GOOGLE FILE SYSTEM

OVERVIEW :

Google File System is divided into three parts :

1. Client
2. Master Server
3. Chunk Manager Server

1. Client :
   Client facilitates the requests made by the applications to the Master server by modifying the request into proper metadata format which can be understood by the master server eg. when application requests the filename for reading, the GFS client calculates the chunk index of the file using the chunk size and the file size and sends it further to the master server.

2. Master Server :
   Master server is responsible for maintaining the file namespace and the mapping between various chunk handles and the chunk servers in which a specific chunk resides. It supervises garbage collection, re-replication when a chunk server is down, chunk migration for balancing load and disk space.

3. Chunk Manager Server :
   These are inexpensive commodity hardware where data is actually stored. For any READ, WRITE operation data flows directly between the client and the chunkserver.

IMPLEMENTATION DETAILS :

Remote Procedure Call (RPC) is used for communication between the aforementioned three parts of the Google File System.

RPC Calls registered at Master Server :

1. UNIQUE_CLIENT_ID - When a client is attached to master it calls this function to get a unique client id.

2. CREATE - This function is called by the client to create a file in the namespace. The client passes name of the file as an argument and will get True if successful or errors if any.

3. ADD_CHUNK - Client calls this function to get new chunk handle. It passes the name of the file and chunk index as the parameter and in return, master sends a unique chunk handle along with a list of addresses of chunk servers that will store the chunk handle.

4. FIND_LOCATIONS- This function is called by client to get the chunk handle and respective chunk locations stored with master server. Client passes the filename and chunk index to master and in response it receives the chunk handle and chunk server locations where it is stored.
5. FIND_LEASE_HOLDER- Client calls this function to get the PRIMARY chunk server for a given chunk handle.
   If there is no current lease holder, master will automatically select one of the replicas to be primary, and grant lease to that chunk server and return it to the client.

6. REPORT_CHUNK: This function is called by chunk servers to tell the master that they have a certain chunk and the number of bytes defined in that chunk. Whenever Chunk Server write any data to any chunk it uses this function to notify master about the changes made to any chunk.

7. CREATE_DIR: This function is called by client to create a directory in the namespace.Client provide the name of the directory and in return master server confirms if this directory is created or not along with any errors, if present.

8. LIST_ALLFILES: This function will be called by a client to list all files present in a given directory name. The client passes the name of the directory and in return, the master server sends a list of all files present in a given directory.

9. DELETE: This function will be called by client to delete a specific file or a directory. Master checks if given filename is a directory or file.
   → If directory, it validates if it is empty or not. If empty, it deletes the directory and send success message to client.
   → If file, it sends all chunk handles related to file for garbage collection and then deletes the file from namespace.

10. GET_FILE_LENGTH: This function will be called by client to get length of specified filename.

11. NOTIFY_MASTER: When chunk server is up, it calls this function to notify master of its presence. It sends master its address so that master can use this address for future communication.

RPC calls registered at Chunk Server:

1. PUSH_DATA: It handles client RPC request to store data in memory.Client sends the data to be stored to all chunk server location it received from master server. Along with data it sends its client id and timestamp to the chunk server so that chunk server can uniquely identify data it receives from any client.

2. WRITE: It handles client RPC request to the primary chunk.Primary Chunk first applies requested write to its local storage , serializes and records the order of application in which write requests are processed then send the write request to all secondary replicas.

3. SERIALIZED_WRITE: This RPC is called by primary replicas once data send by client is stored persistently on disk. Now Primary replicas uses this function to send write request to all secondary replicas to make data persistent on all secondary chunk servers.

4. READ: This RPC is called by client to read content related to specific chunk. Client passes the chunk_handle, offset and length of data to be read to the chunk server. Chunk server reads the data from given chunk handle and sends it back to the client.

5. APPEND: This RPC is called by client whenever to append data to the existing file.Data is only appended if its size is less than the append size, which is generally one fourth of chunk size.

6. ORDER_CHUNK_COPY_FROM_PEER: This RPC is called by master to order a chunk server to copy some chunks from peers chunk server so as to meet the replication goal for that chunk.Whenever some chunkserver is down, master calls this RPC on other chunk server to maintain the replication factor of chunks present in dead chunk server.

