Google File System

Architecture

Availability

Fault tolerance

Optimization for large scale data

Google File System (GFS)

- Google file system is a scalable distributed file system developed by Google to provide efficient and reliable access to data using large clusters of commodity hardware.
- It is designed to meet the rapidly growing demand of Google's data processing need.
- It provides performance, scalability, reliability and availability of data across distributed system for handling and processing big data.

Characteristics of GFS

- The Google File System (GFS) is a distributed file system designed for large-scale data storage and retrieval. It's characterized by several key features that contribute to its scalability, reliability, and performance:
- Scalability:
- Horizontal scaling: GFS can easily scale to handle massive amounts of data by adding more servers to the cluster. This allows it to grow alongside increasing data needs.
- Chunk-based data: Dividing files into fixed-size chunks (64MB) enables parallel processing and efficient data access across multiple servers.

Characteristics of GFS

- Reliability:
  - Master-slave architecture: A master node manages metadata and coordinates operations, while slave nodes store and manage the actual data. This ensures high availability even if the master node fails.
  - Data replication: Each chunk is replicated multiple times across different servers. This redundancy ensures data accessibility even if some servers fail.
  - Automatic failover and recovery: GFS automatically detects and recovers from server failures, minimizing downtime and data loss.

Characteristics of GFS

- Performance:
  - High aggregate throughput: GFS can handle concurrent read and write requests from a large number of clients. This is achieved through parallel processing of chunks across multiple servers.
  - Large chunk size: The large 64MB chunk size reduces overhead and improves read/write performance compared to smaller block sizes.
  - Simple design: GFS's architecture is designed to be simple and efficient, avoiding unnecessary complexity that can impact performance.

Characteristics of GFS

- Cost-effectiveness: GFS utilizes commodity hardware, which makes it a cost-effective solution for storing large amounts of data compared to specialized hardware solutions.
- Fault tolerance: GFS's distributed nature and data replication make it highly resistant to failures and disruptions.
- Ease of use: GFS provides a simple API and management tools for users to easily interact with the system.
- Write-once, read-many: Once written, data in a chunk cannot be modified directly. Instead, new chunks are created for updates, while the original chunk remains unchanged.

Characteristics of GFS

- Append-optimized: GFS is optimized for append operations, making it suitable for storing large, continuously growing datasets.
- Relaxed consistency model: GFS uses a relaxed consistency model, which prioritizes availability and performance over strict consistency guarantees. This means that different clients may see slightly different versions of the same data at any given time.
- The combination of these characteristics makes GFS a powerful and versatile distributed file system for large-scale data storage and processing. It has been instrumental in supporting Google's data-intensive applications and continues to be a key component of its infrastructure.

Common goals of GFS

- The development of the Google File System (GFS) was driven by the need to address specific challenges and requirements posed by Google's distributed computing environment.
- The fundamental principles that influenced the development of GFS are as follow:
  - Performance
  - Reliability
  - Automation
  - Fault Tolerance
  - Scalability
  - Availability

Common goals of GFS

- **Performance:**
  - Google processes and analyzes vast amounts of data, requiring a file system optimized for high performance. GFS is designed to provide efficient data access, especially for applications that involve large datasets and sequential processing.
- **Reliability:**
  - In a distributed environment, hardware failures are expected to occur. GFS prioritizes reliability by implementing features such as data replication, ensuring that even in the event of node failures, data remains accessible from redundant copies.
- **Automation:**
  - The scale and complexity of Google's infrastructure necessitate automation to manage various aspects of the file system efficiently. Automation in GFS includes tasks like data placement, load balancing, and fault detection, reducing the need for manual intervention.

Common goals of GFS

- **Fault Tolerance:**
  - Given the scale of Google's operations, hardware failures are considered inevitable. GFS is designed with fault-tolerant mechanisms to handle these failures, ensuring the system remains operational and data remains accessible.
- **Scalability:**
  - Google's data processing needs are continually growing. GFS is engineered to scale horizontally, allowing for the addition of more nodes to the system. This scalability ensures that the file system can handle increasing amounts of data and user demands.
- **Availability:**
  - High availability is critical for Google's services that rely on constant access to data. GFS is designed to be available at all times, minimizing downtime and ensuring that applications and services can access data without interruption.
- By addressing these common goals, the development of GFS has provided Google with a distributed file system that meets the unique challenges of their infrastructure. GFS has become a foundational component of Google's data storage and processing capabilities, supporting the company's diverse set of applications and services.

Terminology

- chunk—fixed-size piece of file
- chunk server—holds chunks
- master—coordinates chunk servers
- chunk handle—ID of a chunk (64 bit, globally unique)

Chunk

- In the context of the Google File System (GFS), a chunk refers to a fixed-size unit of data that files are divided into. Each chunk has the following characteristics:
- Size: 64 MB. This size is chosen to be larger than typical file system block sizes for better performance and reduced overhead.
- Identification: Each chunk is assigned a unique 64-bit identifier called a chunk handle. This handle allows the GFS master node to track and manage individual chunks regardless of their location.
- Location: Chunks are stored on chunkservers, which are commodity hardware machines dedicated to storing GFS data. The master node maintains a map of chunk handles to their locations, ensuring that data can be retrieved efficiently.
- Replication: Each chunk is replicated multiple times (typically 3) across different chunkservers to ensure data availability even if one or more machines fail.
- Immutability: Once created, a chunk's data cannot be directly modified. Instead, new chunks are created for updates, while the original chunk remains unchanged.

Chunk

- The use of chunks in GFS provides several benefits:
- Scalability: Chunks allow GFS to efficiently store and manage large amounts of data across many machines.
- Reliability: Replication of chunks ensures data availability even in the event of hardware failures.
- Performance: Larger chunk sizes improve read and write performance compared to smaller block sizes.
- Parallelism: Chunks can be accessed and processed concurrently, making GFS suitable for large-scale data processing tasks.
- Chunkservers only store chunks and perform read/write operations as instructed by the master node.
- The master node maintains all metadata about chunks, including their locations and replicas.
- GFS clients interact with chunks through the master node, which maps file offsets to chunk handles and locations.
- GFS can efficiently handle large files that span many chunks.

Chunk Server

- In the context of the Google File System (GFS), a chunk server is a dedicated server responsible for storing and managing data chunks. It plays a vital role in GFS's distributed architecture, ensuring efficient and reliable data storage and retrieval.
- Here are the key responsibilities of a chunk server:
- Data storage: Chunk servers store data chunks, which are fixed-size units (64MB by default) that GFS uses to divide large files.
- Data retrieval: Chunk servers respond to client requests for data by reading and sending the requested chunk.
- Replication management: Chunk servers replicate their data to other chunk servers as instructed by the master node. This ensures that data remains available even if one or more chunk servers fail.
- Failure recovery: Chunk servers can automatically recover from failures by retrieving missing data from replicas.
- Heartbeat communication: Chunk servers regularly send heartbeat messages to the master node to indicate their health and availability.

Chunk Server

- Here are some key characteristics of chunk servers:
- Commodity hardware: GFS utilizes commodity hardware for chunk servers, making the system cost-effective and scalable.
- Simple design: Chunk servers have a simple design with a limited set of functions. This design makes them reliable and easier to maintain.
- High availability: GFS's distributed architecture and data replication mechanisms ensure that chunk servers are highly available.
- Scalability: GFS can easily scale by adding more chunk servers to the cluster. This allows for efficient handling of increasing data needs.

Chunk Server

- Here are the typical components of a chunk server:
- Storage devices: Chunk servers utilize hard drives or other storage devices to store data chunks.
- Network interface: The network interface allows chunk servers to communicate with the master node, clients, and other chunk servers.
- Software: Chunk servers run a lightweight software stack that includes code for data storage, retrieval, replication, and heartbeat communication.
- Overall, chunk servers are fundamental components of GFS's architecture. Their efficient data storage and retrieval capabilities, combined with their simplicity, high availability, and scalability, make them crucial for GFS's success in managing large-scale data sets.

Chunk Handle

- In the Google File System (GFS), a chunk handle is a 64-bit integer that uniquely identifies a specific chunk of data. These handles play a crucial role in GFS's operation and contribute significantly to its efficiency, reliability, and scalability.
- Key functions of chunk handles:
- Data identification: Each chunk handle uniquely identifies a specific chunk of data within the GFS cluster, regardless of its physical location. This allows the master node and clients to efficiently locate and access the desired data.
- Data access: Clients use chunk handles to request specific chunks of data from the master node, which then directs them to the relevant chunk servers. This eliminates the need for clients to navigate the entire file system hierarchy.

