GROUP C

UNIVERSITY OF TEXAS AT DALLAS | 800 W CaMPBELL RD RICHARDSON TX 75080

MIDTERM REPORT

CLOUD COMPUTING PROJECT



Table of Contents

[Group Information 2](#_Toc433371886)

[Details of team members 2](#_Toc433371887)

[Problem Statement 3](#_Toc433371888)

[System architecture 4](#_Toc433371889)

[High-Level Design 5](#_Toc433371890)

[Detailed Design 6](#_Toc433371891)

[Ceph 6](#_Toc433371892)

[GFS 6](#_Toc433371893)

[DHT 7](#_Toc433371894)

[Implementation Details 8](#_Toc433371895)

[Client 8](#_Toc433371896)

[Commons 8](#_Toc433371897)

[Master 9](#_Toc433371898)

[GFS Directory Operations 9](#_Toc433371899)

[Ceph Directory Operations 9](#_Toc433371900)

[DHT Directory Operations 10](#_Toc433371901)

[Problems Encountered 11](#_Toc433371902)

[References 12](#_Toc433371903)

# Group Information

Group: C

Project Name: Directory Structure Maintenance

Team Members: 4

TA: Yongtao Huang

## Details of team members

1. Ekal Golas – exg140230
2. Jayakarthigayan Sridharan – jxs143730
3. Ketan Joshi – kkj140030
4. Sahith Katukuri – sxk145130

# Problem Statement

The project problem statement (Yen, Cloud Computing, 2015) is stated as:-

Compare different methods in implementing directory files, including three solutions

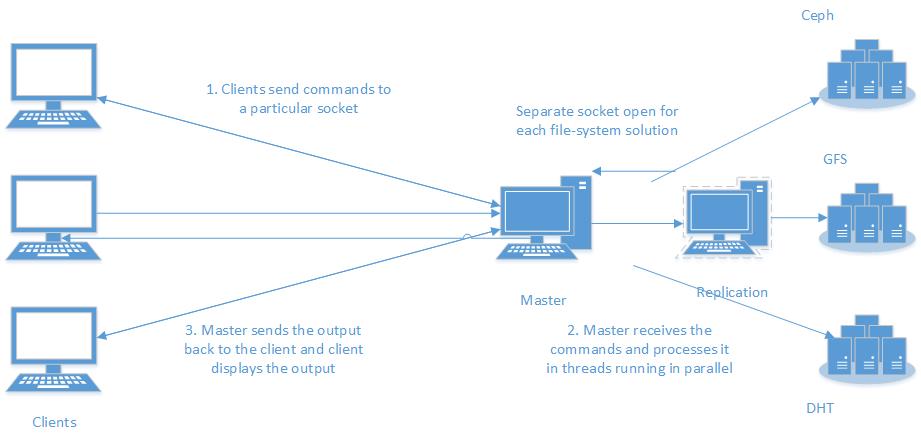
* Solution 1: Use a centralized server to store the entire directory
* Solution 2: Treat directory files as regular files, but may merge a subtree of directories into one file, with a fixed number of levels (the fixed number of levels is configurable)
* Solution 3: Ceph solution

For the midterm report, the progress covers the following:-

* Complete the basic directory maintenance systems
  + Implement all three systems in memory without replication and accept a single request at a time
    - For Ceph, do not consider dynamic load partitioning, but develop the mechanism to decide which partitioning is the best for the system
    - For HDFS, same as Ceph, except that there is no partitioning
    - For Solution 2, Yongtao provides the file system to host the directory files
  + Support create, delete, ls commands
* Implement the basic client
  + Generate the basic directory system on three maintenance systems
  + Generate a mix of client requests for accessing the directories
  + Submit the commands to the three directory management systems
* Support replication
  + Provide replication and master/slave update for HDFS
  + Ceph is the same, except that there are multiple partitions
  + For Solution 2, the system already supports replication

# System architecture

The following diagram depicts the design of client and the master components and the communication between them:-



# High-Level Design

As discussed in the problem statement (Yen, Cloud Computing, 2015), the design for the different solutions and the overall implementation is stated as follows:-

