Distributed File System Design Document

Components of DFS

- Controller
- Storage node
- Client
- POSIX Client

Controller

Controller will be responsible for detecting if any node failure has happened. Controller will be checking the last heartbeat received timestamp for each storage node and if any node's heartbeat didn't come for more than 1 min , it will assume that node is failed and remove it from list of active storage nodes and also starts handling replica maintenance for the failed node.

Storage Node

Storage Node will be responsible for sending heartbeat messages to the controller every 5s to show that node is alive. Once write happens every storage node is responsible for replicating the data to 2 replicas for that storage node

Basic Client implementation

Client connects to the Controller using <Controller_hostname> <Client_name> parameters.

Client sends write and read requests giving the <source_filepath> and <destination_filepath>

Client gets a list of Storage Nodes available from the Controller, connects and writes to multiple Storage Nodes by breaking the file to chunks of 98Mb each in a pipelined fashion. If a Storage Node goes down while writing a chunk, it shows "Write failed,

please try again” error message and the Client has to re-execute the write command. A success message is shown on write completion.

Client reads from the Storage Nodes in parallel using multi-threading and builds the complete file from the chunks read.

POSIX Client implementation

For mounting the DFS on a directory at client we run the ClientFS with <Controller_hostname> <mount_directory> parameters

POSIX Client will create the mount_directory if it doesn’t exist.

To read the DFS mounted in the mount_directory, open a new terminal and ssh to the same Client Orion instance and go to the mount_directory and do *ls*

This will send a Read Directory request to the Controller, to read the file names from the root directory provided while running the Controller or other sub-directories inside the DFS. This is followed by Get Attributes request in that directory, which will get the metadata like the mode, size of the files in the directory. Then an Open request is sent to the Controller to open a directory.

When we try to run *vim* to open a file, the Read File request reads that file which is a chunk of the file stored in that directory of DFS.

The Controller will respond with the corresponding response types for each request type sent by POSIX Client. All the reads on the Controller are multi-threaded.

Design decisions

File Storage: On writing a file to DFS, we break the entire file into chunks of size 98Mb and store each chunk along with its meta file containing the checksum, chunk order, total number of chunks, file type etc in the storage nodes. We wanted the file chunk along with its meta file to be of size 100Mb hence decided to restrict the chunk size to be 98Mb.

Replica Maintenance: Once we have more than 3 Storage nodes, we have started creating two distinct replicas (randomly) for the storage nodes, and after that replica map will be updated in the Controller for each storage node and each SN will get information about their replicas.

Once we are writing chunks, after writing to the Primary storage node, the same chunks are getting stored at two replicas of that storage node. In replicas, chunks will be stored with the file type in metadata as “Secondary” and it will be stored in the parent_id dir of the Primary node.

Node Failure Scenarios:

- Secondary Node Failure: If the node which is failed is the secondary node of some primary nodes, then for that, we are creating a new replica for all those primary nodes and also updating those primary nodes about their newly added replica.

  After that, we are copying chunks in the newly created replica from the previous one existing(non failed) replica.

- Primary Node Failure: If the node which is failed is the primary node we are making one of the replicas (selecting randomly) of the failed node as the Primary node.

  After that, we are copying chunks in the newly created Primary node from the previous one existing replica and also storing chunks into the replicas of the newly created primary node.

Write: For writing a file to the DFS, the Client first sends a read request to the Controller to check if the filename already exists. If the filename exists (check in the Bloom filter), the Controller sends a list of probable Storage Nodes which have the file chunks. The Client then sends a delete request to these Storage Nodes. The Storage Nodes will

delete the primary chunks on their disk and secondary chunks from their corresponding replicas, and send a response back to Client.

Now the Client then sends a write request to the Controller. The Controller sends a list of nodes available to write the file chunks and the Client connects to the storage nodes one after another and writes the file chunk along with a meta file containing information about the chunk number, total number of chunks, file type(Primary/Secondary).