Chunk Handle

- Key functions of chunk handles:
- Data replication: GFS replicates data across multiple chunk servers for redundancy and fault tolerance. Chunk handles ensure that all replicas of a chunk share the same identification, enabling clients to access the data even if one or more replicas become unavailable.
- Failure recovery: In case of a chunk server failure, the master node can quickly identify and locate available replicas using their chunk handles. This allows GFS to automatically recover from failures and ensure data availability.
- Simplified operations: Chunk handles provide a convenient and transparent abstraction for managing data in GFS. Clients do not need to be aware of the underlying distributed architecture or the details of data replication, simplifying their interaction with the system.

Chunk Handle

- Characteristics of chunk handles:
- Immutability: Once assigned, a chunk handle remains unchanged throughout the life of the chunk. This ensures consistency and simplifies data management.
- Globally unique: Each chunk handle is unique across the entire GFS cluster, preventing any ambiguity or conflicts.
- Efficiency: The 64-bit size of chunk handles provides a balance between efficient storage and the ability to address a large number of chunks.

Chunk Handle

- Benefits of using chunk handles:
- Improved scalability: GFS can scale horizontally by adding more chunk servers, and chunk handles facilitate efficient data management across a growing cluster.
- Increased reliability: Data replication and the use of chunk handles ensure data availability even if hardware failures occur.
- Reduced overhead: Chunk handles simplify data identification and access, minimizing unnecessary processing and communication overhead.
- Enhanced security: Chunk handles can be used to implement access control and data security mechanisms.
- In conclusion, chunk handles are fundamental components of GFS that significantly impact its performance, reliability, and scalability. By providing a unique and efficient way to identify and manage data chunks, they contribute to GFS's effectiveness as a distributed file system for large-scale data storage.

Design Overview: Assumptions

- The design of the Google File System (GFS) is based on several key assumptions that reflect the challenges and requirements of Google's distributed computing environment. Here are some of the assumptions that influenced the development of GFS:
  - Component Failure
  - Huge Datasets (Big Data)
  - Scalability
  - Fault Tolerance
  - Simplicity
  - Mutation

Design Overview: Assumptions

- **Component Failure :** Google developers have stated that "component failures are the norm rather than the exception". Their file system should consist of thousands of inexpensive commodity hardware, where auto recovery is essential.
- **Huge Datasets (Big Data) :**Google developers constantly deals with huge volume of datasets generated every minute. As the authors have stated, - "multi-GB files are common". it's quite difficult and tedious to manipulate huge datasets using traditional file system. Thus, there is a call for a solution to handle huge volume of datasets.
- **Scalability:** Scalability refers to easily adding computing capacity. Google developers must make sure system growth would not affect the overall performance of the system.

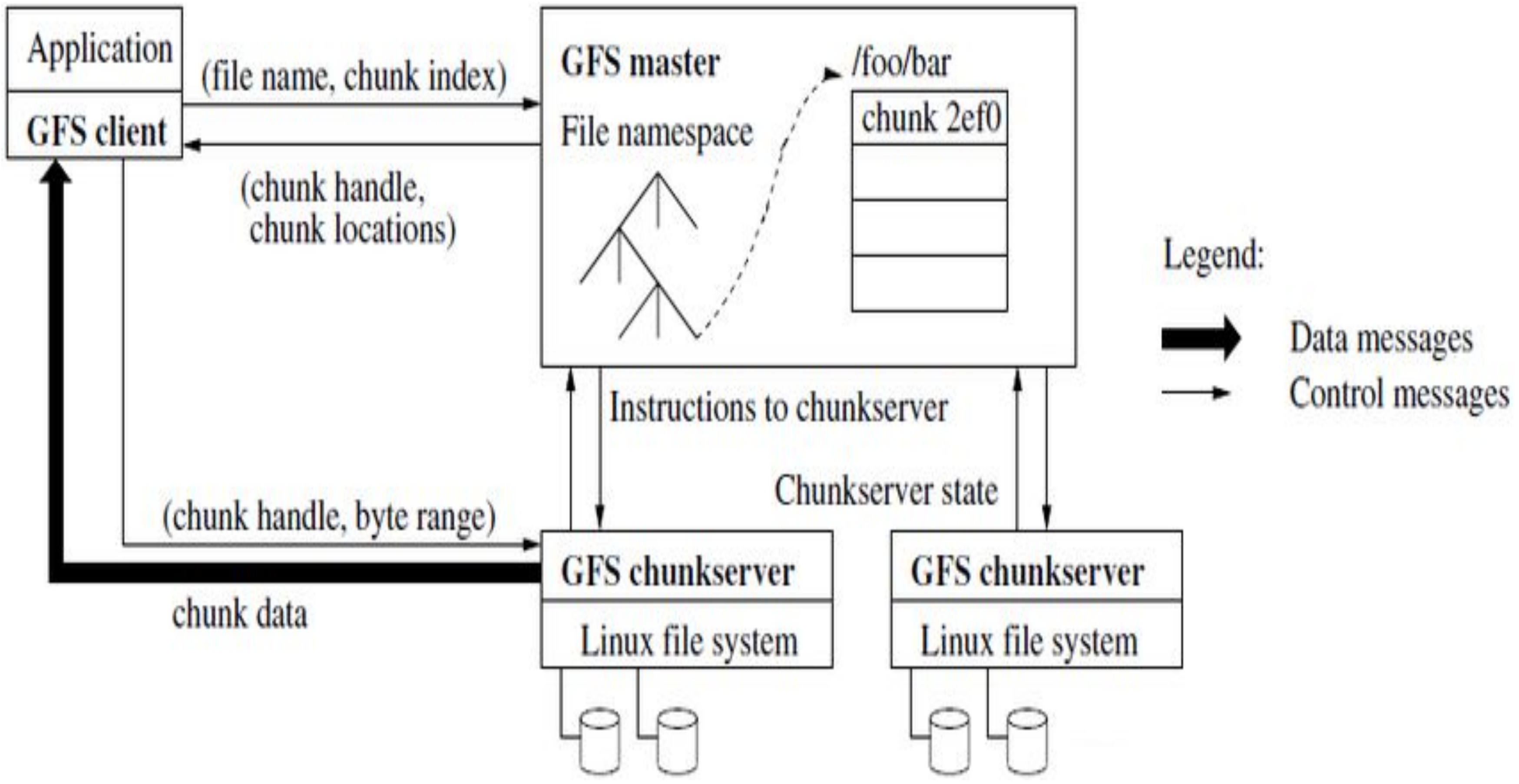
Design Overview: Assumptions

- **Fault Tolerance :**Google developers realized that as the network is so huge, monitoring and diagnosing will be a challenge task. They have assumed that their system should automatically monitor and diagnose itself from failure without human intervention.
- **Simplicity :**Google programmers must make sure that their design is simple as much as possible. They assumed that a simpler approach is easy to maintain and follow, even when the system is huge in scale.
- **Mutation :**Most of the files are mutated by appending to existing data rather than overwriting, unlike traditional file system.

Design Overview: Interface

- GFS provides a familiar file system interface, although it does not implement a standard API like POSIX(Portable Operating System Interface).
- Usual operations to create, delete, open, close, read and write files are supported.
- Additionally, GFS supports **snapshot** and **record append** operations.
- Snapshot creates a copy of file or directory tree.
- Record append allows multiple clients to append data to the same file concurrently while guaranteeing atomicity.

Google File System Architecture



Google File System Architecture

- 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.
- Sends request to a close replica, specifying chunk handle and byte range.
- Requests to master are typically buffered

- GFS is a cluster of computers. A cluster is simply a network of computers. Each cluster might contain hundreds or even thousands of computers. In each GFS cluster, there are 3 main entities.
  - Clients
  - Master Servers (only one)
  - Chunk Servers
- Single master, multiple chunkservers, multiple clients.