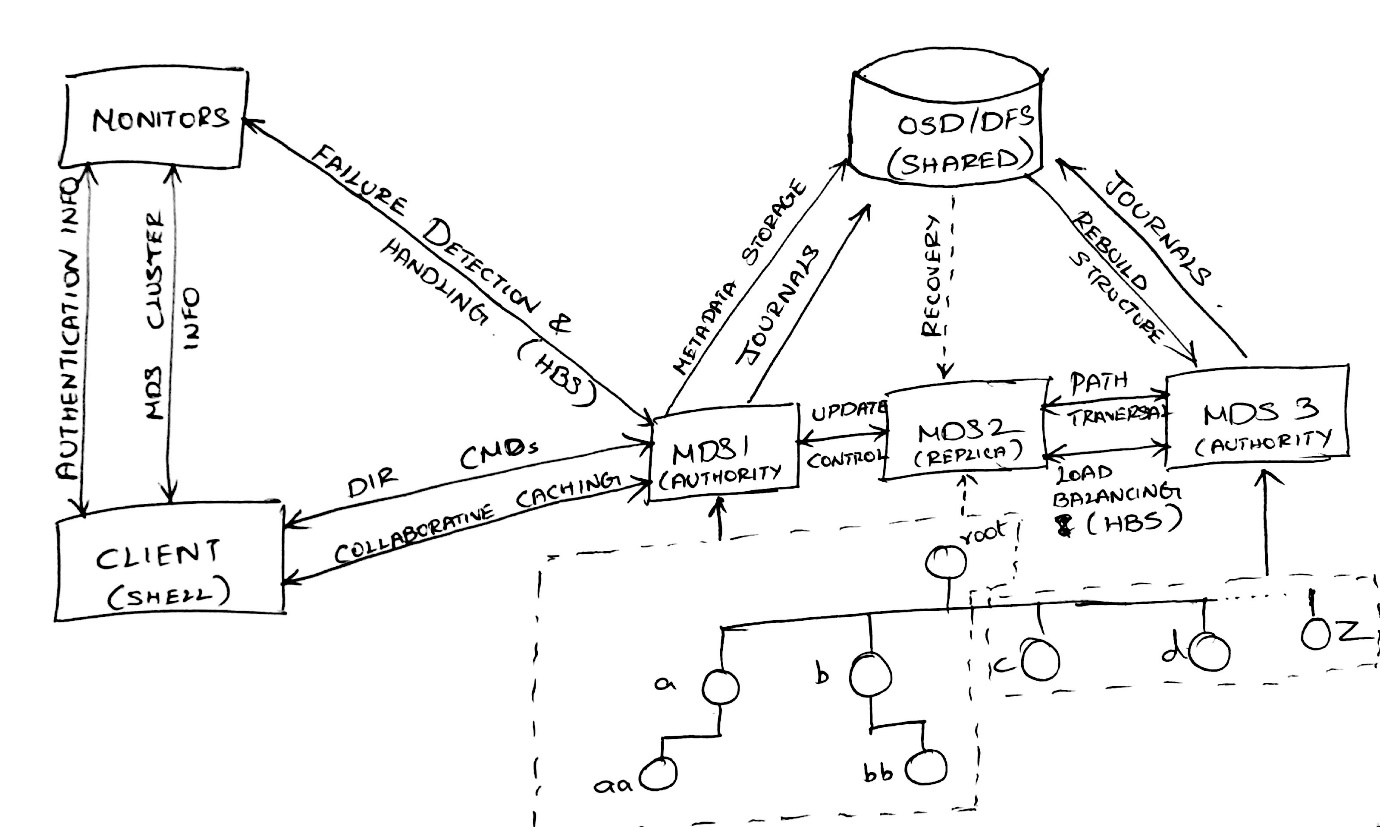
There are three components to the system, each represented as packages which represent the following:-

1. Client – All classes and packages inside it represents the client functionality and implementation
   1. Command generation
   2. Cache
   3. Communication with the master by socket programming
2. Master – All classes and packages inside it represents the master`s functionality, dependencies and implementation
   1. Ceph solution
   2. GFS solution
   3. DHT solution
   4. Communication with multiple clients with socket programming and multithreading
   5. Replication
3. Commons – A package required by both client and master to implemented common utility methods and class representations
   1. Directory representation
   2. Code provided by the TA for communication (danei, 2015)
   3. Configuration and other utilities

