Distributed Systems

Assignment-3

3.1 Introduction:

The assignment focuses on the implementation of mapreduce programming framework based on the paper “MapReduce: Simplified Data Processing on Large Clusters” by Jefferey Dean and Sanjay Ghemawat. The framework mimics the hadoop mapreduce framework.

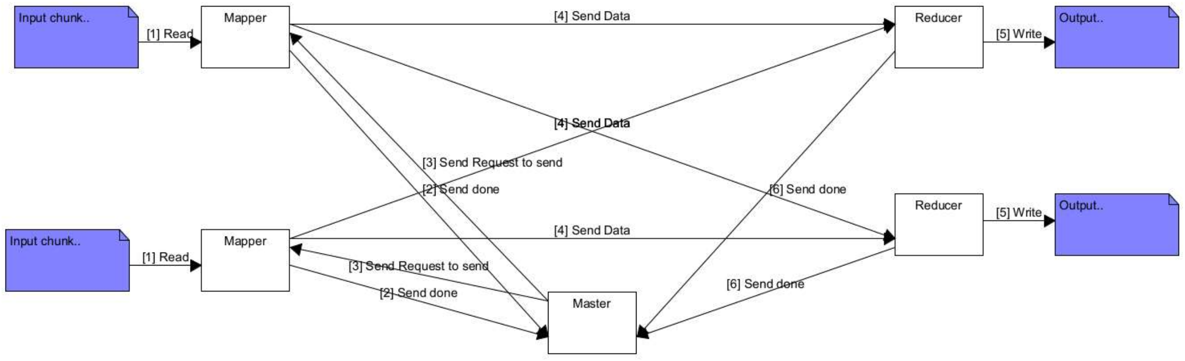


Image showing general mapreduce execution.

3.2 Implementation Details:

The system is built to handle errors and operate reliably. It consists of three parts: the master, reducer, and mapper nodes. Mappers handle the initial file mapping, reducers process data from mappers, and the master coordinates their work. Mappers sort through input files, while reducers combine and simplify data from mappers. The master manages communication between mappers and reducers, ensuring a smooth workflow. This setup ensures that even if there are issues, like hardware failures or network problems, the system can continue running without losing data or functionality.

The master node in the system acts like a manager, moving through different states based on what's happening. It begins by starting up the mapper nodes and then waits until each mapper finishes its job. It keeps track of this by checking for "done" messages from each mapper. Alongside this, it has another thread checking for pulses, which helps it detect if any mapper stops working.Once all the mappers are done, the master moves on to spawning the reducer nodes. Again, it waits for each reducer to finish its task, checking for pulses to ensure they're all still working. As the reducers start working, the master tells each mapper to start sending their mapped data.Once all the reducers have finished reducing, they send a "done" message to the master. At this point, the master knows that all the data processing tasks are complete. It then sends a "kill" message to each mapper and reducer, telling them they can shut down since their job is done.After sending out these messages, the master shuts down itself, effectively ending the entire process.This whole process works smoothly when everything goes as planned, and none of the nodes fail. However, if any mapper or reducer node were to fail during the process, the system would need to have mechanisms in place to handle such failures without causing the entire process to crash. For example, if a mapper fails to send a "done" message, the master might need to have a timeout mechanism to handle such situations. Similarly, if a reducer fails to send its "done" message, the master might need to resend the data or take other appropriate actions to ensure the completion of the task. In essence, the master node's role is to oversee the entire workflow, ensuring that each step is completed successfully and handling any issues that may arise along the way to ensure the smooth execution of the data processing tasks.

The role of the mapper nodes is pivotal in the data processing workflow, primarily tasked with executing the mapping operations. Once a mapper is spawned, it immediately commences the mapping process on the input data provided in the "input.txt" file. Upon completing the mapping task, the mapper initiates a group-by function, consolidating similar inputs and storing the grouped data in the "output.txt" file. Subsequently, the mapper sends a "done" message and enters a waiting state, anticipating a "Request to send" message from the reducers. Upon receiving this message, along with port and host information for the reducers, the mapper employs a hash function to partition and distribute the data to each reducer. This ensures that relevant data is transmitted to the appropriate reducer for further processing. Throughout this data transfer phase, the mapper diligently sends data to each reducer, maintaining the integrity of the processing workflow. Once the mapper has completed its data transmission task, it sends a "done" message to the reducers, signifying the conclusion of the data transfer process. This systematic approach ensures efficient coordination between mapper and reducer nodes, facilitating the smooth flow of data within the system.

