Pradeep Reddy Rokkam, Assignment 2

NOTE: When code is being run, it is working sometime and sometimes it is throwing an error. I am not sure why the exact is working sometime and why it not working sometime. So, when you are testing the code, could you please run the code multiple times so that you get the desired result. Also, what I observed is that my word count is running correctly in every two to three runs. However, my inverted index is taking a greater number of runs than expected.

Design, Data Partitioning and Communication between nodes:

I tried to include all the details in a step-by-step process of execution

Master Node:

- All the inputs are given through a config file. I have used json as the config file
- Master node will be spawned first
  - Initially it will do two things
    - This will initiate the mappers based on the mapper count provided by the user
    - It will also create new threads to handle the connections concurrently by the mappers
  - Now mappers will connect master and request the data and our master will provide the data based and the input size
  - ***Data Partitioning of Input file:***
    - *The way how I performed data is very simple,*
    - *I read the number of lines in the input file and send dividing the number of lines equally between all the mappers*
    - *Assumption: I assume all the lines will have similar number of words.*

- **Mappers:**
  - Based on the no of mappers that re provided in the config file our master node will spawn that many number of the mappers nodes as the OS processes
  - Now each mapper will establish a connection with master and sends a signal that mapper has started
  - Based on the input function master will send the data to the mapper for mapping process
  - Mapper will receive the data and starts the mapping process and send the data to the to the Key value store
  - Once mapping is done then mapper will send the signal to master saying that mapping is done for that mapper and will send only the keys to the master (Because master need to assign work to reducer, and that is the reason why mapper will send keys to the master)
  - **End of mapping**

- **Master Node and Barrier Implementation and Data Partitioning of keys:**
  - Master node will constantly monitor the mappers and will also get the keys to from the mappers

- Master will keep a track of how many mappers and completed and will maintain a global variable that we can access from all the threads
- Once we receive acknowledge from all the mappers then only, we move to the next step of the process: **Barrier Implementation**
- The next step of the process is starting the reduce process. Here we have two steps again.
  - This will initiate the mappers based on the reducer count provided by the user
  - It will also create new threads to handle the connections concurrently by the mappers
- ***Data Partitioning of keys:***
  - *As, I mentioned at each mapper process, each mapper will send the keys that it worked (only unique keys) will be sent back to master*
  - *I am accumulating the keys to the in a set, so that I know what are the unique keys that we worked on*
  - *Since I am capturing **the keys in set fashion**, this essentially mean that **data is shuffled** as set does not have any order (Because set does not have any data order)*
  -
  - *Now I have to ways to send the data to reducer once the reduction process started*
    - *Since I know the keys list, just like I how I divided the input data based on the line count, Here I split the keys based on the key count*
      - *This way each I will make sure I am dividing the work in equal amount to all the reducer workers*
    - *I can also use simple hashing in combination with modulo operator, however, I did not see a lot of significance here with usage of hashing*

- **Reducers:**
  - Based on the no of reducers that are provided in the config file our master node will spawn that many number of the reducers nodes as the OS processes
  - Now each reducer will establish a connection with master and sends a signal that reducer has started
  - Based on the input function master will send the keys to the reducer for reducer process
  - Reducer will receive the keys that it needs to fetch from the **key value store**
    - All the data that is obtained by reducer is through remote calls to the **key Value stores**
  - Using the keys, the reducer will fetch the data for each key
  - Now reducer will aggregate the value for each of these and send the aggregated data to the key value store again
  - Once reducing is done then reducer will send the signal to master saying that reducing is done for that reducer

- **Master Node and Barrier Implementation:**
  - Master node will constantly monitor the reducers and will keep a note of how many reducers are completed and how many or remaining
  - Once all the reducers are completed master will generate the output file that will be stored in the key value store

  - Thus, Entire Map Reduce process is competed

- **Data Storage:**
  - Assumption is that Master can access all the data from the input location and can also load into in the memory
  - However, all the intermediate files and Final output fields are stored in a key value store

- **Key-Value Store:**
  - **File format: JSON**
  - Key - Value store will always run and a given a address and port
  - This implements multi-threading, so that it can handle concurrent connections from the workers i.e. from mappers and reducers
  - Our mappers will store the data in the key value store and reducer also use the key value store.
    - And our key value values are stored in JSON format, so we may get in to th problem of overwriting keys when to mappers have the same keys
    - To avoid this, our key value store is modified in such a way that it will append mapper and reducer key word to the keys along with thread ids, So that we can identify key is from mapper or reducer and will also know from which mapper or reducer it came from.

