect any replica that has become stale because it has missed mutations while its chunk server was down.

Implications for Applications:

- By relying on appends rather than overwrites, checkpointing, and writing self-validating, self-identifying records.
- Appending is far more efficient and more resilient to application failures than random writes. Checkpointing allows writers to restart incrementally and keeps readers from processing successfully written file data that is still incomplete from the application's perspective.

SYSTEM INTERACTION:

The system is designed to minimize the master's involvement in all operations.

Leases and Mutation Order:

- Leases to maintain a consistent mutation order across replicas.
- The master grants a chunk lease to one of the replicas, which we call the primary. The primary picks a serial order for all mutations to the chunk. All replicas follow this order when applying mutations.
- Even if the master loses communication with a primary, it can safely grant a new lease to another replica after the old lease expires.
- Process:
    - The client asks the master which chunk server holds the current lease for the chunk and the locations of the other replicas. If no one has a lease, the master grants one to a replica it chooses (not shown).
    - The master replies with the identity of the primary and the locations of the other (secondary) replicas. The
      client caches this data for future mutations. It needs to contact the master again only when the primary becomes unreachable or replies that it no longer holds a lease.
    - The client pushes the data to all the replicas.
    - Once all the replicas have acknowledged receiving the data, the client sends a write request to the primary. The request identifies the data pushed earlier to all of the replicas. The primary assigns consecutive serial numbers to all the mutations it receives, possibly from multiple clients, which provides the necessary serialization.
    - The primary forwards the write request to all secondary replicas.

  - The secondaries all reply to the primary indicating that they have completed the operation.
  - The primary replies to the client. Any errors encountered at any of the replicas are reported to the client.

Data Flow:

- Control flows from the client to the primary and then to all secondaries, data is pushed linearly along a carefully picked chain of chunk servers in a pipelined fashion. Goals were to fully utilize each machine's network bandwidth, avoid network bottlenecks and high-latency links, and minimize the latency to push through all the data.
- To fully utilize each machine's network bandwidth, the data is pushed linearly along a chain of chunk servers. To avoid network bottlenecks and high-latency links each machine forwards the data to the "closest" machine in the network topology that has not received it.

Atomic Record Appends:

- In a record append, however, the client specifies only the data. GFS appends it to the file at least once at an offset of GFS's choosing and returns that offset to the client.
- The client pushes the data to all replicas of the last chunk of the file Then, it sends its request to the primary. The primary checks to see if appending the record to the current chunk would cause the chunk to exceed the maximum size (64 MB). If so, it pads the chunk to the maximum size, tells secondaries to do the same, and replies to the client indicating that the operation should be retried on the next chunk.
- It only guarantees that the data is written at least once as an atomic unit.

Snapshot:

- The snapshot operation makes a copy of a file. Users use it to quickly create branch copies of huge data sets.
- When the master receives a snapshot request, it first revokes any outstanding leases on the chunks in the files it is about to snapshot. This ensures that any subsequent writes to these chunks will require an interaction with the master to find the lease holder.
- After the leases have been revoked or have expired, the master logs the operation to disk. It then applies this log record to its in-memory state by duplicating the metadata for the source file or directory tree.

MASTER OPERATION:

- It executes all namespaces operations.
- It manages chunk replicas throughout the system, makes placement decisions, creates new chunks and hence replicas, and coordinates various system-wide activities to keep chunks fully replicated, to balance load across all the chunk servers and to reclaim unused storage.

Namespace Management and locking:

Many master operations can take a long time: a snapshot operation has to revoke chunk server leases on all chunks covered by the snapshot. We do not want to delay other master operations while they are running. Therefore, we allow multiple operations to be active and use locks over regions of the namespace to ensure proper serialization.
Advantage:

- It allows concurrent mutations in the same directory. For example, multiple file creations can be executed concurrently in the same directory: each acquires a read lock on the directory name and a write lock on the file name. The read lock on the directory name suffices to prevent the directory from being deleted, renamed, or snapshotted. The write locks on file names serialize attempts to create a file with the same name twice.

Replica Placement:

- The chunk replica placement policy serves two purposes:
  - maximize data reliability and availability and maximize network bandwidth utilization.
  - Chunk replicas across ensures that some replicas of a chunk will survive and remain available even if an entire rack is damaged or offline. It also means that traffic, especially reads, for a chunk can exploit the aggregate bandwidth of multiple racks.

Creation, Re-replication, Rebalancing:

When the master creates a chunk, it chooses where to place the initially empty replicas.
The master re-replicates a chunk as soon as the number of available replicas falls below a user-specified goal.
The master rebalances replicas periodically: it examines the current replica distribution and moves replica's for better disks pace and load balancing.

Garbage Collection: After a file is deleted, GFS does not immediately reclaim the available physical storage. It does so only lazily during regular garbage collection at both the file and chunk levels.

Stale Replica Detection:

- Chunk replicas may become stale if a chunk server fails and misses mutations to the chunk while it is down.
- Whenever the master grants a new lease on a chunk, it increases the chunk version number and informs the up-to date
  replicas. The master and these replicas all record the new version number in their persistent state.
- The master will detect that this chunk server has a stale replica when the chunk server restarts and reports its set of chunks and their associated version numbers.

FAULT TOLERANCE AND DIAGNOSIS:

- High Availability
  - Fast Recovery
  - Chunk Replication
  - Master Replication
- Data Integrity
  - Diagnostic Tools
- Measurements
  - Micro benchmarks
  - Real World Clusters
  - Work Load Breakdown

Conclusion:

- The Google File System demonstrates the qualities essential for supporting large-scale data processing workloads on commodity hardware.
- Observations have led to radically different points in the design space.
  - Treat component failures as the norm rather than the exception, optimize for huge files that are mostly appended and both extend and relax the standard file system interface to improve the overall system.
  - Our system provides fault tolerance by constant monitoring, replicating crucial data, and fast and automatic recovery.
- Chunk replication allows us to tolerate chunk server failures. Design delivers high aggregate throughput to many
  concurrent readers and writers performing a variety of tasks.
- Improvements in our networking stack will lift the current limitation on the write throughput seen by an individual client. GFS has successfully met storage needs and is widely used within Google as the storage platform for research and development as well as production data processing. It is an important tool that enables to continue to innovate and attack problems on the scale of the entire web.