The reducer nodes play a crucial role in the data processing workflow, primarily tasked with executing the reduction operations. Upon spawning, a reducer immediately sends a pulse message, signalling its availability and indicating that it is operational. Subsequently, upon receiving data messages from the mapper nodes, the reducer begins storing this incoming data in the input file designated for the reducer. This step ensures that all relevant data is collected and prepared for the subsequent reduction process. As each mapper completes its mapping task, it sends a "done" message to the reducer, indicating that all necessary data has been transmitted. Upon receiving "done" messages from all mappers, the reducer initiates the reduction task. Using the accumulated data, the reducer employs the reduction algorithm to process and distil the information, generating meaningful outputs. Once the reduction task is completed, the reducer sends a "done" message to the master node, signalling the successful execution of its assigned task. This systematic approach ensures that reducers efficiently handle their responsibilities within the data processing pipeline, from data collection to reduction.

3.2 Fault tolerance:

The fault tolerance is implemented to recover either of mapper or reducer nodes. Below stated are various scenarios when a mapper or a reducer can fail

1. **Mapper**
    a. **Mapping task:** The mapper can fail while doing the mapping task and to curb this we can simply restart the mapper and we do not need to change anything on the reducer side.
    b. **Sending task:** Say the mapper fails when it is sending data to the reducers so at this point the master will send a STOP signal to all mappers so that they do not send any data further to the reducers. Now the master signals CLEARINPUT to reducers to indicate that the input buffers should be cleared because it has stale data. Now the master will respawn the mapper and as soon as it gets the done message from this mapper it starts the normal execution of sending a request to send again (This will not spawn the reducers again).
2. **Reducer:** when ever a reducer fails we need to signal a STOP to all the mappers and a CLEARINPUT message to all other reducers indicating that the reducers is down and cannot accept messages. After that we respawn the reducer and as soon as the reducer is up we send a request to send message to all the mappers to indicate sending of data.

3.3 How to test

3.3.1 Folder structure

The folder structure within the "tests" directory is organized to facilitate testing of the mapper and reducer nodes within the system. Each test case, denoted by <TESTCASENUMBER>, contains subfolders representing the home directory for each mapper or reducer node involved in the test scenario. Alongside these directories lies a "conf.json" file, housing various parameters crucial for the configuration and execution of the test cases, such as receive port and host information.

Within each mapper directory, identified by its unique ID, resides an "input.py" file, containing the mapping function responsible for processing chunks of data. Additionally, there's an "input.txt" file, serving as the input data chunk to be mapped by the corresponding mapper. These components collectively enable the mapper node to execute its mapping task effectively.

On the other hand, each reducer directory contains a "reduce.py" file, which houses the reducing function responsible for consolidating and processing data received from the mapper nodes. Additionally, there's a "reducerinput.txt" file, acting as the input file for the reducer node, containing data chunks to be reduced. Furthermore, a "reducerout.txt" file is present, indicating the output produced by the reducer after processing the input data.

3.3.2 Conf.json:

The provided "conf.json" file serves as a configuration file for the system, containing essential parameters for both mapper and reducer nodes. Each mapper is identified by a unique mapper ID and is configured with details such as port number and host information. Additionally, a field named "DEATH" is included to determine whether the mapper node is set to terminate or not. If "DEATH" is set to "True," the mapper node will terminate, and the specific state at which it terminates is indicated by the "DIESTATE" field, which can be either "mapping" or "sending." This allows for testing fault tolerance by simulating mapper node failures at different stages of the mapping process.

Similarly, for reducer nodes, the "conf.json" file includes a "DEATH" field, indicating whether the reducer node is set to terminate. This parameter allows for testing the fault tolerance of reducer nodes by simulating failures during the reduction process.