- The GFS system mainly stores 2 types of data
  - File metadata
  - File data
- **The master maintains all file system metadata which includes**
  - Namespace — A lookup table mapping full pathnames to metadata
  - Filename — Array of chunk handles (where to find the data)
  - Chunk handle — list of all chunk servers available with the master
  - List of chunk servers
  - Primary — Mapping the server as primary during the lease time.
  - Version number
  - Lease expiration time
  - Log & checkpoints on disk

- **A file**
  - Represented as fixed-sized chunks
  - Labeled with 64-bit unique global IDs
  - Divided in chunks , stored at Chunkservers
  - Files divided into fixed-size chunks.
    - Each chunk identified by immutable and globally unique chunk handle.
    - Stored by chunkservers locally as regular files.
    - Each chunk is replicated.
- **A GFS cluster**
  - A single master
  - Multiple chunkservers per master
    - Accessed by multiple clients
  - Running on commodity Linux machines

- **The client does not cache**
  - Streaming large files that cannot be cached
  - The working set is too large to be cached
- **The chunk server does not cache**
  - Data blocks are stored in local files

The Master Server

- It is responsible for the activities of the system such as managing chunk leases, load balancing and so on.
- It maintains all the file system metadata.
- It contains operation log that stores namespaces and file to chunk mappings.
- It periodically communicates with chunk server to determine chunk locations and assesses state of the overall system.
- Each node on the namespace tree has its own read-write lock to manage concurrency.

Single Master

- General disadvantages for distributed systems:
  - Single point of failure
  - Bottleneck (scalability)
- Solution?
  - Clients use Master only for metadata
  - Reading/writing goes directly through the chunkservers
  - Separating the file system metadata (stored solely by the master) and file data (stored across chunk servers) ensures that the single master does not result in a bottleneck.

Reasons for Single Master in GFS

- In Google File System (GFS), a single master node is responsible for managing the file system metadata, such as the location of each chunk of data and the mapping of file names to chunks. The master node also plays a key role in ensuring the consistency and availability of the file system by using a lease-based protocol to coordinate access to data.
- There are several reasons why GFS uses a single master node:
  - **Simplicity**: Using a single master node simplifies the overall design of the file system and reduces the complexity of managing and coordinating access to data.

Reasons for Single Master in GFS

  - **Scalability**: The master node can handle a large number of requests from clients without becoming a bottleneck, allowing GFS to scale to support a large number of clients.
  - **Fault tolerance:** GFS is designed to be highly available and can recover quickly from failures. The single master node can be replicated to ensure availability, and the use of chunk servers allows the file system to continue functioning even if some servers fail.
  - **Performance**: The single master node can process metadata requests quickly, allowing clients to access data efficiently.
- Overall, the use of a single master node in GFS helps to ensure the reliability, availability, and performance of the file system.

Reasons for Single Master in GFS

- Single master design simplifies the GFS design.
- It enables the master to make sophisticated chunk placement and replication decisions.
- Since, the master interacts with client for sharing metadata only, there is no necessity to have multiple master.
- All the heavy processes of read and write is handled by the chunk servers. So, lot of chunk servers are necessary for performance of the system.

Manage overloading of the Master

- The read and write operation is completely separated from master. Instead, these operations are performed by the chunk servers.
- For reliability, master state is replicated on multiple machines, using the operation logs and checkpoints.
- If master fails, GFS starts a new master process at any of these replica.
- Such replica of master state is known as shadow master.
- Shadow master is also able to perform read only access to the file system even when the primary master is down, but it lags the primary master by few fractions of second.

Shadow Master

- A shadow master is a replica of the primary master node in GFS. It maintains a consistent copy of the metadata and operation log, allowing it to take over if the primary master fails. This ensures that the file system remains operational with minimal downtime.
- Functionality of the Shadow Master:
- Read-only access: The shadow master can only perform read operations on the file system. It cannot modify any data or metadata directly.
- Continuous synchronization: The shadow master continuously synchronizes its state with the primary master to ensure consistency. This involves replicating the operation log and applying the same sequence of changes as the primary master.
- Fast failover: If the primary master fails, the shadow master can immediately take over its role. This eliminates the need for a lengthy election process or startup procedure, minimizing downtime and data loss.

Shadow Master

- Benefits of the Shadow Master:
- Improved availability: The shadow master provides a backup for the primary master, ensuring that the file system remains accessible even if the primary node fails.
- Increased read performance: Clients can read data from the shadow master, which can improve overall read performance by reducing load on the primary master.
- Reduced complexity: The shadow master design is simple and efficient, making it easier to manage and maintain.

Shadow Master

- Limitations of the Shadow Master:
- Lag: The shadow master may be slightly behind the primary master due to the replication process. This means that clients may see slightly different versions of the same data at any given time.
- Overhead: Maintaining a shadow master adds additional overhead to the system, as it requires resources for storage, memory, and processing.

Shadow Master

- The shadow master can be configured to automatically take over if the primary master fails, or it can be manually activated by an administrator.
- GFS typically uses multiple shadow masters for further redundancy and improved availability.
- Overall, the shadow master plays a crucial role in ensuring the high availability and performance of the Google File System. By providing a reliable backup for the primary master, it helps to minimize downtime and data loss, making GFS a robust and scalable solution for large-scale data storage and management.

Master Operation

- The Google File System (GFS) master server plays a crucial role in coordinating and managing various operations within the file system. Here's an overview of the key operations performed by the GFS master:
  - Executes all namespace operations
  - Manages chunk replicas throughout the system
  - Makes placement decisions, create new chunks
  - Ensures chunks are fully replicated
  - Balances load across all chunkservers
  - Reclaim unused storage

Master Operation

- **Executes All Namespace Operations:**
  - The master is responsible for managing the file system namespace, which includes operations related to file and directory creation, deletion, and modification. It maintains the hierarchy of files and directories and handles metadata operations.
- **Manages Chunk Replicas Throughout the System:**
  - The master keeps track of the location and status of all chunk replicas in the system. It maintains metadata about which chunks are stored on which chunk servers. This information is essential for data availability and fault tolerance.
- **Makes Placement Decisions, Creates New Chunks:**
  - When a file is created or extended, the master makes decisions about where to place the new chunks. It selects appropriate chunk servers for storing replicas of the new chunks. The master is involved in chunk placement decisions to ensure load balancing and efficient data storage.
- **Ensures Chunks Are Fully Replicated:**
  - The master ensures that each chunk is replicated on multiple chunk servers to provide fault tolerance. It monitors the status of replicas and takes corrective actions in the event of chunk server failures or unavailability.
- **Balances Load Across All Chunkservers:**
  - The master is responsible for load balancing across all chunk servers in the system. It monitors the load on each chunk server and may redistribute chunks or adjust placements to balance the overall load, preventing hotspots and optimizing system performance.

Master Operation

- **Reclaims Unused Storage:**
  - The master is involved in the management of storage resources. It tracks the usage of chunks and may initiate actions to reclaim storage space when chunks are no longer needed or when certain replicas become unavailable due to chunk server failures.
- **Handles Chunk Server Registration and Heartbeats:**
  - Chunk servers regularly send heartbeats to the master to indicate their availability and status. The master monitors these heartbeats and manages the registration and removal of chunk servers from the system. This information is crucial for maintaining an up-to-date view of the chunk servers in the GFS cluster.
- **Maintains Operation Logs:**
  - The master maintains logs of file system operations. These logs are used for recovery in the event of a master failure. By replaying these logs, the master can restore its state to a consistent point and continue operation.
- Overall, the GFS master server plays a central role in coordinating metadata operations, managing data placement, ensuring fault tolerance, and optimizing the performance and resource utilization of the file system.

Master Operation: Namespace management and locking

- GFS allow multiple operations to be active and use lock over regions of the namespace to ensure proper serialization
- GFS logically represents its namespace as a lookup table mapping full pathnames to metadata
- Each node in the namespace tree has an associated read write lock
- Each master operation acquires a set of locks before it runs
- Typically if it involves /d1/d2/leaf it will acquire read locks on the directory names /d1, /d1/d2and d1/d2/leaf. Note that leaf may be a file or directory depending on the operation
- 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
- Locks are acquired in a consistent total order to prevent deadlock.

Master Operation: Replica Placement

- GFS typically has hundreds of chunkservers spread across many machine racks, these chunkservers in turn may be accessed from hundreds of clients from the same or different racks
- Communication between the servers on different racks may require additional bandwidth
- The chunk replica placement policy serves two purposes: maximize data reliability and availability, and maximize network bandwidth utilization
- For both it is not enough to spread replicas across machines, which only guards disk or machine failures
- We must also spread chunk replicas across racks, this ensures that some replicas of a chunk will survive and remain available even if an entire rack is damaged.

