YAH - Yet Another Hadoop

Team_ID BD_321_491_582_592

Palak Kothari	PES1UG19CS321
Snigdha Sinha	PES1UG19CS491
Vridhi Goyal	PES1UG19CS582
Yashi Chawla	PES1UG19CS592

Design details

Data nodes

Datanodes have been implemented as folders, and blocks have been implemented as files. The initial setup involved making a specified number of data nodes as folders. There is a log text file for each data node.

Name nodes

Namenode consists of primary name node, secondary name node, checkpoints, and name node log. The primary name node consists of four files block information, data node information, filesystem, and file information.
Block information consists of the mapping of blocks to data nodes, essentially the location where the data nodes are stored.
Datanode information consists of data node to block mapping, essentially a list of all blocks available in the data node.
The **filesystem** is the HDFS file system information.
File information consists of file details such as file id, file path, number of blocks, file size.

The design for the implementation of a heartbeat signal is that a timeloop is used to periodically implement a few functions after sync_period time.

YAH commands

A CLI was implemented to execute various commands and their corresponding paths used for the same. Commands implemented include - PUT, CAT, RMDIR, RM, MKDIR, LS, RUN, FORMAT. Commands were taken in from the terminal.
Running Hadoop jobs- **RUN** allows you to perform a Hadoop job, given the input file path, path to mapper and reducer, and the path to an output directory.

Surface level implementation

Pathlib library was used to delete files and folders and make new files and folders as well as to copy files from one place to the other.
Timeloop was used to implement certain functions periodically to implement sync.
Filesplit was used to implement the splitting of files in certain sizes.
Processes were used to implement multiple sub-process for mappers for each block.

Setting up the YAH system involves reading all the config variables from the file provided and initializing the data node and name node directories with the required content. Variable such as paths to the data node, name node, logging files, sync period, block size, data node size, etc. is obtained here.

The initial setup for data nodes involves making n number of data node directories based on the given number of data nodes. As well as a log text file for each data node in the data node log directory.

The initial setup for the name node involves the creation of 3 directories- primary name node, secondary name node, checkpoints, and a text file that contains the name node logging. The primary name node consists of four files filesystem, file info, data node info, and block info, all these files are created and initialized to empty at the time of setup.

PUT - takes the source and destination file paths as inputs. It verifies the existence of both paths. Then it breaks the file into n number of blocks depending on the block size and file size using file split. Each file and each block has been assigned a unique ID. Then all blocks are replicated the given factor times. Each block is mapped to one of the available data nodes. The choice of the data node can be either made randomly, or using the least occupied method, or based on hashing. The file is not added to the HDFS, and file details are updated in all four files in the primary name node.

CAT- used to display the contents of a file. Firstly it verifies if the file exists. The command extracts file id, file path, number of blocks, and using the information obtained, it extracts the blocks from data nodes.

LS- used to view the contents of a folder. Also lets you view the contents recursively using -r.

RMDIR- used to delete a folder. Check if the folder exists, and then deletes the folder and its contents from the filesystem as well as remove all of the files in the subdirectories from the system.

RM- used to delete a file. Checks if the file exists, if it does, removes the blocks corresponding to the file from the system and updates the filesystem.

MKDIR- used to make a new folder. Checks the given path, if it doesn't exist, creates a new folder in the filesystem.

RUN- is used to run a map-reduce job. The commands take in four arguments, -i path to the input file, -o path to the output directory, -m path to the mapper file, -r path to the reducer file. It checks if all the paths exist. Using the details of the input file path extracts all required information from the file info, and then gets the path to all blocks associated with the file. We implement a subprocess to execute the mapper given the block. The outputs from all mappers are then aggregated and sorted and then streamed to the reducer. In the output directory, two files are created, one for the output from the reducer and one for the status of the job.

FORMAT- used to format the name node and data node. Delete all the contents of the data node and name nodes, and is used to create and initialize the structure again. When the system HDFS is loaded for the first time, it prompts the user to format.

Namenode logging- logs in the command executed on the CLI to the name node log.

Datanode logging- implements logging for each data node, and logs when and which block was accessed and also the command that resulted in the access.

Checks that are performed every sync_period include

1. **Check and revive primary name node**- in case the primary name node is not found, all recent files from secondary name node as copied to primary. Also checks if all the required metadata exists in the primary name node
2. **Namenode logging**- log in the commands executed on CLI
3. **Update data node and block information**
4. **Update secondary name node**- copies all the existing files from the primary name node to the secondary.
5. **Create a checkpoint**- creates a folder with the files from the primary name node at that point in time.

The reason behind design decisions

Datanodes have been implemented as folders and blocks as files. Each block has been assigned a unique ID, which helps in **easy access to the block** for put/cat commands, and the same has been used for logging access to the blocks, in data node logging.

Subprocesses have been implemented to perform the Hadoop job. A subprocess is spawned for each block fed to the mapper, this ensures we have multiple mappers running, similar to the Hadoop system.

The heartbeat feature of Hadoop has been implemented in the form of a few functions being called periodically using timeloop. And the checks include checking the primary name node and reviving it. This was implemented in order to make the implementation **fault-tolerant**.

Block information file in the primary name node is used to map blocks to data node and data node mapping to blocks is present in the data node information file, these files help in easy access and retrieval of the blocks.

Filesystem information is a nested dictionary structure of the filesystem, which provides us with the current filesystem.

File information stores information related to the file such as the unique file id assigned to it, file path, number of blocks, and the file size. This information allows us to retrieve the information related to the file required to perform other functionalities on the YAH system.

There are update functions for the block information, data node information which updates the information memory to disk and vice versa, this was implemented to **ensure consistency** in the local files as well as the files being used at runtime.

Takeaway

Throughout the project, we gained more information on the working of the Hadoop system and in order to implement it on our own, we learned how to use various new python libraries and modules.

We learned more about how Hadoop is made to be fault-tolerant due to the heartbeat feature, about how blocks are replicated over different data nodes to take care of redundancy and make the system fault-tolerant.