Overall, the "conf.json" file provides a structured approach to configuring the system's behaviour during testing, allowing for the simulation of various failure scenarios for both mapper and reducer nodes. By adjusting the parameters within the configuration file, testers can effectively evaluate the system's fault tolerance and robustness in handling node failures at different stages of the data processing workflow. This systematic approach to configuration enhances the testing process, enabling thorough validation of the system's performance under adverse conditions without the need for complex or extravagant setups.

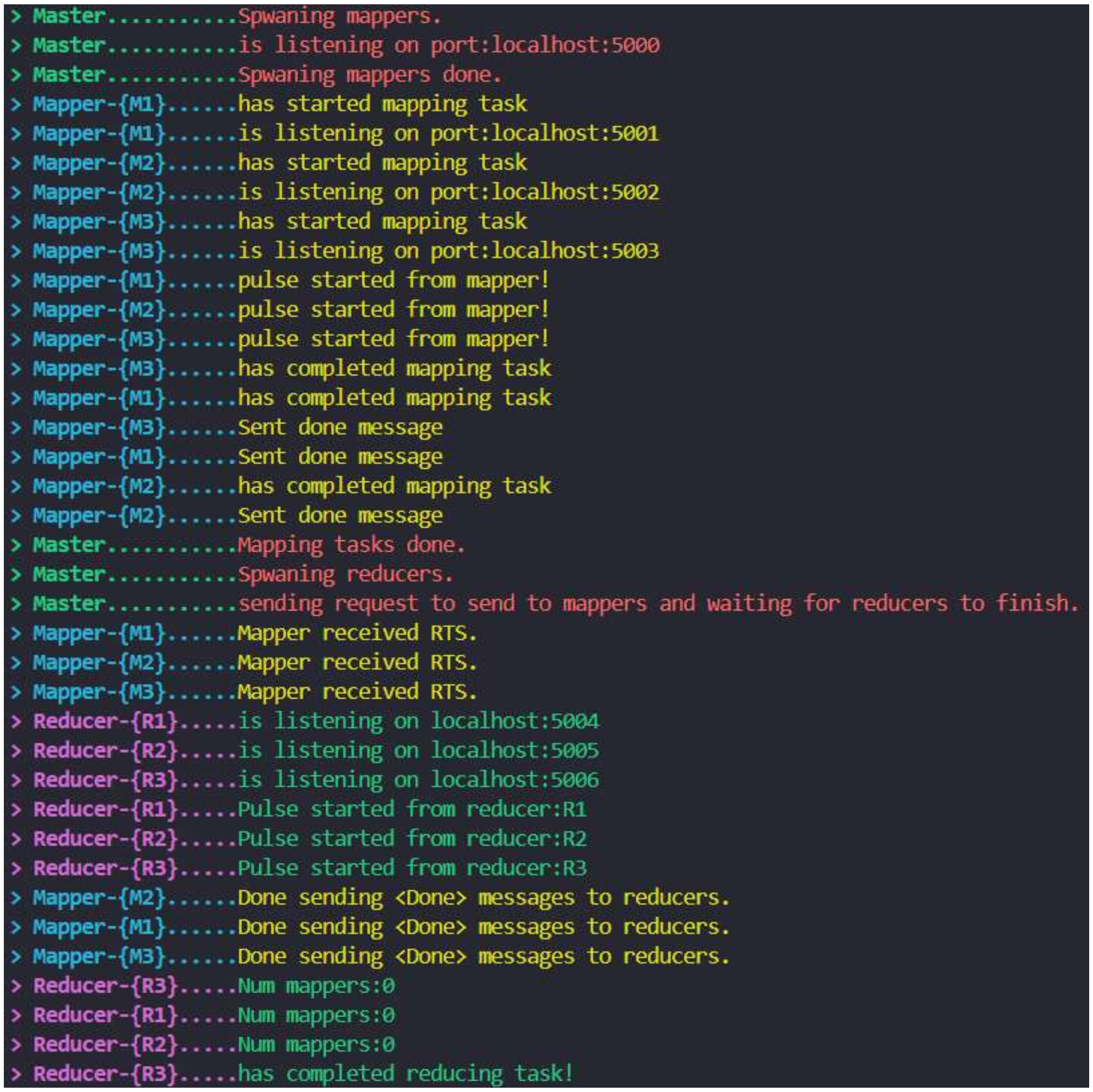
```json
{
  "master": {
    "recvPort": 5000,
    "host": "localhost"
  },
  "mappers": {
    "M1": {
      "id": "M1",
      "recvPort": 5001,
      "host": "localhost",
      "masterPort": 5000,
      "masterHost": "localhost",
      "DEATH": "False",
      "DIESTATE": "None"
    },
    "M2": {
      "id": "M2",
      "recvPort": 5002,
      "host": "localhost",
      "masterPort": 5000,
      "masterHost": "localhost",
      "DEATH": "False",
      "DIESTATE": "Sending"
    },
    "M3": {
      "id": "M3",
      "recvPort": 5003,
      "host": "localhost",
      "masterPort": 5000,
      "masterHost": "localhost",
      "DEATH": "False",
      "DIESTATE": "None"
    },
    "M4": {
      "id": "M4",
      "recvPort": 5004,
      "host": "localhost",
      "masterPort": 5000,
      "masterHost": "localhost",
      "DEATH": "False",
      "DIESTATE": "None"
    }
  },
  "reducers": {
    "R1": {
      "id": "R1",
      "recvPort": 5005,
      "host": "localhost",
      "masterPort": 5000,
      "masterHost": "localhost",
      "DEATH": "False"
    },
    "R2": {
      "id": "R2",
      "recvPort": 5006,
      "host": "localhost",
      "masterPort": 5000,
      "masterHost": "localhost",
      "DEATH": "False"
    }
  }
}
```