Master Operation : Creation, Re-replication, Rebalancing

- When the master creates a chunk, it chooses where to place the initially empty replicas. It considered several factors:
  - New replicas are placed on chunkservers with below average disk space utilization
  - Number of recent creations on each chunkservers should be balanced, although creation itself is cheap, it may cause heavy write traffic because chunks are created when demanded by writes
  - Chunk replicas are created to spread replicas across racks
- The master re-replicates a chunk as soon as the number of available replicas falls below a user specified goal. This could happen for various reasons:
  - A chunkserver becomes unavailable, it reports that its replica may be corrupted or disk is disabled or the replication goal is increased

- Each chunk that needs to be re-replicated is prioritized based on several factors:
  - One is how far it is from its replication goal. For example, we give higher priority to a chunk that has lost two replicas than to a chunk that has lost only one
  - We prefer to first re-replicate chunks for live files as opposed to chunk that belongs to recently deleted files
  - We boost the priority of any chunk that is blocking client progress
- The master rebalances replicas periodically: it examines the current replica distribution and moves replicas for better space and load balancing
- Master gradually fills up a new chunkserverrather than instantly swamps it with new chunks
- In addition the master also choose which existing replica to remove, it prefers to remove those on chunkserverswith below average free space

Master Operation : Garbage Collection

- When a file is deleted by an application, the master server immediately writes the deletion record in the form of a log. The master server does not immediately reclaim the resource but changes the filename to a hidden name containing the deletion timestamp
- When the master node scans the file namespace regularly, it will delete the files before the specified time (the default is 3 days)
- Restore: just name files as the file name displayed normally
- Only when the hidden file is deleted from the namespace, the related metadata stored in the master server will be deleted and result in true disassociation from its related chunk server
- So, it provides a consistent and reliable way to remove useless copies

Master Operation : Stale Replica

- In the context of GFS, a "stale replica" typically refers to a copy of a file or a chunk of data that has become outdated or inconsistent with the latest version due to various reasons.
- In GFS, data is divided into chunks, and these chunks are replicated across multiple servers for fault tolerance and high availability. Staleness in GFS can occur for several reasons:
- **Replica Lag:** If a server hosting a replica falls behind in syncing updates from the master replica, it becomes stale. This can happen due to network issues, server overload, or other factors that delay the replication process.
- **Outdated Metadata:** Staleness can occur if the metadata associated with a file, such as file version or chunk locations, becomes outdated. This might happen if there are delays in propagating metadata updates across the system.

Master Operation : Stale Replica

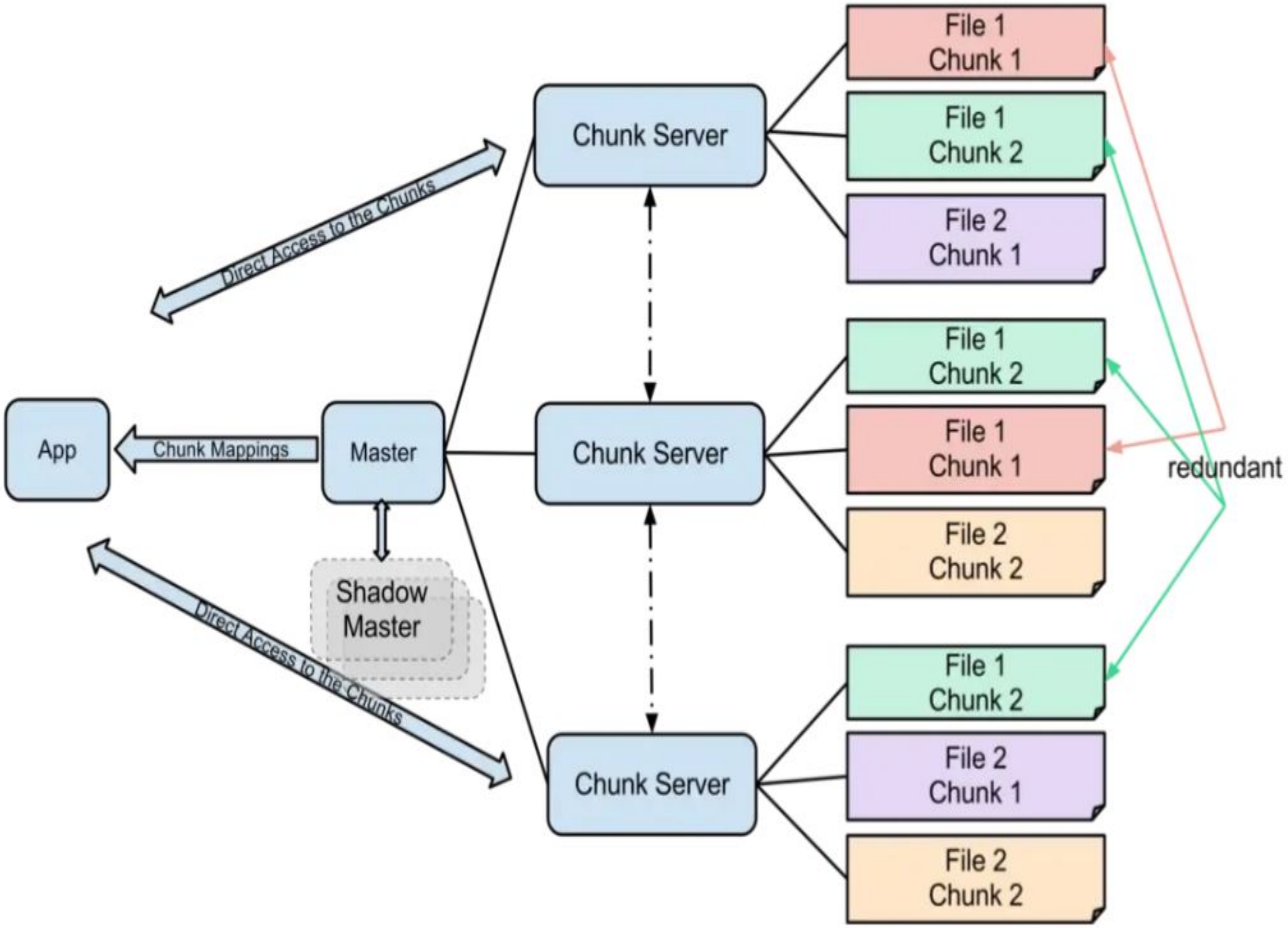
- Staleness in GFS can occur for several reasons:
- **Data Corruption:** If a replica becomes corrupted due to hardware failures, software bugs, or other issues, it may be considered stale. This corruption could make the replica inconsistent with the correct data.
- **Replica Unavailability:** If a replica is temporarily unavailable or unreachable, it may be considered stale until it becomes accessible again.

Master Operation : Stale Replica

- To handle staleness, GFS employs mechanisms such as:
- **Periodic Health Checks:** Regular checks to ensure the health and consistency of replicas.
- **Replica Rejuvenation:** Replacing or updating stale replicas with fresh copies to maintain consistency.
- **Garbage Collection:** Removing or repairing outdated replicas to prevent them from affecting system reliability.
- These mechanisms help GFS maintain data integrity and consistency in a distributed environment. Staleness is an important consideration in distributed file systems, as it can impact the correctness and reliability of data stored across multiple nodes.

Operation Log

- The operation log contains a history of the key metadata changes.
- The operation log (the historical changes in metadata record) is not only the storage record of metadata but also serves as a logical timeline that defines the sequence of concurrent operations. Files and blocks and their versions are unique and permanently identified by the logical time at which they were created
- The log will be copied to multiple remote devices after the corresponding log records are written to the hard disks of the local and remote devices.
- The master server restores its file system state by repeating the operation log. To minimize startup time, we have to keep the log small. Whenever the log grows beyond a certain size, the master server generates a checkpoint on its state so that it can re-execute only a limited number of times after the checkpoint from disk log records for recovery



App
Chunk Mappings
Master
Shadow Master
Direct Access to the Chunks
Direct Access to the Chunks
Chunk Server
Chunk Server
Chunk Server
File 1 Chunk 1
File 1 Chunk 2
File 2 Chunk 1
File 1 Chunk 2
File 1 Chunk 1
File 2 Chunk 2
File 1 Chunk 2
File 2 Chunk 1
File 2 Chunk 2
redundant

Hot spot

- It is possible for multiple clients to access the same chunk of data simultaneously, known as a "hot spot." This can occur when a particular piece of data is frequently accessed or modified by multiple clients, leading to a high level of contention for access to that data.
- There are several strategies that can be used to mitigate the effects of hot spots in a distributed system:
  – Data partitioning: By dividing the data into smaller chunks and distributing them across different nodes, it is possible to reduce the likelihood of a hot spot forming.

