**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 one node to the client. When a new file is being added to a node, it will receive an adding request from that node and copy the file to other nodes. When a new node is added to the system, it will send the copy of all files in system to the new node.

Functions:

registerNode: 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(address): 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.

sendFile(Filename, file, nodeAddress): send a file to a node in system. It will call the node’s function *AddFileFromServer().*

newFile(Filename, file): invoked by a storage node to which a new file has been added by a client. The directory server should send the file copy to all other nodes in system through *sendFile().*

1. 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 a node. After successfully connected, it is able to add new files to the node and read existing files.

Functions:

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

addFileToNode(Filename, file): add a new file to the connected node. It will called node’s function *AddFileFromClient().*

1. 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:

AddFileFromClient(Filename, file): add a new file to the system by a client. It should store the file locally and send this file to the directory server.

AddFileFromServer(Filename, file): add a file from directory server. It should store the file locally.

readFile(Filename): return the file corresponding to the input file name.

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

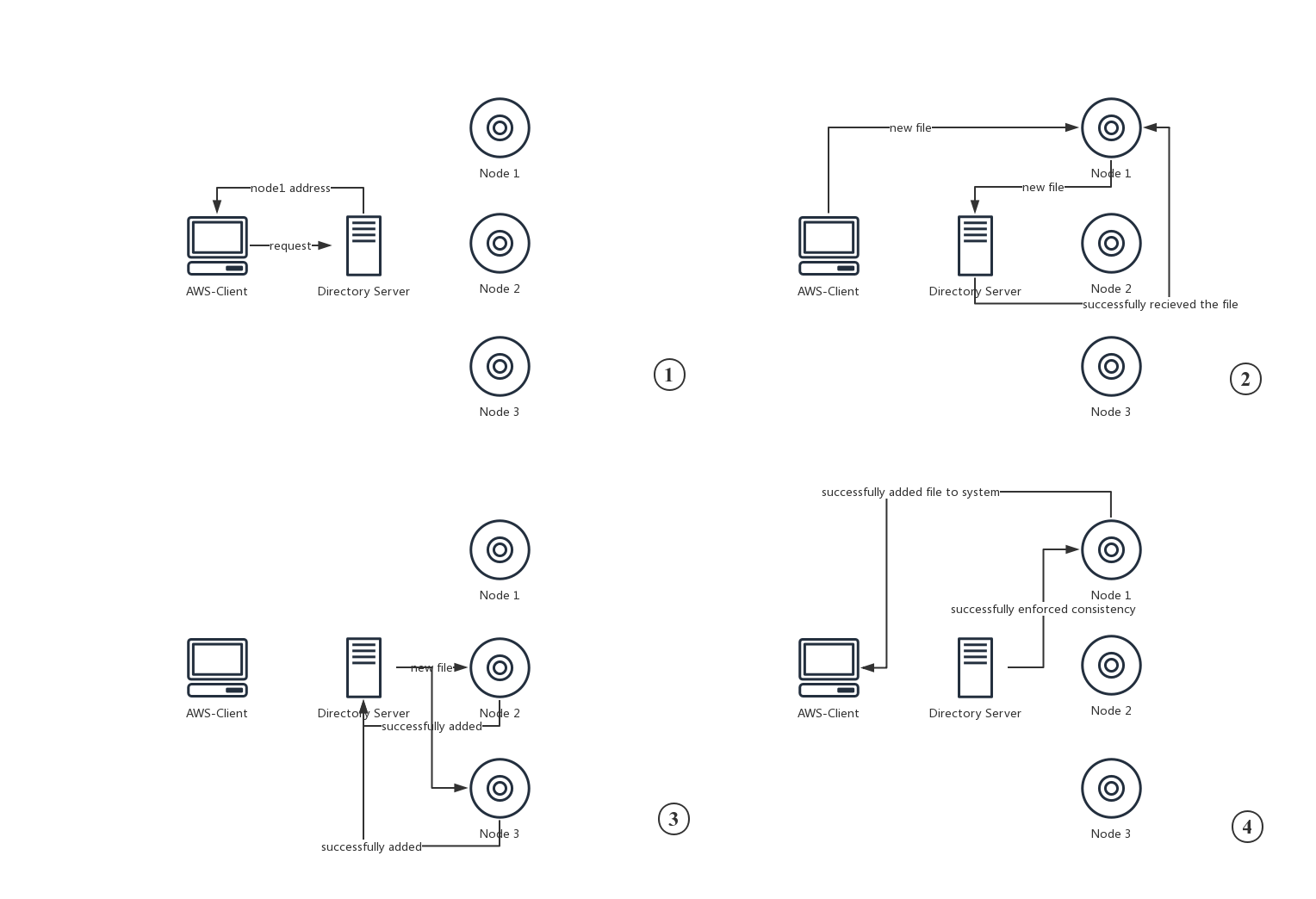
Fault Tolerance:

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. In case the nodes cannot directly communicate with each other, if the directory server is down, the consistency will no longer be maintained. So if the directory server if down, we will cut off all connections between clients and nodes then exit the program.

To detect if a node or the directory server is still working, we let the node react to the caller every time it finishes to process a request. So, if the connected client or the directory server doesn’t hear from the node, it will know the node is down.

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 receive the message from Node 1 which represents all operations has been finished. Then Client 1 will let the directory server to assign a new node to connect. After the file is successfully added to the new node, the directory will try to write the file to Node 1. Since the node is down, the server will not receive a message, then it will delete the address of node 1 in its memory.

Since our system is required to tolerant 2 simultaneous storage nodes failures, we will initialize the system with 3 nodes. Pic 1. is a flow chart of adding a new file to the system.



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

1. Trade-off and possible improvement

Following our design, a client can add a new file to the system with totally 11 messages exchanged. It is a large cost, and we can reduce it with introducing a roll-back mechanism:

if a node or the server fail while doing its job (which means a job is in process however has not been finished), the system will roll back to the original state with sending a message to the requester that the request is fail. In this case, the requester will resend this request to the node or the server to check whether it has failed.