# Google File System

Aman Sharma – 2018201084
Ankit Pant – 2018201035
Ankush Nagpal – 2018201083
Shreya Upadhaya – 2018201091

## Outline

## Outline

## Introduction

## Introduction

- Google File System – a proprietary file system used by Google

## Introduction

- Google File System – a proprietary file system used by Google
- A distributed file system:
    - Hundreds or thousands of inexpensive storage machines

## Introduction

- Google File System – a proprietary file system used by Google
- A distributed file system:
    - Hundreds or thousands of inexpensive storage machines
    - Constant monitoring, error detection, fault tolerance, and automatic recovery

## Introduction

- Google File System – a proprietary file system used by Google
- A distributed file system:
    - Hundreds or thousands of inexpensive storage machines
    - Constant monitoring, error detection, fault tolerance, and automatic recovery
- Characteristics:
    - Huge files (multi GB) are common

## Introduction

- Google File System – a proprietary file system used by Google
- A distributed file system:
  - Hundreds or thousands of inexpensive storage machines
  - Constant monitoring, error detection, fault tolerance, and automatic recovery
- Characteristics:
  - Huge files (multi GB) are common
  - Once written, the files are only read (most often sequentially)

## Introduction

- Google File System – a proprietary file system used by Google
- A distributed file system:
    - Hundreds or thousands of inexpensive storage machines
    - Constant monitoring, error detection, fault tolerance, and automatic recovery
- Characteristics:
    - Huge files (multi GB) are common
    - Once written, the files are only read (most often sequentially)
    - File writes are very rare

## Introduction

- Google File System – a proprietary file system used by Google
- A distributed file system:
    - Hundreds or thousands of inexpensive storage machines
    - Constant monitoring, error detection, fault tolerance, and automatic recovery
- Characteristics:
    - Huge files (multi GB) are common
    - Once written, the files are only read (most often sequentially)
    - File writes are very rare
    - Most files are mutated by appending new data

## Literature Review

## Literature Review

- GFS is proprietary and Not for Sale

## Literature Review

- GFS is proprietary and Not for Sale
- Hence implementation details are vague

## Literature Review

- GFS is proprietary and Not for Sale
- Hence implementation details are vague
- The references mostly outline the architecture and operations

## Literature Review

- GFS is proprietary and Not for Sale
- Hence implementation details are vague
- The references mostly outline the architecture and operations
- Recent Changes made to reduce latency not publicly documented yet

## GFS Architecture

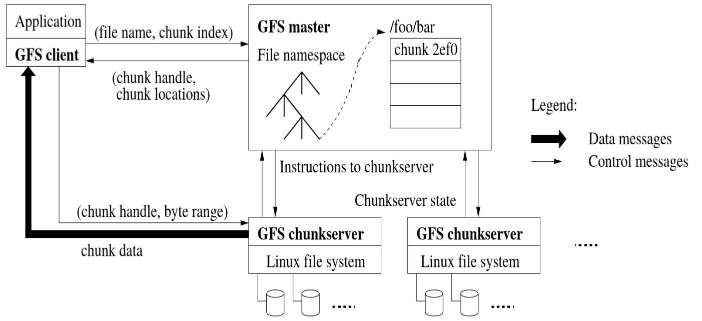

Figure 1: GFS Architecture

The Google File System Architecture [1]

## Methodology

## Methodology

- The implementation of GFS was done in python

## Methodology

- The implementation of GFS was done in python
- The operations supported include:
    - Create File

## Methodology

- The implementation of GFS was done in python
- The operations supported include:
    - Create File
    - Read

## Methodology

- The implementation of GFS was done in python
- The operations supported include:
    - Create File
    - Read
    - Delete File

## Methodology

- The implementation of GFS was done in python
- The operations supported include:
    - Create File
    - Read
    - Delete File
    - Append

## Methodology

- The implementation of GFS was done in python
- The operations supported include:
  - Create File
  - Read
  - Delete File
  - Append
  - Snapshot

## Methodology

- The implementation of GFS was done in python
- The operations supported include:
  - Create File
  - Read
  - Delete File
  - Append
  - Snapshot
  - Load Balancing (chunk replication)

## Methodology

- The implementation of GFS was done in python
- The operations supported include:
    - Create File
    - Read
    - Delete File
    - Append
    - Snapshot
    - Load Balancing (chunk replication)
    - Garbage Collection

## Methodology

- The implementation of GFS was done in python
- The operations supported include:
    - Create File
    - Read
    - Delete File
    - Append
    - Snapshot
    - Load Balancing (chunk replication)
    - Garbage Collection
    - Maintaining Integrity

## Methodology

- The implementation of GFS was done in python
- The operations supported include:
    - Create File
    - Read
    - Delete File
    - Append
    - Snapshot
    - Load Balancing (chunk replication)
    - Garbage Collection
    - Maintaining Integrity
    - Operation Log