3.3.3 How to check the output:

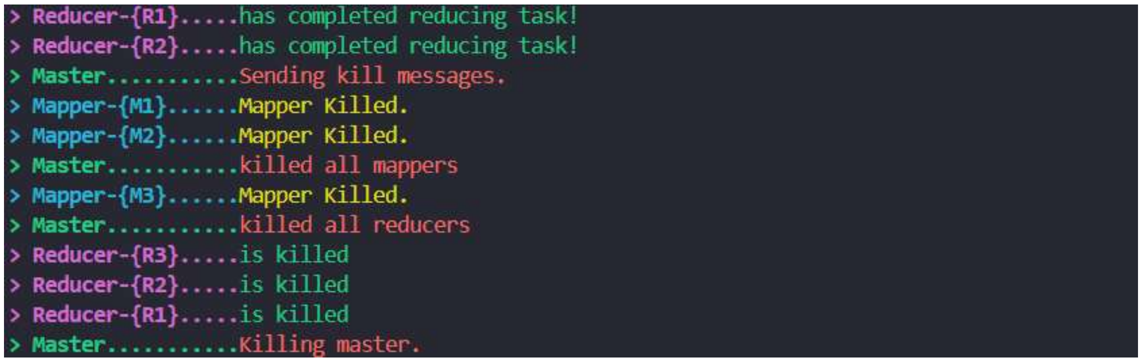
The output of the map reduce task, essential for generating the final output, is stored within the "reducerout.txt" file located within the reducer folders. This file serves as a repository for the processed data chunks, each representing a portion of the overall output. As the reducer nodes complete their reduction tasks, they compile and consolidate the processed data into these "reducerout.txt" file.

