# Design Document

Fangzheng Guo

Zihang Zeng

Aim: Design a fault-tolerant distributed storage system.

Parts: Clients, Directory server, Storage nodes.

## 1. Directory server

The directory server is at a well-known location in our network. It stores the locations of all nodes and the file list of the system, maintains the file-consistency of all nodes. When a client request to connect with a node, it will return the location of the first node on list to the client. When a new file has been added to a node, it will receive an adding request from that node and return the node list to that node to ask it send this file to all other nodes. When a new node is added to the system, including the registration process, it will add the file names holding by the new node to its file list, and ask a node to send all files in system to the new node.

## Functions:

Register(): invoked by a new node to register itself in the system. The directory server will send all files in system to the new node through *sendFile()*.

Connect(): called by a client to build up connection between a node. It will send a node address to the input client address.

getFileList(): invoked by a client to get the file list in the system. The output is a list of file names.

addNewFile(Filename):invoke by a node which is added new file into. Record the new file's name in its file list, return the addressPortList of all nodes in the system.

deleteDeadNodeFromList: When Client or a Node detected a dead node in system, the directory server will delete it from the addressPortList.

## 2. Client

A client can connect to the storage system by obtaining location of storage node from the directory server. It can also request the list of files in system from directory server or the node. After successfully connected, it is able to add new files to the node and read existing files. When it detected the node it connected is dead, it will re-visit the directory server to connect to a new node.

### Functions:

connectToSystem(): send a connection request to the directory server with its address.

createNewFile(Filename): add a new file to the connected node. It will pass a file name to the node and let the new to generate the file.

readFile(Filename): send a request with the file name to the node, then print out file it received.

getFileListFromDirectoryServer(): send a request to the directory server for file list.

getFileListFromNode(): send a request to the node for the file list.

notifyServerNodeDead(): send a message to the directory server with the node information (address, port) when it find out that node is dead.

## 3. Storage node

Each storage node works as a data replica. After connected to a client, it will respond to the client's request. When a new file is added, it should notify the directory server to maintain the consistency.

### Functions:

intializeLocalFiles(): record the file names in its dataFolder.

createFile(Filename): create a new file locally with the file name.

addFileToSystem(Filename): after createFile(), send the file name to the directory server. After the server give back the node list, send this file name to all other nodes on list.

recieveFileToLocal(Filename): receive the file name from socket. Add the file name to the

file list.

sendFile(Filename, address, port): send the filename to a certain address and port which represent a node.

readFile(Filename): send a file to the client.

Register(addressPortList): the synchronize process at system initialization. Send all local files to all other nodes.

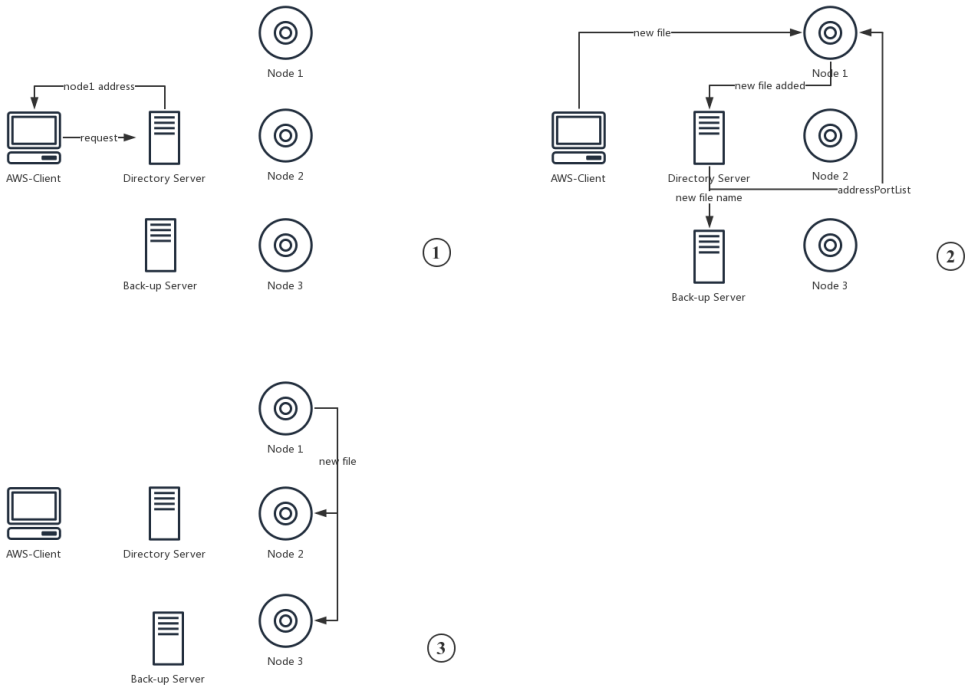
getFileList(): invoked by a client, return the file list in the node.

## Fault Tolerance Design:

Considering the potential faults which may happen during all parts of the communication process, we designed the system which is able to handle both directory server and storage node failure. The directory server is holding the address of all nodes, which is crucial information regarding file consistency. So we set up 2 directory server in the system: one main server and one back-up server. The communication is set up with default main server. If the main server is down, the sender won't receive any reply so that it will send the message to the back-up server again. After the broken server is repaired, the working server will send all information it holds to the repaired one to synchronize it.

A storage node could down at any time. Suppose the storage node which connected to the client downs while the client wants to add new file into it or to read file in it, the client will then recognize it is down and send a message to server with the node's information and also to request a new node. The directory server will then delete this node in its node list and assign a new node to client. Since our system is set up with 3 nodes, it can tolerant 2 concurrent storage node failure. Moreover. if we set up the system will `n` nodes, it will be able to tolerant `n - 1` concurrent storage node failure. If the node downs during sending the latest added file to other nodes, there might exist temporary difference between other nodes. When the broken node is repaired, it will again go through the registration process and all nodes will be consistent again.

For example, if `Client 1` is connected to `Node 1`. The client wants to add a new file to node however the node is down, it will not successfully send the message. Then `Client 1` will firstly send a message to the directory server to tell it `Node 1`'s address and port, and then require a new node `Node 2`. After `new file` is successfully added to `Node 2`, the directory will get a message from it and return all node's address and port. Then `Node 2` will send `new file` to all other nodes. The following figure is a flow chart of adding a new file to the system.




Pic 1. Flow Chart of Adding New File to The System