Hot spot

- Caching: Caching can be used to store frequently accessed data locally, reducing the need to access the central data store.
- Load balancing: By distributing client requests across multiple nodes, it is possible to balance the load and reduce the impact of hot spots.
- Replication: By replicating data across multiple nodes, it is possible to provide multiple copies of the data, reducing the contention for access to a single copy.
- Optimistic concurrency control: By using optimistic concurrency control, it is possible to allow multiple clients to access and modify the same data concurrently, and then resolve any conflicts when the changes are committed back to the central store.

Client Node

- Client node is linked with the application that implements GFS API.
- It communicates with the master and the chunk server to read or write data.
- Client communicates with master to get the metadata.
- For read and write, client directly communicates with the chunk server.

Chunk Leases

- In Google File System (GFS), a lease-based protocol is used to ensure consistency and prevent conflicts when multiple clients try to access the same data simultaneously.
- Under this protocol, the master node grants a lease, or temporary ownership, to a client for a specific chunk of data. The client can then make updates to the data without interference from other clients. When the lease expires, the master node checks to see if any other clients have made updates to the data. If no other updates have been made, the original client can renew the lease and continue to make updates. If another client has made updates, the master node resolves the conflict and grants a new lease to the client that has the most recent version of the data.
- This protocol allows GFS to ensure that data remains consistent and up-to-date, even when multiple clients are accessing and updating the same data simultaneously. It also allows GFS to recover quickly from failures, since the master node can re-create missing replicas of data as needed.

Chunk Leases

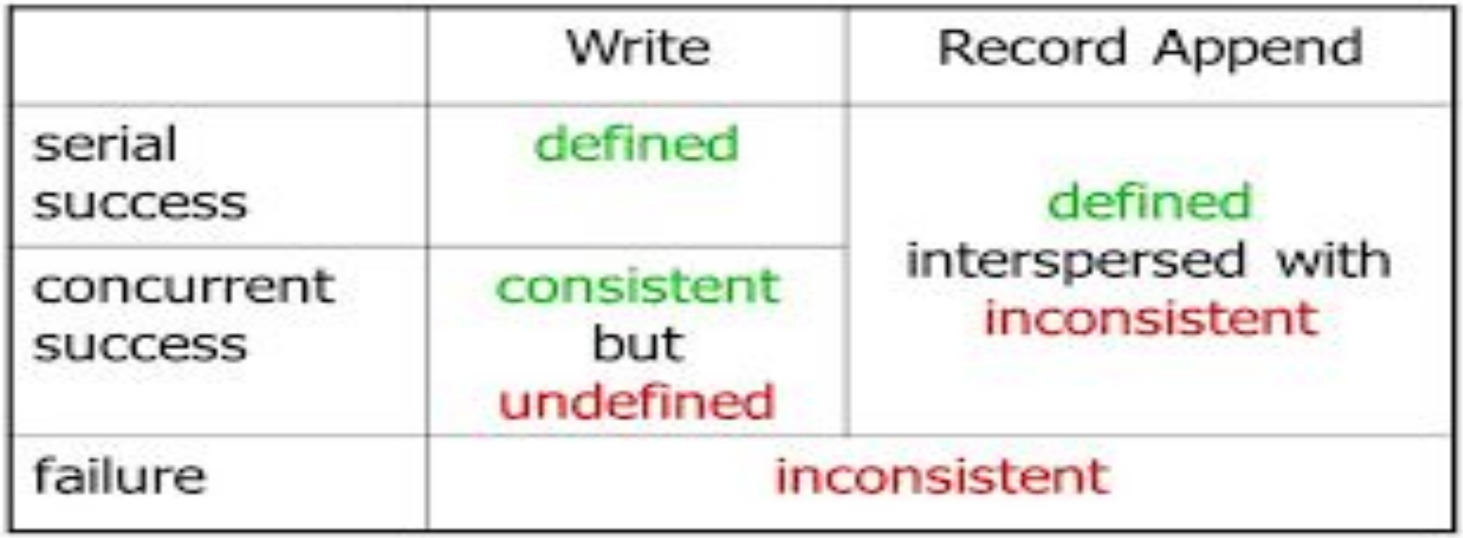
- In Google File System (GFS), a lease-based protocol is used to ensure consistency and prevent conflicts when multiple clients try to access the same data simultaneously.
- Under this protocol, the master node grants a lease, or temporary ownership, to a client for a specific chunk of data. The client can then make updates to the data without interference from other clients. When the lease expires, the master node checks to see if any other clients have made updates to the data. If no other updates have been made, the original client can renew the lease and continue to make updates. If another client has made updates, the master node resolves the conflict and grants a new lease to the client that has the most recent version of the data.
- This protocol allows GFS to ensure that data remains consistent and up-to-date, even when multiple clients are accessing and updating the same data simultaneously. It also allows GFS to recover quickly from failures, since the master node can re-create missing replicas of data as needed.

Chunk Leases

- Since the master is not involved in data operations, it must transfer the power to one of the chunk replicas to determine a serialized mutation order. This is handled by assigning a **lease** to one of the replicas for a particular chunk.
- Chunk leases are revoked after a short timeout, or can be revoked manually by the master if the chunk needs to be shielded from any mutations. For the most part though, chunks which are consistently being mutated will not have their leases revoked as chunkservers can request to extend these leases through communication with the master.

Consistency Model of GFS

- A file region is said to be consistent if all the clients will see the same data each time they load data from any replica of that file region.
- A file region is said to be defined after a file data mutation if it is consistent and the client is able to see what the mutation writes in it.
- The general consistency model is shown in given figure:



	Write	Record Append
serial success	defined	defined interspersed with inconsistent
concurrent success	consistent but undefined	defined interspersed with inconsistent
failure	inconsistent	inconsistent

Interactions in GFS

- **Leases and Mutation Order**
  - A mutation is an operation that changes the metadata of the chunks.
  - It is resulted due to the write or append operations to the file.
  - Leases are used to maintain consistent mutation order across replicas.
  - The master grants chunk lease to one of the replica.
  - The replica is known as primary replica.
  - Primary replica is responsible to design the order for all the mutations to the chunk.

Interactions in GFS

- **Record Append**
  - It is the atomic append operation provided by GFS.
  - The client specifies the data only. The GFS automatically appends it to the file at least once at any offset chosen by the GFS itself and returns the offset to the client.
  - The client then pushes the data to all the replicas of the last chunk of that file and then sends request to the primary replica.
  - The primary replica checks whether appending process will exceed the chunk size. If it exceeds, then it pads the chunk to the maximum size, tells all other replicas to do so and replies client indication reoperation on next chunk. Otherwise, the data is appended on the chunk. And replies the client with success.
  - For operation to be success, the data should be written at the same offset on all the replicas of same chunk.

Interactions in GFS

- **Snapshot**:
  - The snapshot operation is responsible to makes a copy of a file or a directory tree.
  - The client uses snapshot to create branch copies of huge data sets or to checkpoint the current state.
  - When the master receives snapshot request, it revokes any leases on the chunks. After the leases have been revoked or expired, the master logs the operation to disk.
  - The master then applies log record to in-memory state by duplicating the metadata for the source file or directory tree.
  - The created snapshot points at the same chunk as the main files.

