Big Data
Project Title: Yet Another Hadoop

Design Details:

To mimic Hadoop Distributed File System we designed a command line interface which can be implemented by running the python file cli.py, A call for our hdfs clone is made once the cli.py is running.

Regarding file distribution:

Each file is broken into number of blocks based on the config.json file and is stored in the datanotes whose path is again mentioned in the config.json file. The path of the blocks present in datanodes are written to the namenode.txt file along with the offset of each block

The dfs program reads the config.json file and creates directories with the file names if they do not exist and we used text file to mimic datanodes and namenodes.
Writing to these nodes is done through a process but is sequential, the mapper is sent to each datanode asynchronously using threads. The details about where a specific block is stored in the datanode is understood by reading the namenode.

Each block for a file is recognized by the file name before the block path in the data node

Surface Level Implementation about each unit:

DFS unit: This unit reads each and configures the dfs paths for namenode and datanode, the file gets broken down into blocks and is stored in datanodes and the path of these blocks is stored in the namenode. the namenode and datanode are .txt files. The replication of these blocks as an safety measure is also taken care in this block

Asynchronous part: I have used multiprocessing concept in oder to mimic the map reduce part of Hadoop job. Here I have first created a pool of workers in a class and then I initialise it when I create an object of the class. The class receives 2 inputs that are the mapper and reducer then it return an object. The object is used in combination with the map() function which passes every line in a list to the object.The lines are directly accessed from the data node blocks with respect to their offsets and are stored in a list. All these lines are run Asynchronously in order to mimic map reduce and the output is given to a function within the MapReduce class in order to sort and send the values to the reducer. Where the reducer also runs asynchronously. All these together mimic the Mapreduce part of the Hadoop job

CLI unit:We have a implemented a interactive command line interface(CLI) to help with the commands of both Single-node DFS(Distributed File System) and basics OS commands. For DFS part u can use mkdir, rm/rmdir, put, cat and ls commands which are very much required to run a Hadoop job. The above mentioned is accompanied by the asynchronous part to make the process faster. We have used many python modules like threading, prompt_toolkit, subprocess, multiprocessing, importlib to execute the command a efficient way. We also have a dedicated command call YAH to run a Hadoop job which takes in input file(which is to be already put in the DFS) ,output , mapper, reducer and a config file which contains the specifications of our DFS.

Reason behind design decision:

We, as a team felt that the best approach to complete project would be divide and conquer. So we divided to split the work based on the specifications of the project and firstly work on a working solution, and then to of course optimizing the solution. We tried many ways of designing this Distributed file system and decided that the above mentioned implementation is the most appropriate one for this project.

Takeaway from the project:

- We have a better understanding about how HDFS works
- We are more familiar with CLI commands.
- Usage of daemon threads
- Inter process communication
- It was fun to divide the work between teammates, coordinate and integrate our work and ideas.