7. GET_CHUNK_INFO_FROM_PEER: This RPC is used by a chunk server to connect to another chunk server in order to get the chunk data. Master provides the information related to chunk handle and location of chunk server from where chunk handle needs to be copied.

DESIGN OVERVIEW :

MASTER SERVER :

Master is a single process running on a separate machine that stores all metadata, e.g file namespace, file to chunk mappings, chunk location information etc. Client contacts master to get the metadata to contact the chunk servers.
→ Master operations are fast, as all metadata is stored in memory. Furthermore, it is easy and efficient for the master to periodically scan through its entire state in the background. This periodic scanning is used to implement chunk garbage collection, re-replication in the presence of chunkserver failures, and chunk migration to balance load and disk space.

All operations performed by master is separated into two different classes:

1. Namespace Manager.
2. Chunk Manager.

Namespace Manager :
It is responsible for all namespace management. We use a map to store all information related to namespace. The key for the map is path string and the value is all the information (e.g. a boolean to isDir indicate if it is a directory, if it is a file, we will record the length of that file.) that should be stored persistently.

Functions supported by Namespace Manager:

1. Create : Master called this function, whenever clients requests for creating a new file. Before creating new file master checks the conditions:
   → Is a file with given filename already present, if present return error.
   → Parent of file should exist and it must be a directory, if not return error.
2. Create_dir: This function is used to create a new directory. Before creating new directory, master checks the condition:
   → If a directory with given name already exist, if present return error.
   → Parent should exist and must be a directory, if not return error.
3. List_allfiles: It lists all files present in a given directory name provided directory exist and it should be a directory.
4. Delete: It deletes the filename from map used to store mapping from filename to related information.

Chunk Manager: It manages all chunk information at the master.It maintains mapping from:

1. (path, chunk_index) → chunk information(persistent).
2. chunk_handle → path , chunk_index
3. Chunk_handle → chunk locations(list of all chunk servers where chunk is stored)
4. Active_chunk_servers → A list of all active chunk servers.
5. Delete_chunk → A list used for garbage collection. It stores all chunk_handles that needs to be deleted from chunk_servers.
6. Chunk_of_chunk_servers → a mapping from chunk_servers to all chunk handles stored at chunk server.

CHUNK SERVER :

These are inexpensive commodity hardware where data is actually stored. For any READ,WRITE operation data flows directly between client and chunkserver.
As chunk server may be down at any given point of time, master replicates the copy of chunks at multiple locations to provide continuous access of data to clients.
→ Whenever chunk server is up, it notifies master of its presence by sending list of all chunk handles it stores.

→ Master and chunk servers regularly communicate with each other through heartbeat mechanism where master notifies chunk servers about bad chunks that needs to be deleted.

SYSTEM INTERACTION :

Master and chunk servers communicate regularly to obtain the state, if the chunkserver is down, if there is any disk corruption, if any replicas got corrupted, which chunk replicas store Chunkserver, etc. Master also sends instruction to the chunk servers for deleting existing Chunks, creating new chunks.

WRITE ALGORITHM :

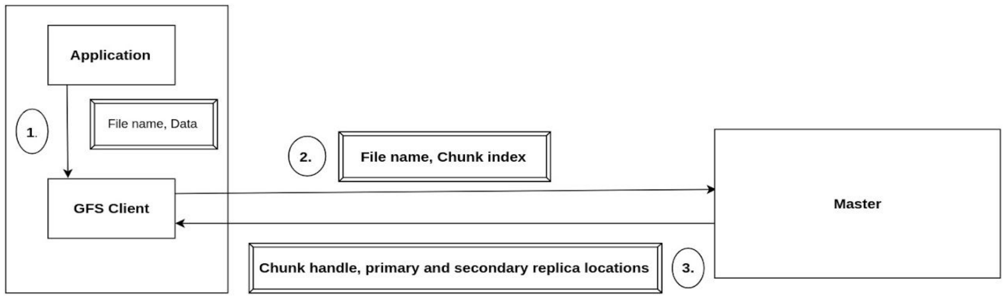
→ A mutation is an operation that changes the contents or metadata of a chunk such as a write or an append operation. Each mutation is performed at all the chunk's replicas.