Write Control and Data Flow in GFS

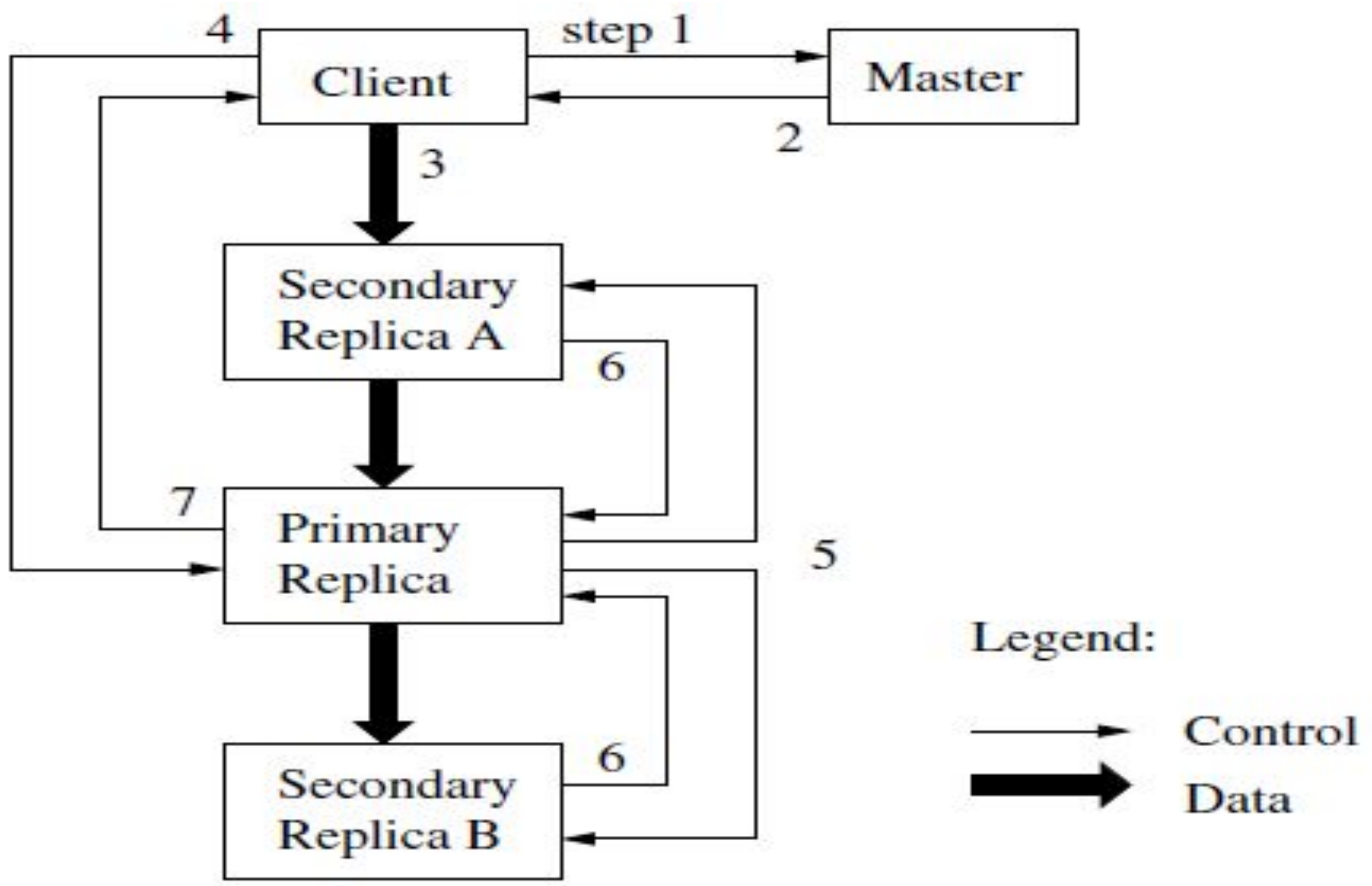


Figure : Write Control and Data Flow

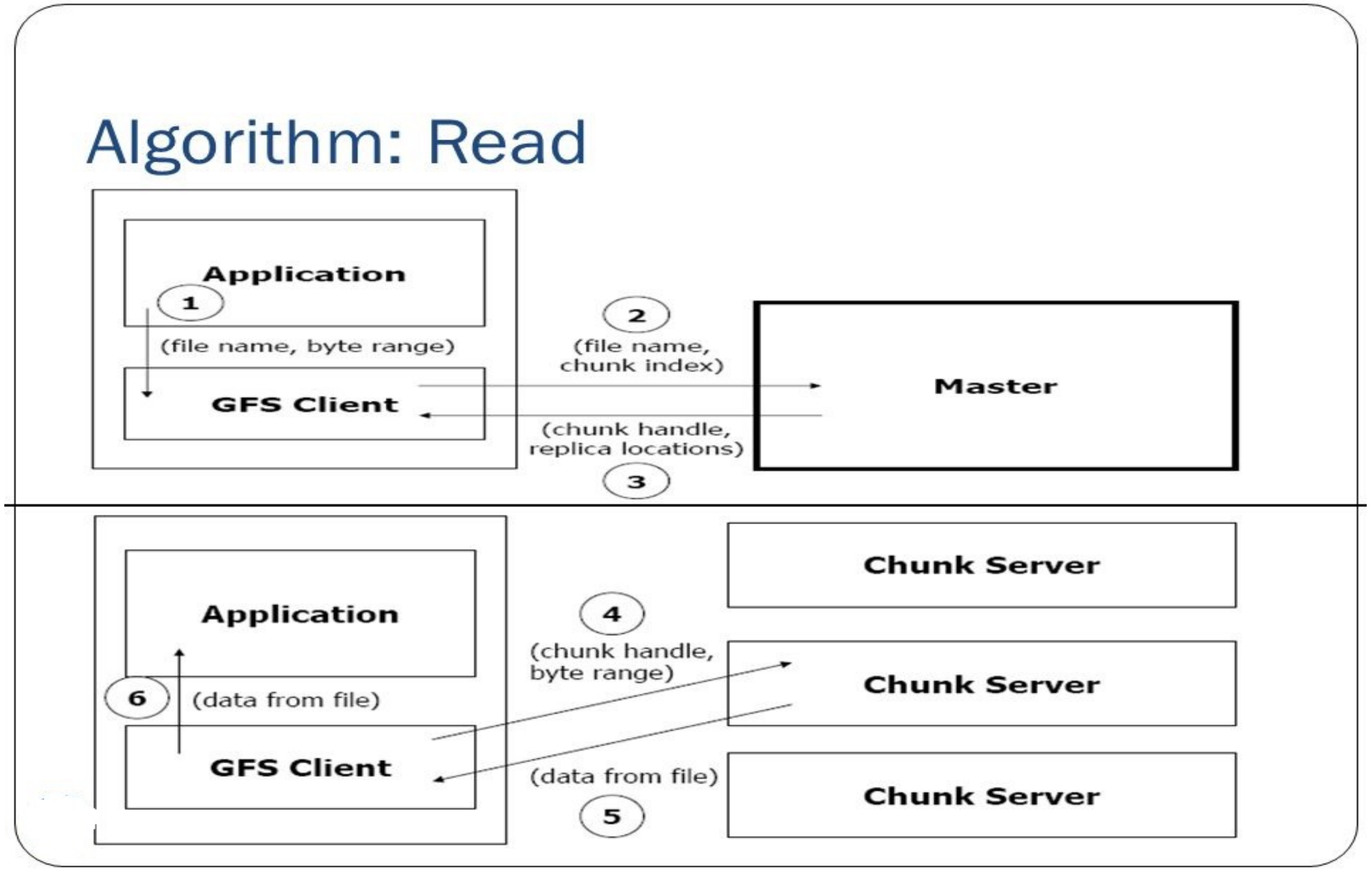
Write Control and Data Flow in GFS

1. 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.
2. 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.
3. The client pushes the data to all the replicas. A client can do so in any order. Each chunk server will store the data in an internal LRU buffer cache until the data is used or aged out.

Write Control and Data Flow in GFS

4. 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 Big Data from multiple clients, which provides the necessary serialization. It applies the mutation to its own local state in serial number order.
5. The primary forwards the write request to all secondary replicas. Each secondary replica applies mutations in the same serial number order assigned by the primary.
6. The secondaries all reply to the primary indicating that they have completed the operation.
7. The primary replies to the client. Any errors encountered at any of the replicas are reported to the client.

