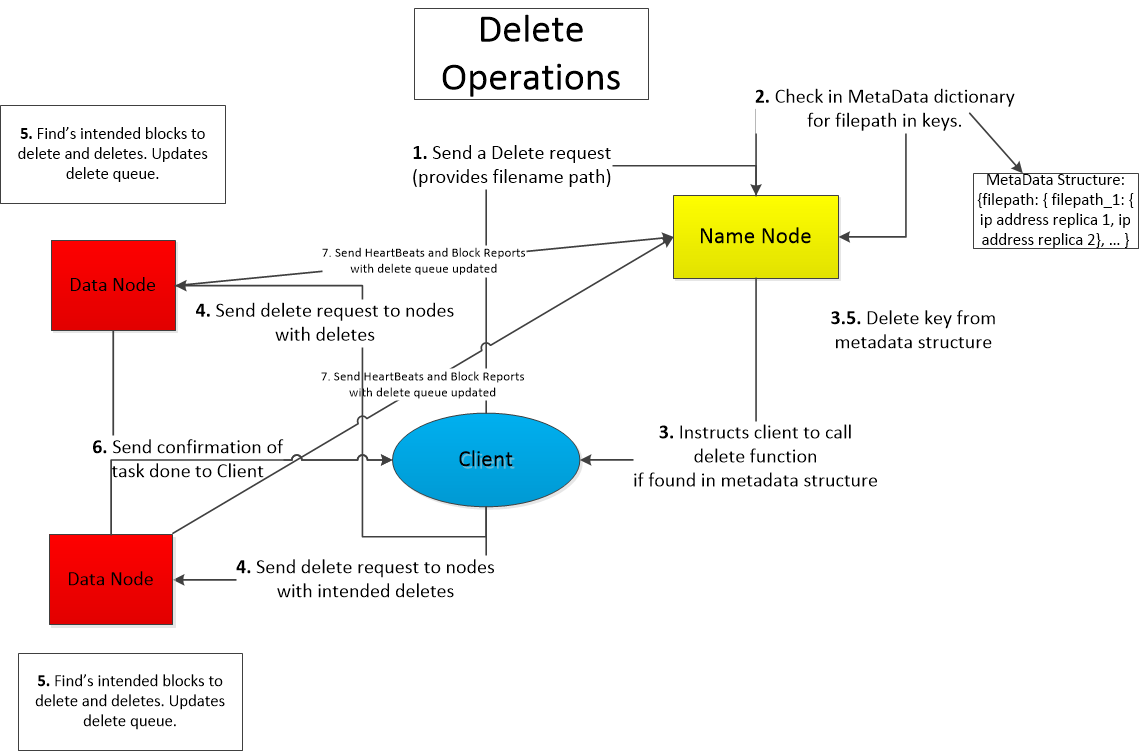


****

**Tymon Schroeder**

**Sunny Yeung**

**Hailey Nam**

**Glenn Wintrode**

**Assumptions:**

* The namenode distributes blocks by a round robin procedure.
* Recovery blocks from datanode failure are distributed randomly with caveats
* Can only delete directory when the directory is empty
* Ran with threads on all three components.

**Implementation details:**

**Replication factor:**  2

**Data nodes:** 4

**Blocks:**  32 MB

Our nodes communicate with each other through JSON messages using HTTP requests

A typical json would be

**JSONFILE:**

Status message/error:

Type of action i.e. delete\_file, create file,

Filepath

Block size

**Platforms used:**

EC2 for distribution and nodes

Flask as the framework for EC2

Python

S3 for the file paths

**Client**:

The client uses the flask framework and is used by the user as a website to send request to our SUFS namenodes and datanodes.

Fields:

Namenode Address

Contents of datanode

Folder of files downloaded from datanode

Functions:

**create\_directory(directoryPath):**

This function takes a user inputted directory path in the format of “/temp/” and sends a request to the namenode. It error checks for the end slash to make sure it is a valid path and corrects it upon sending it to namenode. It returns success upon directory creation or failure upon duplicate or other failure.

**create\_file(client path, source filepath):**

This function takes in a source filepath and a user specified file path to put into our hfds directory. It checks whether the filepath exists before creating. It sends filepath to namenode and receives block number and datanode information from namenode. It contacts individual datanode and sends blocked data to them.

Read functions:

**read\_file(FilePath):**

Client sends name node file path it wants to read.

This function takes in a filepath and contacts namenode to get datanode addresses for the file that the user wants to read. Once it has all the info, if it exists, it will contact the datanodes for the specified blocks and builds the data for read/access/downloading to client’s machine.