3.4 Execution of testcases:

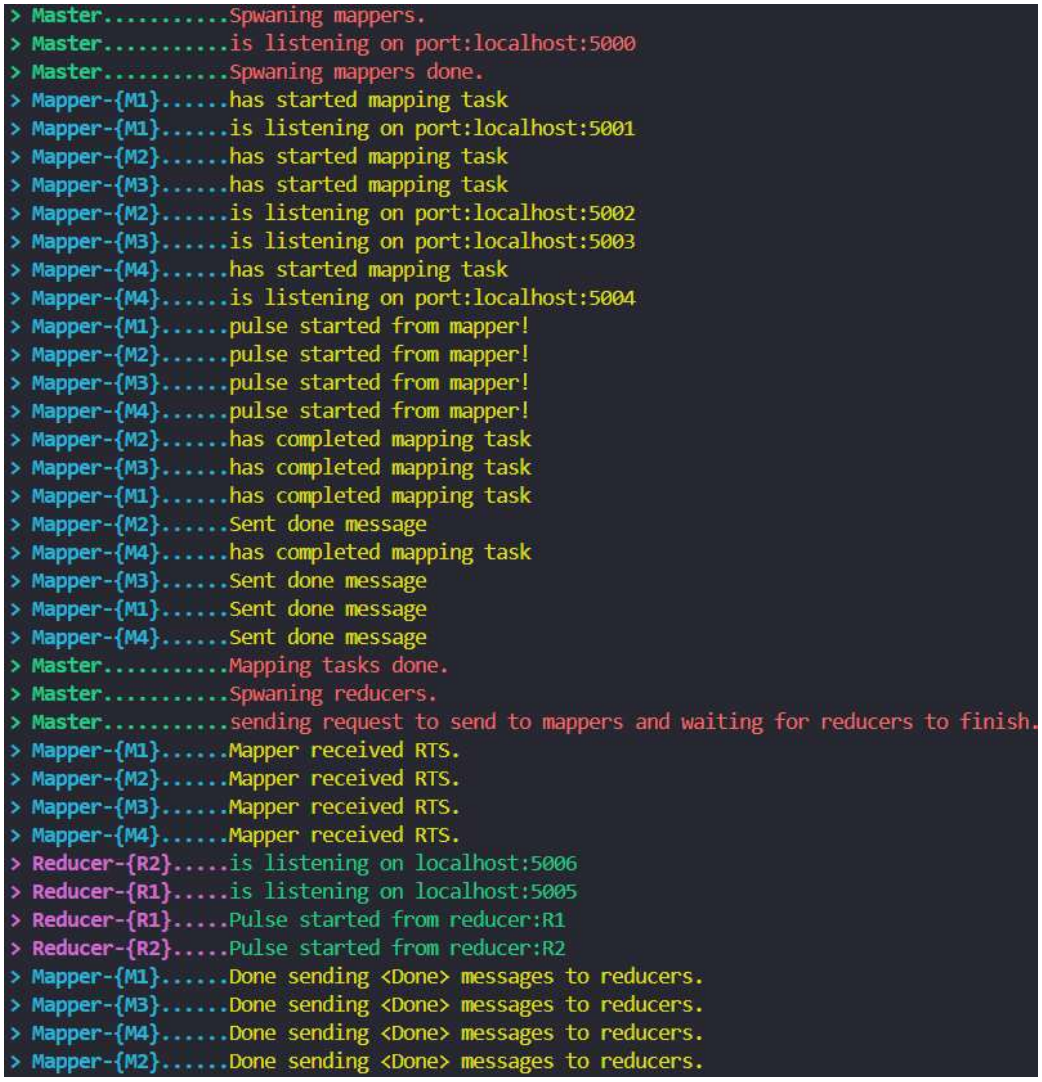
3.4.1 Word count <Normal Execution>



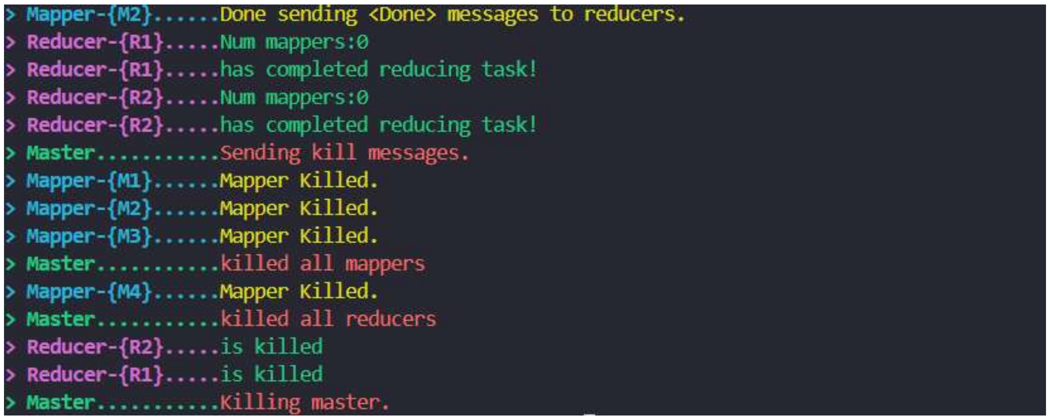
```
> Master...........Spwaning mappers.
> Master...........is listening on port:localhost:5000
> Master...........Spwaning mappers done.
> Mapper-{M1}......has started mapping task
> Mapper-{M1}......is listening on port:localhost:5001
> Mapper-{M2}......has started mapping task
> Mapper-{M2}......is listening on port:localhost:5002
> Mapper-{M3}......has started mapping task
> Mapper-{M3}......is listening on port:localhost:5003
> Mapper-{M1}......pulse started from mapper!
> Mapper-{M2}......pulse started from mapper!
> Mapper-{M3}......pulse started from mapper!
> Mapper-{M3}......has completed mapping task
> Mapper-{M1}......has completed mapping task
> Mapper-{M3}......Sent done message
> Mapper-{M1}......Sent done message
> Mapper-{M2}......has completed mapping task
> Mapper-{M2}......Sent done message
> Master...........Mapping tasks done.
> Master...........Spwaning reducers.
> Master...........sending request to send to mappers and waiting for reducers to finish.
> Mapper-{M1}......Mapper received RTS.
> Mapper-{M2}......Mapper received RTS.