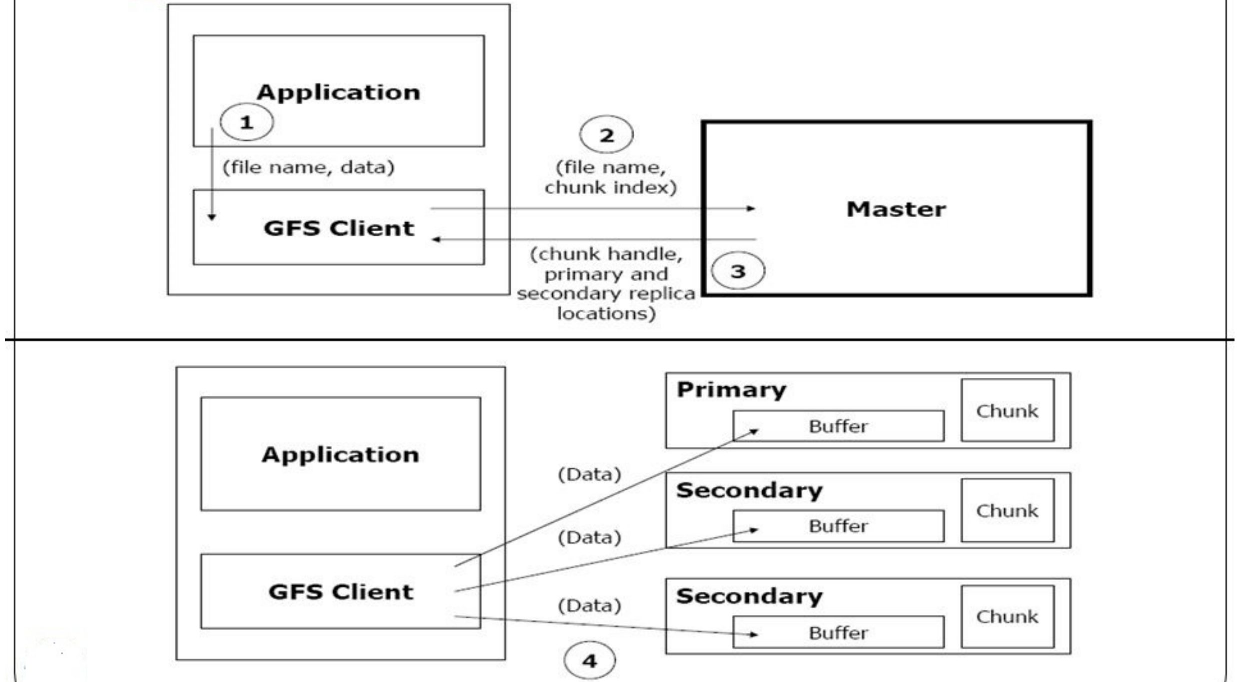
Read Algorithm

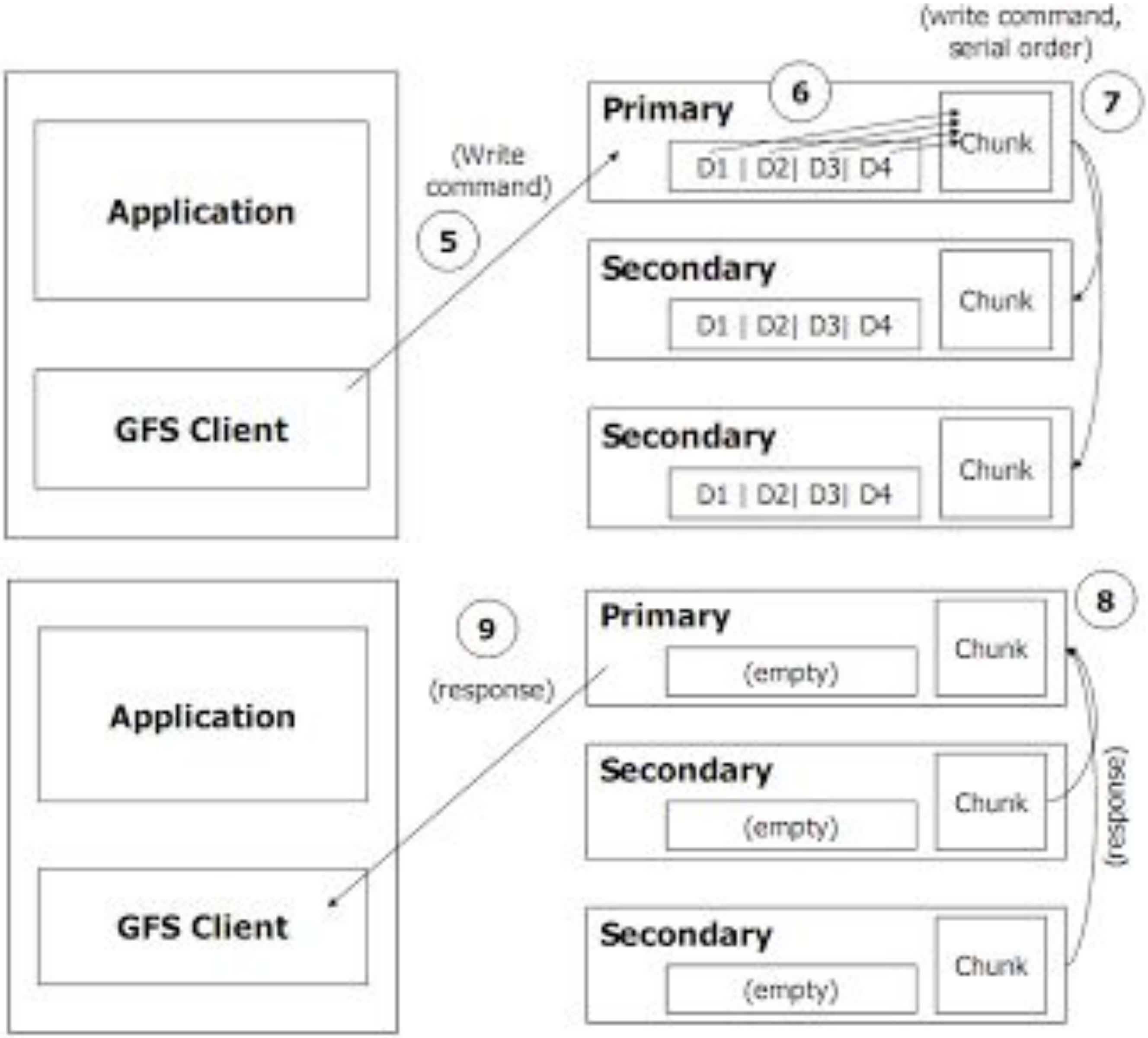


Read Algorithm

1. Application originates the read request.
2. GFS client translates the request from (filename, byte range) -> (filename, chunk index), and sends it to master.
3. Master responds with chunk handle and replica locations (i.e. chunkservers where the replicas are stored).
4. Client picks a location and sends the (chunk handle, byte range) request to that location.
5. Chunkserver sends requested data to the client.
6. Client forwards the data to the application.

Algorithm: Write




(write command, serial order)
6
7
Primary
Chunk
D1 | D2| D3| D4
(Write command)
5
Application
GFS Client
Secondary
Chunk
D1 | D2| D3| D4
Secondary
Chunk
D1 | D2| D3| D4
8
9
(response)
Primary
(empty)
Chunk
Application
Secondary
(empty)
Chunk
(response)
GFS Client
Secondary
(empty)
Chunk

Write Algorithm

- Application originates write request
- GFS client translates request from (filename, data)->(filename, chunk index) and send it to master
- Master responds with chunk handle and (primary + secondary) replica locations
- Client pushes write data to all location. Data is stored in chunk servers internal buffers
- Client sends write command to primary
- Primary determines serial order for data instances stored in its buffer and writes the instances in that order to the chunk
- Primary sends serial order to the secondaries and tells them to perform the write
- Secondaries respond to the primary
- Primary responds back to client

Fault Tolerance

- Google File System (GFS) is a distributed file system that was designed to be highly fault-tolerant, meaning it can continue functioning even if some of its components fail. Here are some of the ways that GFS achieves fault tolerance:
  - Data replication: GFS stores multiple replicas of each chunk of data on different servers to ensure that data is available even if some servers fail.
  - Chunk servers: GFS uses chunk servers to store and manage data chunks, allowing the file system to continue functioning even if some servers fail.
  - Single master node: GFS uses a single master node to manage the file system metadata and coordinate access to data. The master node can be replicated to ensure availability.
  - Lease-based protocol: GFS uses a lease-based protocol to ensure consistency and prevent conflicts when multiple clients access the same data simultaneously. This helps to ensure that data remains up-to-date even if some clients or servers fail.
- Overall, GFS is designed to be highly fault-tolerant, ensuring that it can continue to function and provide access to data even in the event of failures or other disruptions.

