Big Data - UE19CS301
Project Report

Date: 06/12/2021

Project title chosen : Yet Another Hadoop

Shamanth KM	PES1UG19CS444
Sharath Hegde	PES1UG19CS451
Vibhav Shiras	PES1UG19CS402
Shreyas Vinayaka Basri KS	PES1UG19CS469

Design details

Datanode design

- Each datanode is a directory with maximum capacity of datanode_size.
- Every file is distributed across various directories according to the hashing function which is file_block_number % number_of_datanodes.

Namenode design:

- Datanodes are tracked by namenodes through two json files.
- primary.json keeps track of every file and folders that is present in the distributed file system with key as absolute path to the file and its value as datanode mapping.
- dnode_tracker.json keeps track of the number of blocks left in each datanode which helps in replication and updating log files.

Secondary namenode design:

- secondary.json acts as a backup for primary.json(primary namenode) and the content of this file is copied to primary.json in case of namenode failure.
- Copying is done by heart.py file.

Hadoop map reduce design:

- Word count operation is performed to check the correctness of the design.
- The file for which word count has to be performed is being catted and the contents are read.
- Since the absolute path to mapper and reducer are given as command line arguments using os.system() both the functions are executed and the output of which is written to a txt file.

Command line design:

- Implemented through a simple while loop in which until alive variable is true the hadoop interface is active and once exit command has been given the interface terminates.

Surface level Implementation details:

Datanode

- All the datanode functionalities are written in main.py and dnode.py
- Datanode directories are created in main.py and also the log files are being created indicating the data nodes have been formed.
- Two hashing functions, one for splitting of files and another for their replication have been written to handle all the edge cases related to them.
- If there is no space available to replicate the block ,corresponding error messages are displayed.
- If the replication of the block is also hashed to the same data node then the replication does not happen.
- Any operation related to particular data nodes are being written to that datanode log file.

Namenode

- Contains three json files primary.json , secondary.json and dnode_tracker.json.
- primary.json acts as a primary data node in which keys are absolute file or folder paths and values are its distribution to data nodes and empty respectively.
- dnode_tracker.json keeps track of data node blocks left and also which blocks are occupied.
- Inside dnode_tracker.json the keys are datanode numbers and value is a dictionary of size = datanode_size.
- Based on the count of data blocks that are left inside the datanodes we decide whether to replicate a block or not.

Secondary name node

- Two functions called checker and sync have been implemented.
- Checker keeps on updating the checkpoints and checks for the namenode failure every checkpoint time.
- Incase of namenode failure during this checkpoint period, contents of secondary.json are dumped to new primary.json using the checker function.
- When the checkpoint period is greater than or equal to the sync period the contents of the primary.json are dumped to secondary.json and hence are synced.

Map reduce

- Word count mapper and reducer code are executed on a text file.
- Contents of the text file are obtained by the cat function and then is read into a text file on which mapper is performed.
- Output of the mapper is recorded to a text file which is the input to the reducer.

cat

- Every data node which has the file blocks details except the ones which contain its replication are read and displayed on the command line.
- If the path is invalid then an error message is displayed.

mkdir

- Folder is created as in the absolute path of the folder is taken as key and value as empty is dumped into primary.json.
- If the path is invalid then an error message is displayed.

ls

- All the files and folders present under the given folder are displayed.
- Logic used is to compare the fs_path+folder_path with the keys present in the primary.json and if present display the files and folders.

rm

- The datanode block which contains the file has to be removed and the corresponding changes to the primary.json has to be made.
- File is removed using os.remove().

rmdir

- Pop the keys of the folders present in the primary.json.
- For files present inside the folder call rm() function.

put

- Splits of the file are made and distributed to different data nodes based on a hashing function.
- If space is available for replication then the blocks are replicated.

Reason behind design decisions

- We have read the file as a byte array since it was easy to handle the file splits.
- Json files for namenode implementation are mainly because the key,value pairs are easy to modify and changes also can be easily dumped.
- Subprocess module is used in secondary namenode implementation to run heart.py in parallel.
- Two hashing functions because there were different edge cases for initial file split and its replication.

Takeaways from the project

- Explored a lot of distributed file system functionalities.
- Understood the features of datanodes and namenodes in depth.
- Working of hadoop distributed file system.
- The way in which files are replicated to provide backup in case of failure while still maintaining data consistency is a great learning outcome.