The write command that our DFS supports is write <sourcePath> <DFSDestPath> and we have designed the storage nodes such that it creates the DFSDestPath as folders in the storage directory and saves each chunks and its meta file inside it as 0 and 0_meta where 0 is the chunk number of the first chunk. Also the storage node sends the file chunks to its replica where they will be stored inside the directory named after the id of the parent storage node and the replica chunks meta file will have the file type as ‘Secondary’.

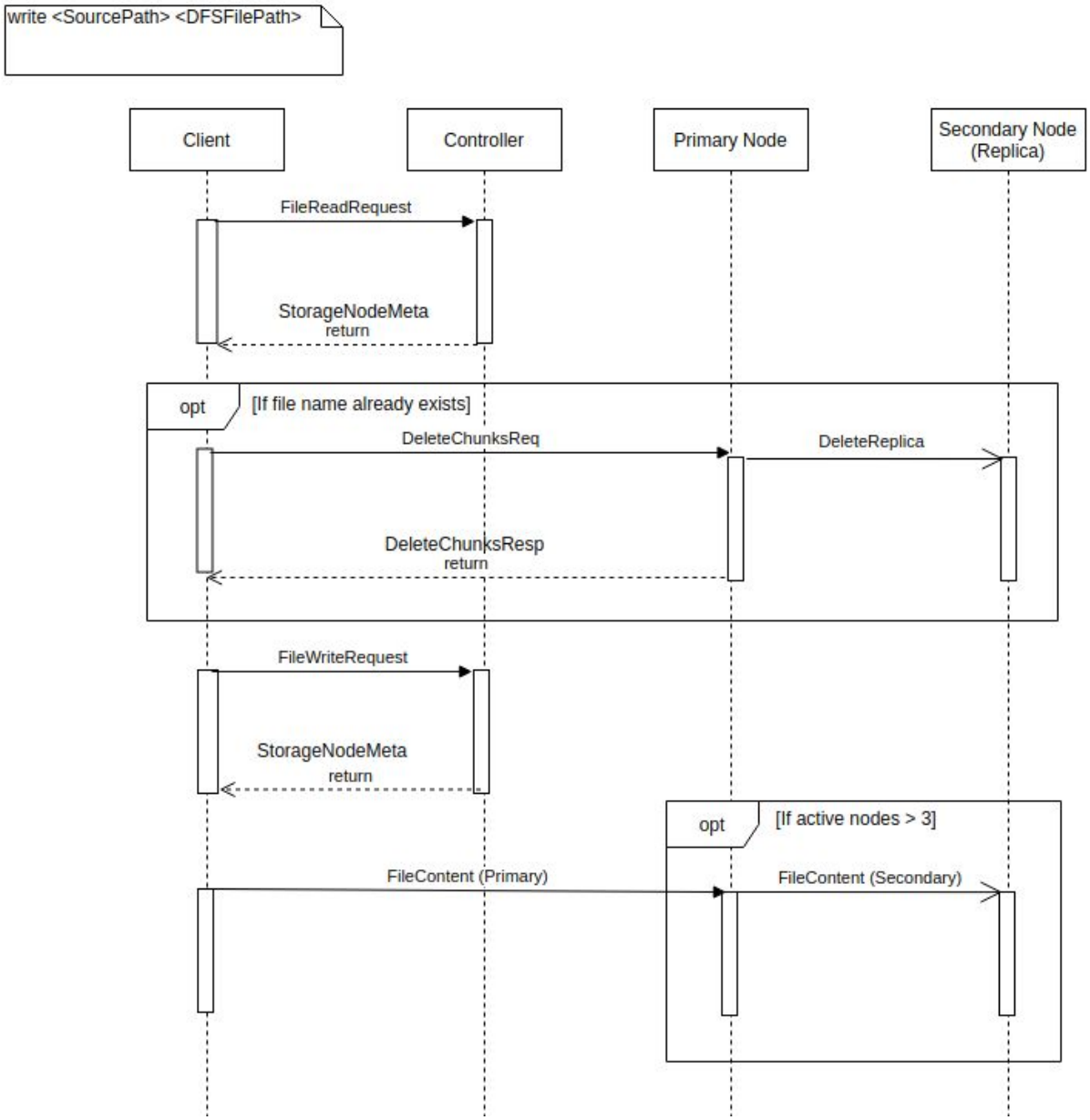

Fig 1: Write flow UML diagram

Read: For reading a file from the DFS, the client first sends a read request to the controller. The controller then sends a list of nodes whose Bloomfilter returns True and along with each node info it also sends the replica nodes. The client connects to each of the Storage nodes on a separate thread and reads the file chunk, in case the connection to a storage node fails the client tries to connect with one of the replica nodes.