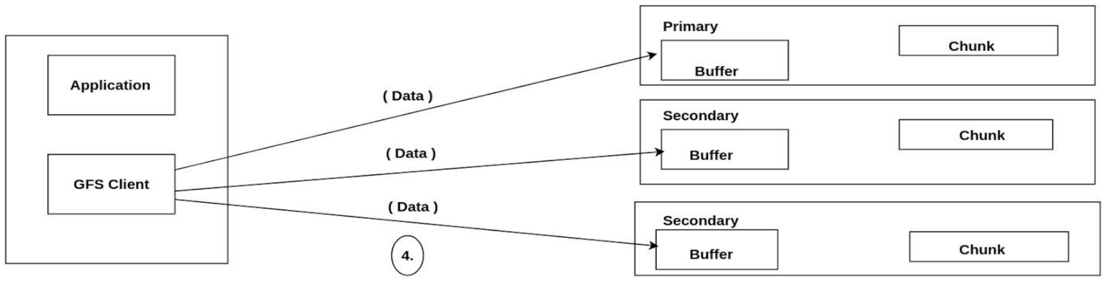
→ Leases are used to maintain a consistent mutation order across replicas. The master grants a chunk lease to one of the replicas, which we call the primary.

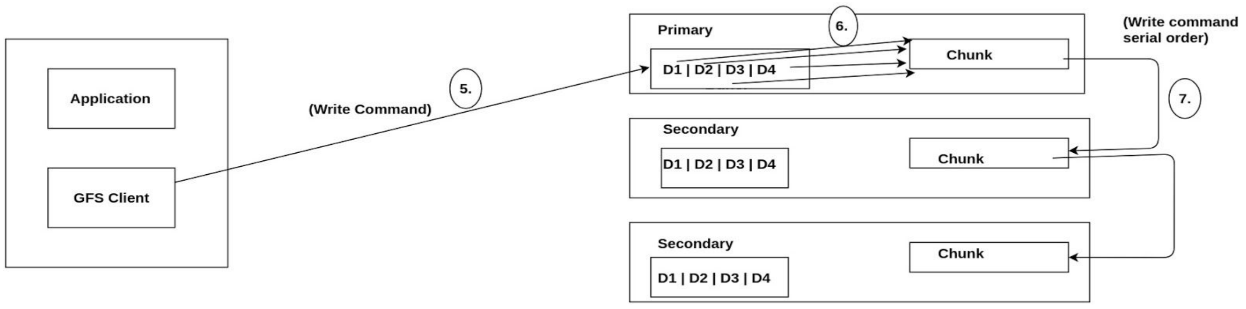
→ The primary picks a serial order for all mutations to the chunk. All replicas follow this order when applying mutations. Thus, the global mutation order is defined first by the lease grant order chosen by the master, and within a lease by the serial numbers assigned by the primary.

STEPS :

1. Application originates the request.
2. GFS client translates request from (filename, data) → (filename, chunk index), and sends it to master.
3. Master responds with chunk handle and (primary + secondary) replica locations.
4. Client pushes write data to all locations. Data is stored in chunk servers' internal buffers.
5. Client sends write command to primary.

