UECS19322: Big Data – Project

Yet Another Hadoop

Team_ID: BD_083_086_242_306

Team Members:

Archit Sagar PES1UG19CS083

Arpan Shetty Y PES1UG19CS086

Likhith PES1UG19CS242

Nithesh A PES1UG19CS306

Design Details:

This is a replica of HDFS using python, on the local machine. Each Namenode and Datanodes are implemented as processes that communicate with each other using RPyC module, which in turn runs on a TCP connection.

A configuration file is provided into the setup.py program, which sets up the DFS. Then, using the generated config file, DFS can be started using the start.py program. Once it gets started, Namenode and all the Datanode start running. Then client.py program can be used to interface with the DFS.

A map-reduce job can also be run on the DFS using the mapred.py file. The logs are generated from the processes and are stored in the specified files. Fault tolerance has been implemented to deal with possible failures.

Surface-level implementation:

- setup.py
  - Here, a user configuration is taken, validated and DFS is setup with the provided configuration.
  - It initialises the Namenode, Datanode and Log-files folders.
  - It generates hash for the given configuration, to identify and verify while loading it later.

- start.py
  - It reads the DFS config file and verifies the hash, to make sure configuration isn't modified after setup.
  - It starts Namenode and all Datanodes as subprocesses on an available ports, and ensures they're running properly.

    - It provides debug mode -d, which lets us manually start and stop different Datanodes (for demonstration purpose).

- namenode.py
    - It reads the config file and sets itself up accordingly. Ex: Replication factor, number of Datanodes to manage, Block size etc.
    - It maintains fs-image, which has all the necessary metadata of the DFS.
    - It maintains Datanode details which includes ports, block ids and space available.
    - It frequently communicates (sends heartbeat) with Datanodes after each interval (sync_period) to ensure they are online and have required blocks.
    - It frequently backs up fs_image into checkpoint file so it can be loaded on restart or failure (secondary namenode).
    - It has all the necessary helper functions used by client and Datanodes.
    - It allocates blocks for new files and removes blocks of existing files on demand.

- datanode.py
    - It registers itself with Namenode using the Namenode address (port).
    - It provides interface to write and read individual blocks from Datanode.
    - It handles write operation recursively for each block, that is after write is successful on the current Datanode, it forwards it to next Datanode.
    - Based on the heartbeat received from Namenode, it keeps its blocks up-to date. That is, it removes extra blocks and recovers missing blocks.

- client.py
    - It runs on terminal to provide user-interface to a given DFS via various commands.
    - Commands are: mkdir, cd, put, cat, ls, rm, rmdir, exit
    - It communicates with both the Namenode and Datanodes while running the above commands.
    - It handles reading data from Datanodes even if some Datanodes are unavailable.
    - It shows appropriate error and success message for user to understand.

- mapred.py
    - It is a program that performs map-reduce job on the given file in DFS.
    - It takes in necessary arguments via command line (--input, --output, --config, --mapper, --reducer) and if everything's proper, starts map and reduce tasks.
    - File is read from DFS and is stored temporarily on local disk, then provided as input to mapper program.

- Intermediate output from mapper is stored temporarily, sorted and given as input to reducer program and finally reducer output is stored in destination.

fs_image structure:

file table:

```
{
    folders: {
        foldername:{
            folders: {...},
            files: {...}
            metadata:{...}
        }
    },
    files: {
        filename1: {
                metadata: {...},
                blocks: [][]
                (each row: [block_id, datanode1_id, datanode2_id, datanode3_id])
            }
    }
}
```

datanode_blocks: {datanodeID: [list of blocks], … }

Fs_image = (file_table, datanode_blocks)

Reasons behind Design Decisions:

- RPyC module is used because, it provides easier functionalities to communicate among different processes using TCP which even the original HDFS uses.
- Hash file is generated and stored on DFS setup, to validate the config files in future.
- Subprocesses are used to start Namenode and Datanodes so that they run in parallel and easy to manage (start and stop).
- Pickle module is used to serialize and deserialize the object structure for saving and loading whenever its necessary (fs_image).
- Fs_image is designed that way, so we can traverse it easily and easy to understand.

Take away from project:

- Understanding the underlying architecture of Hadoop DFS.
- Learning the communication process between Namenode and Datanodes.
- Learning how DFS implements fault-tolerance.
- Understanding the map-reduce framework, how inputs and intermediate data are handled.
- Understanding Git and GitHub.