The storage nodes send a ‘FileContent’ message back to the client in case the file exists else they send a ‘NoFileContent’ message to the client. At the client side, we maintain two concurrent maps: one with ‘DestFilePath’ as key and the value as Boolean array of size number of chunks and the other map with ‘DestFilePath’ as key and the value as Boolean arraylist. As we get the FileContent / NoFileContent message from the storage nodes that the client contacted, we update the maps respectively and check using a synchronized method if all the storage nodes returned NoFileContent message in which case we inform the client that the file does not exist or check if all the file chunks are returned and then we merge the file chunks which were stored in a temporary directory on the client into the entire file and delete off the temporary directory.

While reading a file chunk from a storage node, we also verify that the checksum of the file chunk is the same as that of the checksum stored in the meta file of the chunk before sending the file content to the client. In case the verification fails, we then connect to one of the replica nodes and fetch the file content, rewrite the file chunk with the replica file content and send the file content to the client.

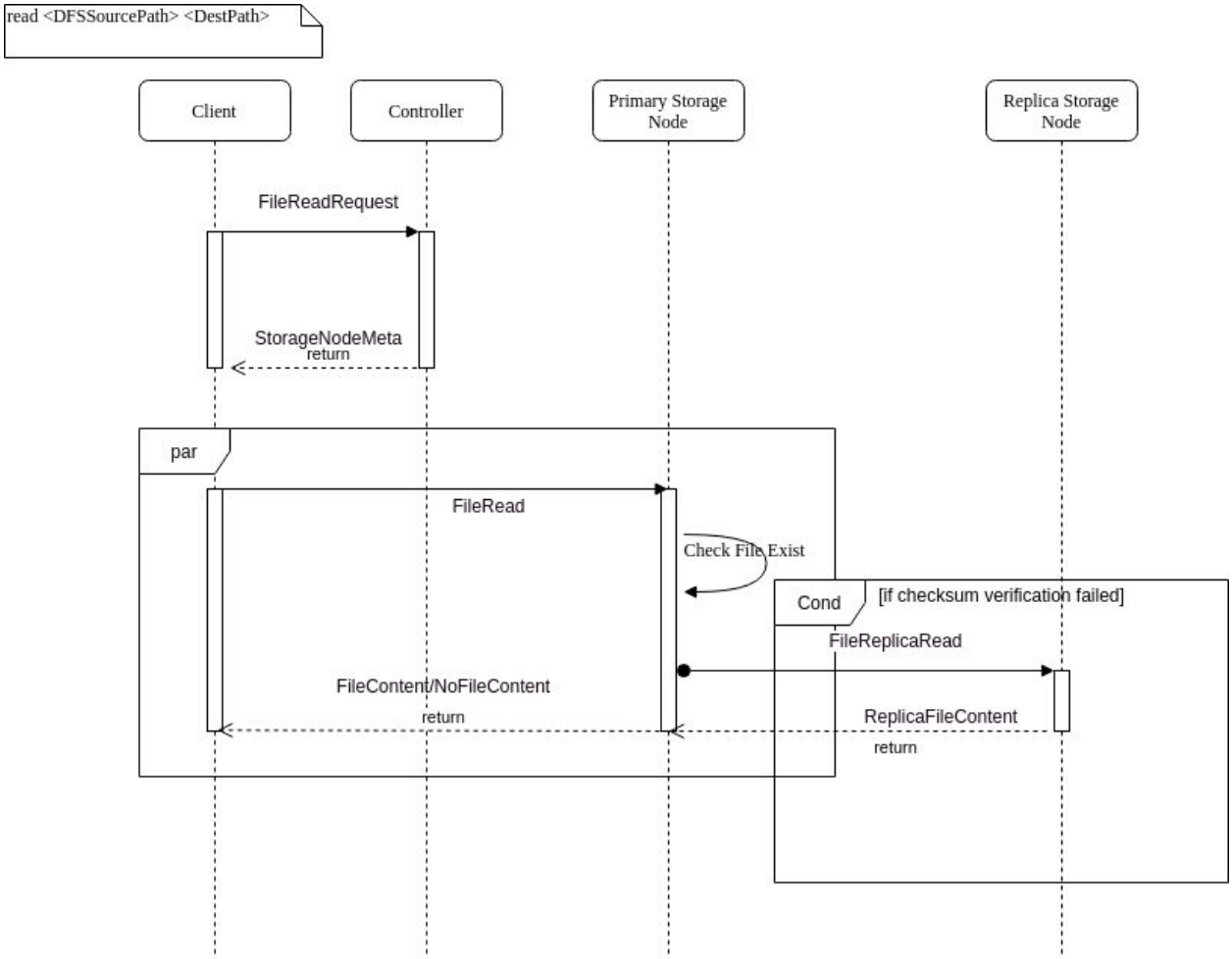

Fig 2: Read flow UML diagram

Messages

DFS components use the following messages for communication in various scenarios

- ➢ SNRegistration - Storage Node to Controller on the start of a storage node, which updates the maps in the controller with the SN info like port number, SN Id.
- ➢ FileContent - Client to StorageNode on file write contains the file content of the chunk and the meta information like total number of chunks, chunk number etc. Also used to communicate from Storage Node to Client on file read.
- ➢ FileWriteRequest - Client to Controller to get a list of Storage Nodes on a file write command.

- StorageNodeMeta - Controller to Client containing the list of Storage Nodes to connect on a file write/ read request. In case of a read request, the replica nodes information is also sent for each node in the list.
- FileReadRequest - Client to Controller to get a list of Storage Nodes that might contain the file on a read request.
- Error - Any error message to Client either from Controller(in case of no file exists on read) or from Storage Node(in case of checksum error and no replica exists to rectify it) is sent through this.