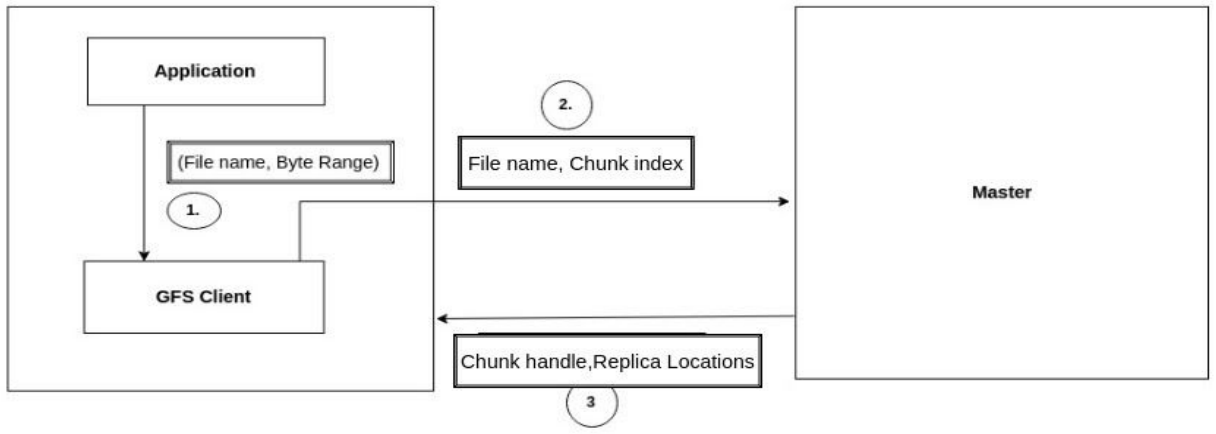


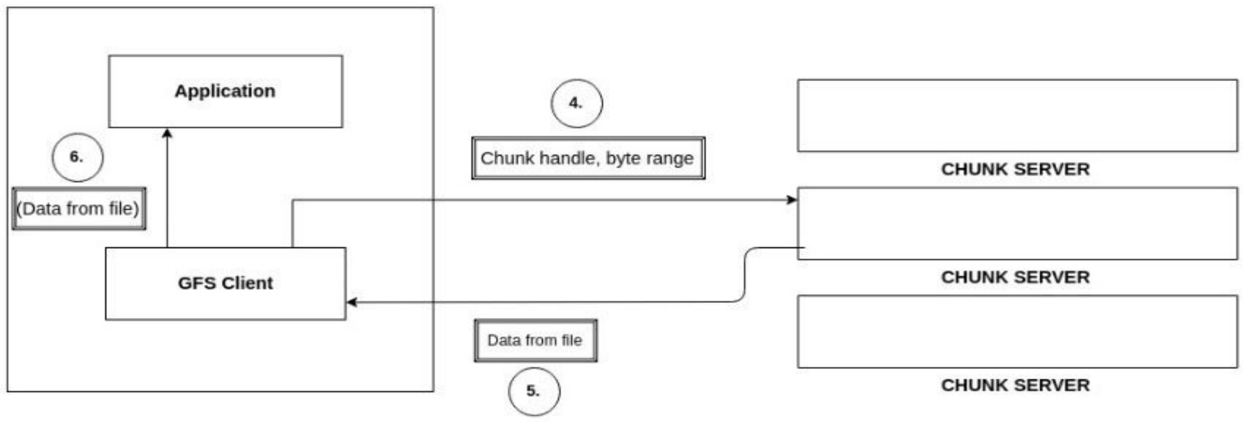




6. Primary determines serial order for data instances stored in its buffer and writes the instances in that order to the chunk.
7. Primary sends the serial order to the secondaries and tells them to perform the write
8. Secondaries respond to the primary.
9. Primary responds back to the client.
10. If write fails at one of chunk servers, client is informed and retries the write.

READ ALGORITHM :





STEPS :