## Create File

- Create is implemented as:

## Create File

- Create is implemented as:
    - Client can issue the create command

## Create File

- Create is implemented as:
  - Client can issue the create command
  - Client provides the directory to create file in

## Create File

- Create is implemented as:
    - Client can issue the create command
    - Client provides the directory to create file in
    - Client transfers the data to the Master

## Create File

- Create is implemented as:
    - Client can issue the create command
    - Client provides the directory to create file in
    - Client transfers the data to the Master
    - Master creates chunks of 64MB from the data (adding padding if required)

## Create File

- Create is implemented as:
    - Client can issue the create command
    - Client provides the directory to create file in
    - Client transfers the data to the Master
    - Master creates chunks of 64MB from the data (adding padding if required)
    - Master forwards these chunks to the chunk-servers based on their free disk space

## Read

- Read is implemented as:

## Read

- Read is implemented as:
    - Client can read chunks of various files using read command

## Read

- Read is implemented as:
    - Client can read chunks of various files using read command
    - Client provides the filename to read from

## Read

- Read is implemented as:
    - Client can read chunks of various files using read command
    - Client provides the filename to read from
    - Client provides the chunk index (from bytes offset input by user)

## Read

- Read is implemented as:
  - Client can read chunks of various files using read command
  - Client provides the filename to read from
  - Client provides the chunk index (from bytes offset input by user)
  - Master looks for the file

## Read

- Read is implemented as:
    - Client can read chunks of various files using read command
    - Client provides the filename to read from
    - Client provides the chunk index (from bytes offset input by user)
    - Master looks for the file
    - If found directs the address of closest chunk-server to the client

## Read

- Read is implemented as:
    - Client can read chunks of various files using read command
    - Client provides the filename to read from
    - Client provides the chunk index (from bytes offset input by user)
    - Master looks for the file
    - If found directs the address of closest chunk-server to the client
    - Client communicated with the nearest chunk-server and receives the required data

## Delete File

- Delete is implemented as:
    - Client can issue delete command

## Delete File

- Delete is implemented as:
    - Client can issue delete command
    - Client provides the filename to delete

## Delete File

- Delete is implemented as:
  - Client can issue delete command
  - Client provides the filename to delete
  - Master looks for the file and retrieves the chunk handle

## Delete File

- Delete is implemented as:
  - Client can issue delete command
  - Client provides the filename to delete
  - Master looks for the file and retrieves the chunk handle
  - Master then removes it form the directory structure

## Delete File

- Delete is implemented as:
  - Client can issue delete command
  - Client provides the filename to delete
  - Master looks for the file and retrieves the chunk handle
  - Master then removes it form the directory structure
  - The garbage cleaning mechanism then removes the file in various chunk-servers (which are now treated as orphaned chunks) on subsequent heartbeat (synchronization) messages

## Append

- Append is implemented as:
    - Client can issue append command

## Append

- Append is implemented as:
    - Client can issue append command
    - Master retrieves the chunk handle of the last chunk

## Append

- Append is implemented as:
  - Client can issue append command
  - Master retrieves the chunk handle of the last chunk
  - The chunk servers containing this handle are retrieved and sent to client

## Append

- Append is implemented as:
    - Client can issue append command
    - Master retrieves the chunk handle of the last chunk
    - The chunk servers containing this handle are retrieved and sent to client
    - All append information is pushed on these chunk-servers by the client

## Append

- Append is implemented as:
    - Client can issue append command
    - Master retrieves the chunk handle of the last chunk
    - The chunk servers containing this handle are retrieved and sent to client
    - All append information is pushed on these chunk-servers by the client
    - Client issues write command which writes the pushed information on the chunk-servers

## Snapshot

- Snapshot is implemented as:
    - Client can issue snapshot command

## Snapshot

- Snapshot is implemented as:
    - Client can issue snapshot command
    - Client provides the directory name to snapshot

## Snapshot

- Snapshot is implemented as:
    - Client can issue snapshot command
    - Client provides the directory name to snapshot
    - Master notes the timestamp and looks for the file in the directory

## Snapshot

- Snapshot is implemented as:
  - Client can issue snapshot command
  - Client provides the directory name to snapshot
  - Master notes the timestamp and looks for the file in the directory
  - Master then checks for the nearest slave

## Snapshot

- Snapshot is implemented as:
    - Client can issue snapshot command
    - Client provides the directory name to snapshot
    - Master notes the timestamp and looks for the file in the directory
    - Master then checks for the nearest slave
    - Master then dumps the directory structure and state information in a file