- FileRead - Client to Storage Node on file read.
- FileWriteAck - Storage Node to Controller on successful write of a file in order to update the Bloom Filter of the storage node and the file system tree of the DFS.
- FileReplicaRead - Primary Storage node to Replica node for reading a replica file in case of checksum verification failure in the primary node.
- ReplicaFileContent - Replica node to Primary Node with the file content of the replica file requested in the case of checksum verification failure in the primary node for the file chunk.
- NoFileContent - Storage Node to Client in case the file does not exist on file read.
- RequestInfo - Client to Controller for requesting the list of active nodes in the DFS or for listing the file system tree. If infoType is ‘info’ the list of active nodes in the DFS is requested and if the infoType is ‘ls’ the file system tree is requested.
- ActiveNodesInfo - Controller to Client with the active nodes information.
- FileTreeInfo - Controller to Client with the file system tree information.
- ReaddirRequest - POSIX Client to Controller with directory path.
- ReaddirResponse - Controller to POSIX Client with the contents inside the directory and status.
- GetattrRequest - POSIX Client to Controller with directory/file path.
- GetattrResponse - Controller to POSIX Client with mode, size and status.
- OpenRequest - POSIX Client to Controller with directory/file path.
- OpenResponse - Controller to POSIX Client with status information.
- ReadRequest - POSIX Client to Controller with file path, size and offset.

- ReadResponse - Controller to POSIX Client with status and contents.
- DeleteChunksReq - Client to Storage Node to delete the existing chunks of a file during file rewrite.
- DeleteChunksResp - Storage Node to Client acknowledging the chunks are deleted during file rewrite.
- Heartbeat - Storage Node to Controller with metadata like free space available on the Storage Node and number of requests processed by that Storage Node.
- FileReplicaLocation - Controller to Storage Node for sending replica informations to respective storage nodes
- CopyReplica - Controller to Storage node for copying the files in case of node failure in the newly created storage node
- DeleteReplica - Controller to Storage node for deleting the folders from storage node in case of Primary Node failure