> Mapper-{M3}......Mapper received RTS.
> Reducer-{R1}.....is listening on localhost:5004
> Reducer-{R2}.....is listening on localhost:5005
> Reducer-{R3}.....is listening on localhost:5006
> Reducer-{R1}.....Pulse started from reducer:R1
> Reducer-{R2}.....Pulse started from reducer:R2
> Reducer-{R3}.....Pulse started from reducer:R3
> Mapper-{M2}......Done sending <Done> messages to reducers.
> Mapper-{M1}......Done sending <Done> messages to reducers.
> Mapper-{M3}......Done sending <Done> messages to reducers.
> Reducer-{R3}.....Num mappers:0
> Reducer-{R1}.....Num mappers:0
> Reducer-{R2}.....Num mappers:0
> Reducer-{R3}.....has completed reducing task!
```


```
> Reducer-{R1}.....has completed reducing task!
> Reducer-{R2}.....has completed reducing task!
> Master...........Sending kill messages.
> Mapper-{M1}......Mapper Killed.
> Mapper-{M2}......Mapper Killed.
> Master...........killed all mappers
> Mapper-{M3}......Mapper Killed.
> Master...........killed all reducers
> Reducer-{R3}.....is killed
> Reducer-{R2}.....is killed
> Reducer-{R1}.....is killed
> Master...........Killing master.
```
Outputs can be seen in the reducerout.txt file. Not attaching here because would make the report too lengthy.