## Snapshot

- Snapshot is implemented as:
    - Client can issue snapshot command
    - Client provides the directory name to snapshot
    - Master notes the timestamp and looks for the file in the directory
    - Master then checks for the nearest slave
    - Master then dumps the directory structure and state information in a file
    - This file is then dumped (written to) the chosen chunk-server

## Chunk Replication

- Chunk-replication is done automatically when a chunk-server goes down or a new chunk-server is added

## Chunk Replication

- Chunk-replication is done automatically when a chunk-server goes down or a new chunk-server is added
- It is implemented as:
  - Master retrieves the state information of the chunk-servers through the heartbeat messages

## Chunk Replication

- Chunk-replication is done automatically when a chunk-server goes down or a new chunk-server is added
- It is implemented as:
  - Master retrieves the state information of the chunk-servers through the heartbeat messages
  - Master then decides the balancing strategy based on the number of chunks and the number of chunk-servers

## Chunk Replication

- Chunk-replication is done automatically when a chunk-server goes down or a new chunk-server is added
- It is implemented as:
    - Master retrieves the state information of the chunk-servers through the heartbeat messages
    - Master then decides the balancing strategy based on the number of chunks and the number of chunk-servers
    - The chunks are then transferred between the chunk-servers based on the balancing strategy

## Garbage Collection

- Garbage collection is done automatically when a chunk becomes orphaned

## Garbage Collection

- Garbage collection is done automatically when a chunk becomes orphaned
- An orphaned chunk is a chunk whose entry is not in the master (because it was deleted)

## Garbage Collection

- Garbage collection is done automatically when a chunk becomes orphaned
- An orphaned chunk is a chunk whose entry is not in the master (because it was deleted)
- It is implemented as:
    - Mater retrieves the state information of the chunk-servers through the heartbeat messages

## Garbage Collection

- Garbage collection is done automatically when a chunk becomes orphaned
- An orphaned chunk is a chunk whose entry is not in the master (because it was deleted)
- It is implemented as:
    - Mater retrieves the state information of the chunk-servers through the heartbeat messages
    - Master then decides finds out the orphaned chunks based on this information

## Garbage Collection

- Garbage collection is done automatically when a chunk becomes orphaned
- An orphaned chunk is a chunk whose entry is not in the master (because it was deleted)
- It is implemented as:
  - Mater retrieves the state information of the chunk-servers through the heartbeat messages
  - Master then decides finds out the orphaned chunks based on this information
  - Master then issues a command to the chunk-server to delete the orphaned chunk in the respective chunk-server

## Maintaining Integrity

- Integrity of various chunks in the system is maintained by using checksum

## Maintaining Integrity

- Integrity of various chunks in the system is maintained by using checksum
- Chunks are logically divided into 64KB block

## Maintaining Integrity

- Integrity of various chunks in the system is maintained by using checksum
- Chunks are logically divided into 64KB block
- It is implemented as:
    - When a chunk-server boots, it calculates the checksum of each chunk and stores it

## Maintaining Integrity

- Integrity of various chunks in the system is maintained by using checksum
- Chunks are logically divided into 64KB block
- It is implemented as:
    - When a chunk-server boots, it calculates the checksum of each chunk and stores it
    - Anytime there is a request to retrieve a particular chunk, the checksum for the particular chunk is calculated again

## Maintaining Integrity

- Integrity of various chunks in the system is maintained by using checksum
- Chunks are logically divided into 64KB block
- It is implemented as:
    - When a chunk-server boots, it calculates the checksum of each chunk and stores it
    - Anytime there is a request to retrieve a particular chunk, the checksum for the particular chunk is calculated again
    - In case this checksum does not match with the checksum initially calculate, master is informed

## Maintaining Integrity

- Integrity of various chunks in the system is maintained by using checksum
- Chunks are logically divided into 64KB block
- It is implemented as:
    - When a chunk-server boots, it calculates the checksum of each chunk and stores it
    - Anytime there is a request to retrieve a particular chunk, the checksum for the particular chunk is calculated again
    - In case this checksum does not match with the checksum initially calculate, master is informed
    - Master then retrieves the other chunk-servers that contain the same chunk

## Maintaining Integrity

- Integrity of various chunks in the system is maintained by using checksum
- Chunks are logically divided into 64KB block
- It is implemented as:
  - When a chunk-server boots, it calculates the checksum of each chunk and stores it
  - Anytime there is a request to retrieve a particular chunk, the checksum for the particular chunk is calculated again
  - In case this checksum does not match with the checksum initially calculate, master is informed
  - Master then retrieves the other chunk-servers that contain the same chunk
  - These chunk-servers then transfer the particular chunk to the primary chunk-server