**list\_contents(DirectoryPath):**

This function will take in the path to a directory and contact namenode and get a list of all the files inside of that directory and send it back to the client. This list will be displayed on the GUI for client to see. It will return an error if the directory does not exist or namenode connection errors.

**list\_datanodes(filename):**

This function takes in a get request from the user to display the contents of each datanode, as in it shows the file blocks stored on the other data nodes. Displays it as a list of datenode IPs’ per file.

Delete functions:

**delete\_file(client path):**

This function takes in a filepath that the user wants to delete from our file directory. It will contact namenode and try to get the addresses for the data if it exists, once it gets the information of the datanodes, client sends a delete request to the specified data nodes.

**delete\_directory(dirPath):**

Client is given a directory path from the user, the client then sends a delete\_directory request to the namenode. Depending the response, it deletes the directory only if no files are present within the directory, else it restricts deleting until all files are gone from the current directory.

**NameNode:**

The NameNode

Fields:

Metadata Data Structure stores block names and block locations of the data

A dictionary of dictionaries of lists

The first key in the dictionary is the file. Second key is the part of the file. The list contains the ip addresses of replicas.

{FileA: { \_1: [ip address of data block 1, ip address of replica...], ... }, FileB:{}, …}

Block\_Size

Replica\_Num

RoundRobin\_counter

**Functions:**

**trigger\_all\_report():**

This function allows our namenode to start all datanodes’ while loops to periodically block report in a set interval. It does this by asking for a heart beat first.

**create\_file(client filePath, data size in bytes):**

This function creates a key in the metadata structure using the filepath that it got from the client, it checks if it exists before creating. It has a function that divides up the block into the size we specified. Chooses datanodes based on number of blocks based on a round robin algorithm. It requests a heartbeat from the intended datanodes to check for availability before sending the datanode information(addresses, and block amounts) back to the client.

**read\_file(client filepath):**

This function takes in the filepath from the client, it then checks the metadata structure for the key and see if it exists. If it exists the namenode returns back the ips of the datanodes for each block that has the data associated with the filepath.

**receive\_report(blockreport):**

This function receives blockreports from datanodes and updates the metadata structure based on contents of the block report. It also takes into the account of the deleted files to not overwrite the metadata structure.

**get\_directory(dirPath):**

Receives a directory path from the client node and returns a list of files in that directory to the client node

**create\_directory(DirectoryPath):**

This function takes in a directory path from the client and checks to see if it exists in our metadata structure, if it does not exist, then it inserts it into the metadata structure otherwise returns it already exists.

**delete\_directory(DirectoryPath):**

This function pattern match on all keys in metadata and return the files that have the prefix of the dirPath given. It pops the found entries from our metadata structure.

**list\_contents(DirectoryPath):**

This function accepts from the client request to see contents of a directory. It checks the metadata structure for matching directory and returns a list of content in that directory.

**list\_datanodes():**

This function takes in a client request to see datanode contents, it grabs the dictionary from our metadata structure and returns that.

**replica\_check():**

This function checks for datanode failure and upon datanode failure it removes the bad datanode from the addresses the datanode it has. It then looks in the metadata structure for the bad ip and remove it from each replica. It will then for each insufficient replica length, call datanodes to make additional replicas.

**DataNode:**

Has a contents directory which stores all the blocks

Fields:

Namenode address

List of datanodes and I.P. s

DeleteSet- this data structure is a set that contains all the files/blocks that the client wants to delete, this prevents blockreport from overwriting our namenode by seeing if items are in the set.

Functions:

**trigger\_report(namenode ip):**

This function sends a blockreport to the namenode every 10 seconds regarding its stored blocks on its content directory. It sends the delete set to notify namenode to not put back in the metadata structure.

**get\_block(filename):**

This function takes in a filename sent from the client and finds the specified block that the client intends to read. Returns the block to the client through scp command.

**delete\_block(filename):**

This function takes in a filename sent from the client and finds the specified block that the client intends to delete. It adds the deleted block into its delete set for sending in the block report. It then removes the file/block from its content directory and resets the delete queue back to neutral for future deletes.

**make\_replica(file, datanode ip):**

This function is called when a datanode first receives a block from the client, to which it then sends a copy to a random datanode IP that is not itself to make a replica. It will send a scp command to copy the block to another datanode.

**make\_failed\_replica(block, other DataNode):**

Namenode calls this function upon detecting a failed datanode. This function takes in the bad ip and removes it from its datanode ip list. It then sends a heartbeat to new recipient to see if that datanode has not failed. It then attempts to contact another datanode and scp its blocks to the new data node location.

**heartBeat():**

This function sends a response code ok to requester, the namenode, that it is alive and able to take request.