Configuration file:

- All the details to the MapReduce function should be passed as a configuration file.
- The format used here is **JSON**
- Configuration looks like the following
-
```
{"function": "invertedindex", "inputlocations":["D2.txt", "D3.txt", "D4.txt","D5.txt"],"no_of_mappers":5, "no_of_reducers":5,"master_adrress":"127.0.0.1","master_port":9800,"kv_address":"127.0.0.1","kv_port":10000}
```

- The details that are passed are the

  - FunctionName
  - Input Locations
  - No_of_mappers
  - No_of_Reducers
  - Master_Address
  - Master_Port
  - Key Value Store Address

- Key Value Store Port

Functions implemented:

- The following functions are implemented
    - Currently the functions are baked in the code,
    - However, use has the flexibility to choose between the wordcount and inverted index by mentioning it in the configuration files
- Word Count
- Inverted Index
    - I was able to implement the both the functions, however, I saw that some of the times some of the keys are not being captured by the program, I mean they are lost either during the map process or reduce process. However, when I clear the output and run the code again, like few times (5-6 times), I was able to capture all the keys. To compare this with the **Google's paper,** it was very evident that failure is very common in distributed setting and that's why we need to implement some fault tolerance techniques, like running back up tasks

RUNNING the code:

- Make sure you change the configuration options before you run the code
- Once you made changes, please run the key value store code (KVMultithreading.py)
    - **Key Value store should always be in the running state, and this is very important**
    - **Just run the key value store, no inputs are required**
- Once you run the key value store run, the code MapReduce.Py

Comparison with Google's Paper:

- My implementation is almost similar to Google's Map Reduce paper. Of course Googles will be much better, at least the core idea is similar
- Googles Map Reduce is splitting the data into chunks of 16 MB to 64 MB, I am splitting the data into equal parts based on the number of mappers
    - I could have improved this in my code
- Google's Map Reduce assigns M mappers and R workers, My program is performing the similar operation
- In Google Map Reduce periodically the buffered pairs are written to local disk and then they are sent to master who is responsible to forwarding these locations reduce workers. And when the reducer is informed by Master about the reduction process, Reducer will use RPC to get the data from these locations.
    - I am also doing similar operation where Reducer are using RPC to get the data from the key Value Store.
        - Because here we are storing all the intermediate results key value store

        - Instead of locations being sent to the reducer we are send the keys(Remember this is not the actual data that is the output of mapping process, It is just the key information, like the location information)
- In Google's MapReduce after the reduction process is completed the output is stored in the R output files one in each of the reducer and then the map reduce does something to aggregate these files.
    - I have also done a similar design where the reducer will send the data to the key value store and store the data directly in the key value store
        - This data will be stored along with the un – aggregated key value pairs, However, to distinct the mappers and reducers out key s will be appended with mapper and reducer key word to identify which keys are aggregated or not
    - As I mentioned earlier, our key values store will identify the keys of mapper and reducers. Even in the keys of mappers we will be able to identify from which mapper it came from or from which reducer it came from
    - Also, just like how Google does some process to generate the output file,
        - We leverage the barrier here, In our design once we know that we all the reducer are completed then we send a signal to our key value store asking to aggregate the results of WordCount or Inverted Index based on the result
- One of the main differences between thee Googles implementation ad my simple design is Fault Tolearance
    - It very surprising that the same code when ran once throws error (mainly due to the network issues or connection resets) and when ran at other time works perfectly fine. As discussed in the paper, Google implements some back up tasks to perform fault tolerance.
        - However, in my case I assume that all the workers will work, obviously this is not a great assumption as I saw that my code faced issues due to some errors in the network
        - This will be the improvement that I will work on when I move my code to the Distributed virtual machines
- Google solved the problem of network issues very cleverly by implementing the Locality (Section 3.4 of the paper). Where workers are directly assigned to the machines where the data is located or somewhere near to those machines. This way they were able to solve the network issues
- Just like I faced the issue of straggler tasks where sometimes task take too much of time to run. Google was able to solve this issue by running back up tasks,
    - Again, this is part of fault tolerance, I will try to implement in the next task
- For partitioning Google used hash key, and I used my approach of set to shuffle the data
- Google order the keys while producing the result and I just used the order in which they are stored