References

- Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung The Google File System
- https://www.linkedin.com/pulse/google-file-system-gfs-comprehensive-summary-reuben-lilo-jr-
- https://abhishekkumar2718.github.io/programming/2020-03-15-google-file-system.html
- https://medium.com/geekculture/google-file-system-architecture-cdeabef3f1ea
- https://medium.com/swlh/google-file-system-paper-review-4af386330669
- https://medium.com/@roshan3munjal/google-file-system-gfs-overview-eed15f3e6f6e

Shadow Master

- In the context of the Google File System (GFS) architecture, the term "shadow master" is often used to refer to a backup or replica of the primary master server. The shadow master plays a crucial role in ensuring fault tolerance and high availability in the event that the primary master fails.
- Here's how the concept of a shadow master is typically implemented in GFS:
- **Master Replication:**
  - The state of the primary master server is periodically replicated to one or more backup machines, creating shadow masters.
  - Replication involves copying the metadata, including the file namespace, file-to-chunk mappings, and other critical information maintained by the master.
- **Quick Master Recovery:**
  - In the event of a failure of the primary master, one of the shadow masters can be quickly promoted to the role of the primary master.
  - This promotion allows the system to maintain continuity in metadata operations and file system management.
- **Redundancy and Fault Tolerance:**
  - The existence of shadow masters contributes to the overall fault tolerance of the GFS architecture. If the primary master becomes unavailable, a shadow master can take over without significant disruption to file system operations.
- **Operation Logs:**
  - The master maintains operation logs that record changes and operations related to metadata. These logs are also replicated to ensure that the shadow master has the most recent information.
- The concept of a shadow master helps address the potential single point of failure associated with having a single master in the GFS architecture. By having a replicated and up-to-date backup of the master's state, the system can quickly recover in the event of a master failure, minimizing downtime and ensuring the continued availability and reliability of the file system.

Google File System architecture

- The Google File System (GFS) is a distributed file system developed by Google to manage and store large volumes of data across a cluster of commodity hardware. The architecture of GFS is designed to meet the demands of Google's distributed computing environment, which involves processing massive amounts of data across a large number of machines. Here are the key components and features of the Google File System architecture:
- **Master Server:**
  - GFS has a single master server that is responsible for coordinating and managing metadata operations. The master maintains metadata such as the file namespace, file-to-chunk mappings, and access control information.
  - While there is a single master, its state is periodically replicated to other machines to ensure fault tolerance. Replication allows for quick recovery in the event of a master failure.
- **Chunk Servers:**
  - Chunk servers are responsible for storing the actual data in the form of fixed-size chunks (typically 64 MB). These chunks are the basic units of storage in GFS.
  - Data is distributed and replicated across multiple chunk servers to provide fault tolerance and high availability. Each chunk is replicated across multiple servers, typically three, to ensure data durability.
- **Chunks:**
  - Chunks are the fundamental units of data storage in GFS. Each file is divided into fixed-size chunks, and each chunk is assigned a unique identifier. Chunks are stored on chunk servers.
- **Clients:**
  - Clients are the entities that interact with the GFS for reading and writing data. Clients communicate with the master for metadata operations, such as file creation and file-to-chunk mapping. Actual data reads and writes are performed directly with chunk servers.
- **Namespace:**
  - The master maintains the namespace, which is a hierarchical structure that represents the file system's directory and file structure. It keeps track of file names, directories, and their corresponding metadata.
- **Operation Logs:**
  - The master maintains an operation log, recording metadata changes and operations. This log is used for recovery in the event of a master failure. It allows the master to replay operations and restore its state to a consistent point.
- **Replication and Fault Tolerance:**
  - GFS replicates chunks across multiple chunk servers to ensure fault tolerance. Data is stored redundantly, and in the event of a chunk server failure, data can be retrieved from replicas.
  - The master server replication, regular checkpoints, and operation logs contribute to the fault-tolerant nature of GFS.
- **Data Flow:**
  - When a client needs to read or write data, it communicates directly with the relevant chunk servers. The master is involved in metadata operations, such as file creation, deletion, and file-to-chunk mapping.
  - The master's role is primarily for metadata coordination, and the actual data is read and written directly between clients and chunk servers.
- Overall, the GFS architecture is designed for scalability, fault tolerance, and efficient storage and retrieval of large datasets, making it well-suited for Google's distributed computing needs. The system optimizes for high-throughput access to large files and is capable of handling the challenges associated with a massive-scale distributed environment.

Single Master does not become a bottleneck

- In the Google File System (GFS) architecture, the use of a single master does introduce a potential point of contention and coordination, but Google has designed the system to mitigate this and avoid it becoming a bottleneck under normal operating conditions. Here are some key features and design choices in GFS that contribute to the master not becoming a bottleneck:
- **Metadata Operations:**
  - The master in GFS primarily handles metadata operations, such as namespace management and file-to-chunk mappings. Data read and write operations, on the other hand, involve direct communication between clients and chunk servers without the need for the master's involvement.
- **Decentralized Data Storage:**
  - The actual data storage is decentralized across multiple chunk servers. Clients can read and write data directly to these chunk servers without needing to go through the master for every operation. This decentralized approach reduces the load on the master for data access operations.
- **Chunk Replication and Distribution:**
  - GFS replicates data across multiple chunk servers to ensure fault tolerance. The distribution and replication of data are handled by the chunk servers themselves, reducing the need for constant intervention from the master.
- **Caching:**
  - The master uses caching to store frequently accessed metadata, reducing the need to repeatedly fetch metadata for common operations. This caching mechanism helps in speeding up metadata-related operations.
- **Large Chunk Size:**
  - GFS uses relatively large chunk sizes (64 MB by default). This design choice minimizes the metadata overhead, as each file consists of a smaller number of large chunks, reducing the frequency of metadata operations.
- **Master Failover:**
  - Although there is a single master at any given time, GFS employs mechanisms for quick master recovery in the event of a failure. The use of master replication, regular checkpoints, and operation logs allows for a new master to be quickly brought up in case of a failure, minimizing downtime.
- While the master does play a critical role in managing metadata and coordinating certain aspects of the file system, these design choices and features help ensure that the master is not a performance bottleneck in typical operating scenarios. However, the system's efficiency also relies on proper tuning, sizing, and configuration based on the specific requirements of the applications using GFS.

Fault-tolerant despite having a single master

- In the Google File System (GFS) architecture, there is indeed a single master (also known as the "master server" or "master node"). The master is responsible for managing metadata, coordinating access to files, and maintaining the overall namespace of the file system. However, Google has implemented mechanisms to ensure that the single master does not become a single point of failure. Here's how GFS achieves fault tolerance despite having a single master:
- **Master Replication:**
  - While there is a single master in the system, its state is periodically replicated to other machines. In the event of a master failure, one of the replica masters can be quickly promoted to the role of the primary master.
- **Regular Checkpointing:**
  - The master regularly takes checkpoints of its state. These checkpoints are stored on the distributed file system. In case of a master failure, the system can use the most recent checkpoint to recover the master's state.
- **Operation Logging:**
  - The master logs all file system operations. This log is stored in a reliable and distributed manner. In case of a master failure, the system can use the log to replay operations and recover the master's state up to the point of failure.
- **Quick Master Recovery:**
  - When a master failure occurs, the recovery process is designed to be quick. A new master can be brought up and recover its state from checkpoints and operation logs, minimizing downtime.
- While there is a single master at any given time, these measures ensure that the system can quickly recover from master failures and maintain the continuity of file system operations. The use of replication, regular checkpoints, and operation logging are key elements in making the GFS architecture fault-tolerant despite having a single master.

Master Operation

- In the Google File System (GFS), the GFS master is a special machine that is responsible for managing the file system. It stores metadata about the file system, such as the names and locations of files and directories, and it coordinates access to the file system by clients.
- The GFS master performs several important functions:
  - **File and directory creation**: When a client creates a new file or directory, it sends a request to the GFS master, which updates the metadata to reflect the new file or directory.
  - **Lease management:** When a client wants to read or write a file, it must request a lease from the GFS master. The master grants the lease if no other client currently has a write lease on the file, and the client can then read or write the file until the lease expires.
  - **Chunk management:** GFS stores data in fixed-size chunks, typically 64MB in size. The GFS master is responsible for storing metadata about the location of each chunk, and it returns this metadata to clients when they request data from the file system.
  - **Replication:** GFS stores multiple copies of each chunk to provide fault tolerance. The GFS master is responsible for managing the replication of chunks and ensuring that the appropriate number of copies are stored on different machines in the cluster.
  - **Load balancing:** The GFS master monitors the workload of each machine in the cluster and tries to distribute the workload evenly across the machines to ensure good performance.
  - **Recovery:** If a machine fails or becomes unavailable, the GFS master is responsible for recovering data from the failed machine and re-replicating it on other machines in the cluster.

Master Operation : Stale Replica

- A replica is said to be stale if a chunk server fails and misses mutations to the chunk while it is down.
- The master is responsible to maintain a chunk version number to distinguish between up-to-date replica and stale replica.