## Maintaining Integrity

- Integrity of various chunks in the system is maintained by using checksum
- Chunks are logically divided into 64KB block
- It is implemented as:
  - When a chunk-server boots, it calculates the checksum of each chunk and stores it
  - Anytime there is a request to retrieve a particular chunk, the checksum for the particular chunk is calculated again
  - In case this checksum does not match with the checksum initially calculate, master is informed
  - Master then retrieves the other chunk-servers that contain the same chunk
  - These chunk-servers then transfer the particular chunk to the primary chunk-server
  - The chunk-server then transfers the data to the client

## Operation Log

- Operation log is maintained by the Master

## Operation Log

- Operation log is maintained by the Master
- The following information is dumped to the log file

## Operation Log

- Operation log is maintained by the Master
- The following information is dumped to the log file
    - File Name-space: Contains the directory structure

## Operation Log

- Operation log is maintained by the Master
- The following information is dumped to the log file
    - File Name-space: Contains the directory structure
    - Meta-data: Contains the File-Hashname and the corresponding chunk handle and index

## Operation Log

- Operation log is maintained by the Master
- The following information is dumped to the log file
    - File Name-space: Contains the directory structure
    - Meta-data: Contains the File-Hashname and the corresponding chunk handle and index
    - chunk db: list of all chunks (in all slaves)

## Operation Log

- Operation log is maintained by the Master
- The following information is dumped to the log file
  - File Name-space: Contains the directory structure
  - Meta-data: Contains the File-Hashname and the corresponding chunk handle and index
  - chunk db: list of all chunks (in all slaves)
  - slave list: list of active slaves

## Operation Log

- Operation log is maintained by the Master
- The following information is dumped to the log file
    - File Name-space: Contains the directory structure
    - Meta-data: Contains the File-Hashname and the corresponding chunk handle and index
    - chunk db: list of all chunks (in all slaves)
    - slave list: list of active slaves
    - snapshot record: timestamp, IP and port information from where snapshot was invoked

## Conclusion

## Conclusion

- Google File System can be implemented to support large files through chunk mechanism

## Conclusion

- Google File System can be implemented to support large files through chunk mechanism
- The master serves as the main control node and communicator between client and chunks

## Conclusion

- Google File System can be implemented to support large files through chunk mechanism
- The master serves as the main control node and communicator between client and chunks
- Client can issue various commands to perform different operations on files

## Conclusion

- Google File System can be implemented to support large files through chunk mechanism
- The master serves as the main control node and communicator between client and chunks
- Client can issue various commands to perform different operations on files
- The heartbeat messages provide an efficient synchronization mechanism between the master and the chunk-servers

## Conclusion

- Google File System can be implemented to support large files through chunk mechanism
- The master serves as the main control node and communicator between client and chunks
- Client can issue various commands to perform different operations on files
- The heartbeat messages provide an efficient synchronization mechanism between the master and the chunk-servers
- The garbage cleaning mechanism ensures that no redundant chunks are present in any chunk-server

## Conclusion

- Google File System can be implemented to support large files through chunk mechanism
- The master serves as the main control node and communicator between client and chunks
- Client can issue various commands to perform different operations on files
- The heartbeat messages provide an efficient synchronization mechanism between the master and the chunk-servers
- The garbage cleaning mechanism ensures that no redundant chunks are present in any chunk-server
- The load-balancing mechanism ensures equitable distribution of the chunks stored by the various chunk-servers

## Future Scope

## Future Scope

- Some recent changes[2] made to GFS by google include:

## Future Scope

- Some recent changes[2] made to GFS by google include:
    - New applications now require low latency

## Future Scope

- Some recent changes[2] made to GFS by google include:
    - New applications now require low latency
    - Multiple GFS masters now on a pool of chunk-servers

## Future Scope

- Some recent changes[2] made to GFS by google include:
    - New applications now require low latency
    - Multiple GFS masters now on a pool of chunk-servers
    - Multi-cell approach with master for each cell

## Future Scope

- Some recent changes[2] made to GFS by google include:
  - New applications now require low latency
  - Multiple GFS masters now on a pool of chunk-servers
  - Multi-cell approach with master for each cell
  - More than one cell per data center

## Future Scope

- Some recent changes[2] made to GFS by google include:
  - New applications now require low latency
  - Multiple GFS masters now on a pool of chunk-servers
  - Multi-cell approach with master for each cell
  - More than one cell per data center
  - Cells across network function as distinct file systems

## Future Scope

- Some recent changes[2] made to GFS by google include:
    - New applications now require low latency
    - Multiple GFS masters now on a pool of chunk-servers
    - Multi-cell approach with master for each cell
    - More than one cell per data center
    - Cells across network function as distinct file systems
- Implementation can be extended to incorporate these changes

## References

## References

[1] The Google File System; Sanjay Ghemawat et al.
http://www.cs.cornell.edu/courses/cs614/2004sp/papers/gfs.pdf
[2] Google File System
http://google-file-system.wikispaces.asu.edu/

Thank You!