# Detailed Design

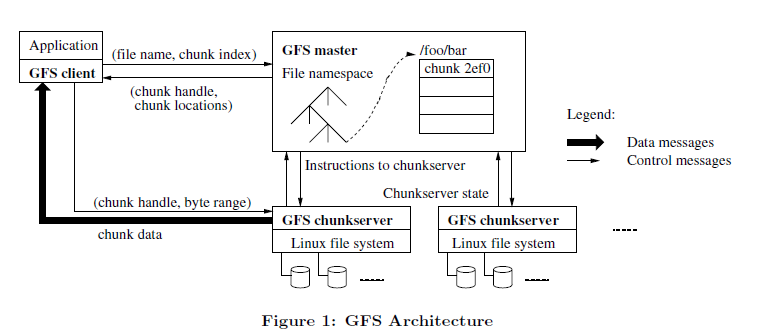
## Ceph

A design was developed using the Ceph paper (University of California, 2006). The design diagram is stated as follows:-



## GFS

A design was developed using the GFS paper (Google, 2003) and the class notes (Yen, Bigdata, 2015). The design diagram is stated as follows:-



## DHT

A design was developed using the DHT filesystem code (Yinzi, 2015) provided by the TA and class notes (Yen, Bigdata, 2015). The design is stated as follows:-

TODO

# Implementation Details

## Client

The client is launched by calling the Client class. It defines a main method which carries out the client operations. After the client class is called and the client is started, the process flow is implemented as:-

1. As the client starts, it calls the AppConfig class in [commons](#_Commons) to load configuration
2. Gets the master`s IP address and port to communicate, and creates a socket on that
3. Uses CommandGenerator class to generate commands to the server.
   1. CommandGenerator class gets the directory structure from the file in the configuration
   2. Gets all possible paths in the directory structure as an array
   3. Randomly shuffles the list of command operations defined in CommandsSupported enum in [commons](#_Commons) and the array of all possible paths
   4. Uses Zipf distribution provided by Apache`s math library to get a weighted distribution of a preconfigured number of commands that will be sent to the server
4. Client obtains the commands and serializes the data using the Message class in [commons](#_Commons)
5. executes them sequentially and displays the output to the console using the OutputFormatter class in [commons](#_Commons)

## Commons

This component is defines utilities and classes used by both client and master. It contains the following:-

1. The code provided by the TA to implement communication between client and server using wrapper over TCP
2. The Directory class that represents the directory structure as a tree. Each node in the tree stores the following information:-
   1. Name of this directory/file
   2. Inode
   3. Access rights
   4. Timestamp last modified
   5. Whether this node is a file or directory
   6. Operation counter to keep track of number of accesses
   7. Size
   8. List of sub-directories at this node
3. DirectoryParser class that reads an input file from configuration which has output of directory hierarchy and creates a directory structure using Directory class
4. Interface ICommandOperations, that defines the directory operations supported as follows:-
   1. Ls – List directory, just the name and type for now
   2. Touch – Create a file
   3. Mkdir – Create a directory
   4. Rmdir – Remove a directory, recursively for now
5. AppConfig class to parse and load the configuration
6. Message class to serialize data for communication between client and master
7. OutputFormatter class to pretty print the output on the console
8. Globals class to store global metadata
9. CommandsSupported enum to list possible commands

## Master

This component defines each directory solution and the working of master server in order to communicate with clients. The master is started by launching the Master class. It defines a main method which carries out the server operations. The process flow is as follows:-

1. As the master starts, it calls the AppConfig class in [commons](#_Commons) to load configuration
2. Opens up a socket for each directory solution
3. Creates the directory structure using DirectoryParser class in [commons](#_Commons) and stores this data in a serialized format to a file read by configuration
4. Starts Worker threads for each socket that listen to clients
   1. As a thread receives a command from a client, it establishes connection with that client
   2. Gets the command and executes it on the desired directory solution
   3. Computes the output and sends it back to the client
   4. Repeats the process until an exit command is issued by the client

Master contains the implementation of different directory solutions. Each of them define their own implementation of commands defined in [commons](#_Commons). These implementations are as follows:-

## GFS Directory Operations

The commands for GFS are implemented as follows:-

1. LS – Performs a search operation on the tree to get the path that matches the path in the directory tree. If the path is not found or the directory is empty, an error message is returned, else we use the OutputFormatter class in [commons](#_Commons) to display the directory information
2. MKDIR – Performs a search similar to LS to get the parent directory. If LS returns an error, error is returned, else a new child node is created in the parent node as a directory.
3. TOUCH – Similar to MKDIR except that the new child gets added as a file.
4. RMDIR - Performs a search similar to LS to get the parent directory. If LS returns an error, error is returned, else the node matching the directory is removed from the list of child nodes of the parent directory