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

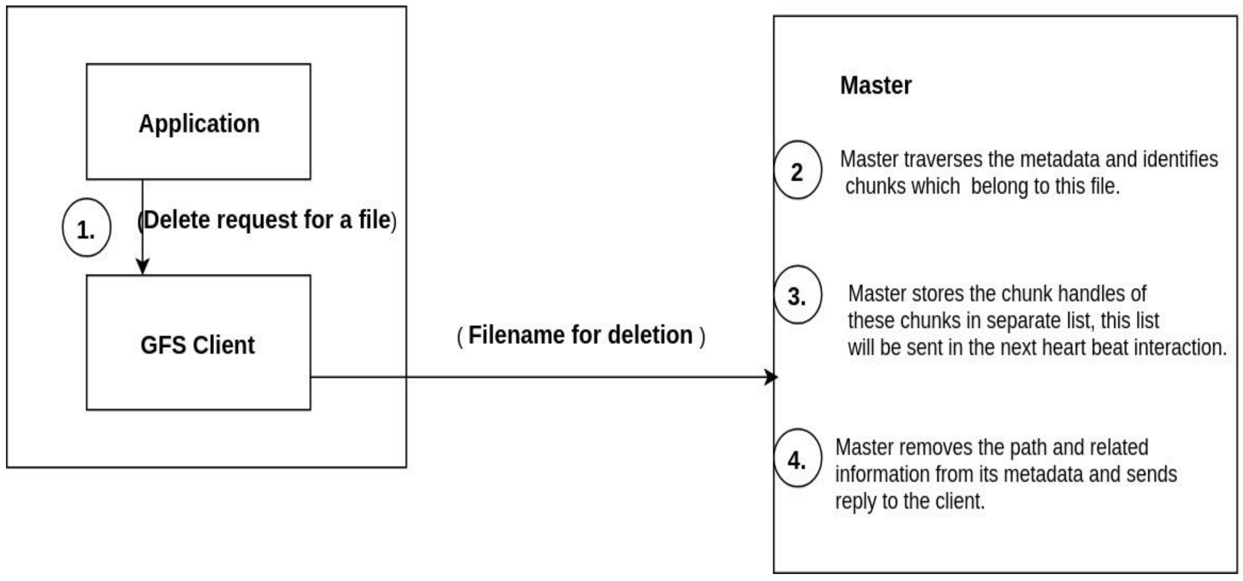
APPEND ALGORITHM :

1. Application originates record append request.
2. GFS client translates requests and sends it to master.
3. Master responds with chunk handle and (primary + secondary) replica locations.
4. Client pushes write data to all replicas of the last chunk of the file.

5. Primary checks if record fits in specified chunk.
6. If record doesn't fit, then the primary:
   → Pads the chunk
   → Tell secondaries to do the same
   → And informs the client
   → Client then retries the append with the next chunk.
7. If record fits, then the primary:
   → Appends the record.
   → Tells secondaries to write data at exact offset
   → Receives responses from secondaries
   → And sends final response to the client.

DELETE ALGORITHM :

1. Client initiates the request for deleting any file to master.
2. Master traverses the metadata and identify all chunks belongs to specific file.
3. Master stores the chunk handle of these chunks in separate list which will be transmitted to chunk servers during next heartbeat interaction with chunk server.
4. Master removes the path and related information from its metadata and send reply to client.


Application
1.
(Delete request for a file)
GFS Client
(Filename for deletion)
Master
2
Master traverses the metadata and identifies chunks which belong to this file.
3.
Master stores the chunk handles of these chunks in separate list, this list will be sent in the next heart beat interaction.
4.
Master removes the path and related information from its metadata and sends reply to the client.

HEARTBEAT MECHANISM :

1. It is used by the master to communicate with all chunk servers periodically.
2. Master pings the chunk servers to check if it is up or down.
3. If a chunk server is up, the master sends the list of bad chunks to be deleted.
4. If chunk server is down, it adds chunk server address to dead chunk list.

5. If any chunk server is dead:
   → Remove it from the list of active chunk servers.
   → For all chunk handles that are stored in dead chunk server location, remove the dead chunk location from CHUNK HANDLE → CHUNK LOCATION mapping.
   → After removing dead chunk location from above mapping, if replication factor of chunk falls below required replication factor, master orders other chunk servers to copy the chunks from chunk server to maintain the minimum replication factor of each chunk.
   → if the chunks are successfully replicated, master removes the dead chunk server entry from chunks_of_chunks_server dictionary.
6. Heartbeat mechanism ensures that the master always contains the updated metadata about all chunks stored at different chunk locations.
7. It ensures that data is always available to client even if some chunk servers are down.
8. It helps in re-replication and balancing of chunks.

REFERENCES :

- GFS Explained
- Original GFS paper
- SGFS paper
- https://docs.python.org/3/library/logging.html
- https://realpython.com/python-logging/
- https://timber.io/blog/the-pythonic-guide-to-logging/
- https://pythonhosted.org/Pyro/ (for RMI)
- https://realpython.com/python-sockets/ (for sockets)
- https://docs.python.org/3/library/xmlrpc.html (for RPC)
- https://stackoverflow.com/a/52128389/5463404 (for Caching)