## Ceph Directory Operations

The commands for Ceph are implemented as follows:-

1. LS – Performs a search operation on the tree to get the path that matches the path in the directory tree. If the path is not found or the directory is empty, an error message is returned, else we use the OutputFormatter class in [commons](#_Commons) to display the directory information
2. MKDIR – Performs a search similar to LS to get the parent directory. If LS returns an error, error is returned, else a new child node is created in the parent node as a directory.
3. TOUCH – Similar to MKDIR except that the new child gets added as a file.
4. RMDIR - Performs a search similar to LS to get the parent directory. If LS returns an error, error is returned, else the node matching the directory is removed from the list of child nodes of the parent directory

## DHT Directory Operations

The commands for DHT are implemented as follows:-

1. LS – Performs a search operation on the tree to get the path that matches the path in the directory tree. If the path is not found or the directory is empty, an error message is returned, else we use the OutputFormatter class in [commons](#_Commons) to display the directory information
2. MKDIR – Performs a search similar to LS to get the parent directory. If LS returns an error, error is returned, else a new child node is created in the parent node as a directory.
3. TOUCH – Similar to MKDIR except that the new child gets added as a file.
4. RMDIR - Performs a search similar to LS to get the parent directory. If LS returns an error, error is returned, else the node matching the directory is removed from the list of child nodes of the parent directory

# Problems Encountered

The problems encountered so far are as follows:-

1. Understanding DHT solution – There was a confusion initially around DHT implementation with level cuts which was sorted out after discussions with professor and TA
2. Configuration of static IP address in virtual machines – With the systems setup in the lab, we faced initial difficulties configuring a bridged network between physical hosts and the VMs with static IP addresses. This was resolved after a lot of troubleshooting and help from documentation on the internet
3. End-to-end integration – As we worked on different modules in isolation, there were time-consuming issues and bugs encountered when integrating the code and testing end-to-end functionality

# References

Apache. (2012, May 13). *Log4j*. Retrieved from Logging: https://logging.apache.org/log4j/1.2/download.html

Apache. (2014, Feb 17). *IO*. Retrieved from Commons: https://commons.apache.org/proper/commons-io/download\_io.cgi

Apache. (2014, July 18). *Logging*. Retrieved from Commons: http://commons.apache.org/proper/commons-lang/download\_lang.cgi

Apache. (2015, May 18). *Lang*. Retrieved from Commons: http://commons.apache.org/proper/commons-lang/download\_lang.cgi

Apache. (2015, Sep 11). *Math*. Retrieved from Commons: http://commons.apache.org/proper/commons-math/download\_math.cgi

Apache. (2015, Aug 2). *Pool*. Retrieved from Commons: https://commons.apache.org/proper/commons-pool/download\_pool.cgi

danei. (2015). *Request-response interface*. Retrieved from Bitbucket: https://bitbucket.org/danei/iocontrol/

Google. (2003). *Google File System.* Retrieved from http://static.googleusercontent.com/media/research.google.com/en//archive/gfs-sosp2003.pdf

Google. (2009, June 3). *Collections*. Retrieved from Maven: http://central.maven.org/maven2/com/google/collections/google-collections/1.0-rc2/google-collections-1.0-rc2.jar

Google. (2010, June 1). *Guava*. Retrieved from Maven: http://central.maven.org/maven2/com/google/guava/guava/r05/guava-r05.jar

*Ini4j*. (n.d.). Retrieved from Sourceforge: http://downloads.sourceforge.net/project/ini4j/ini4j-bin/0.5.5-SNAPSHOT/ini4j-0.5.5-SNAPSHOT-bin.zip

University of California. (2006). *Ceph: A Scalable, High-Performance Distributed File System.* Retrieved from http://www.ssrc.ucsc.edu/Papers/weil-osdi06.pdf

Yen, I. L. (2015). *Bigdata.* Retrieved from Class notes: E-learning

Yen, I. L. (2015, 10 12). *Cloud Computing.* Retrieved from I-Ling`s home page: http://www.utdallas.edu/~ilyen/course/cloud/for15f/term-proj-midrep.pdf

Yinzi. (2015). *DHT file system*. Retrieved from Bitbucket: https://bitbucket.org/utdcloudlab/dht-fs