3.4.2 Inverted word index <Normal Execution>

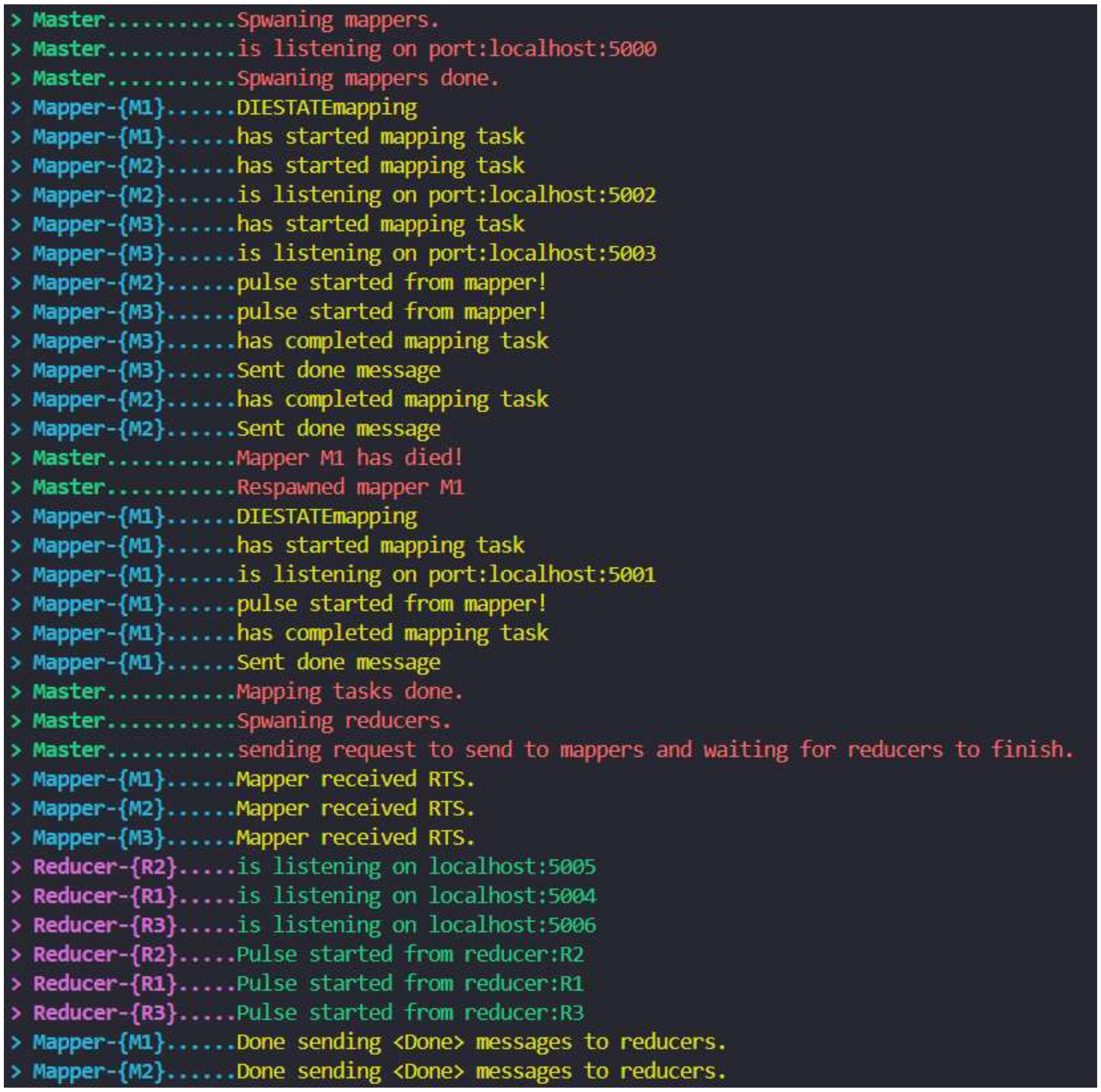


```
> Master...........Spwaning mappers.
> Master...........is listening on port:localhost:5000
> Master...........Spwaning mappers done.
> Mapper-{M1}......has started mapping task
> Mapper-{M1}......is listening on port:localhost:5001
> Mapper-{M2}......has started mapping task
> Mapper-{M3}......has started mapping task
> Mapper-{M2}......is listening on port:localhost:5002
> Mapper-{M3}......is listening on port:localhost:5003
> Mapper-{M4}......has started mapping task
> Mapper-{M4}......is listening on port:localhost:5004
> Mapper-{M1}......pulse started from mapper!
> Mapper-{M2}......pulse started from mapper!
> Mapper-{M3}......pulse started from mapper!
> Mapper-{M4}......pulse started from mapper!
> Mapper-{M2}......has completed mapping task
> Mapper-{M3}......has completed mapping task
> Mapper-{M1}......has completed mapping task
> Mapper-{M2}......Sent done message
> Mapper-{M4}......has completed mapping task
> Mapper-{M3}......Sent done message
> Mapper-{M1}......Sent done message
> Mapper-{M4}......Sent done message
> Master...........Mapping tasks done.
> Master...........Spwaning reducers.
> Master...........sending request to send to mappers and waiting for reducers to finish.
> Mapper-{M1}......Mapper received RTS.
> Mapper-{M2}......Mapper received RTS.
> Mapper-{M3}......Mapper received RTS.
> Mapper-{M4}......Mapper received RTS.
> Reducer-{R2}.....is listening on localhost:5006
> Reducer-{R1}.....is listening on localhost:5005
> Reducer-{R1}.....Pulse started from reducer:R1
> Reducer-{R2}.....Pulse started from reducer:R2
> Mapper-{M1}......Done sending <Done> messages to reducers.
> Mapper-{M3}......Done sending <Done> messages to reducers.
> Mapper-{M4}......Done sending <Done> messages to reducers.
> Mapper-{M2}......Done sending <Done> messages to reducers.
```

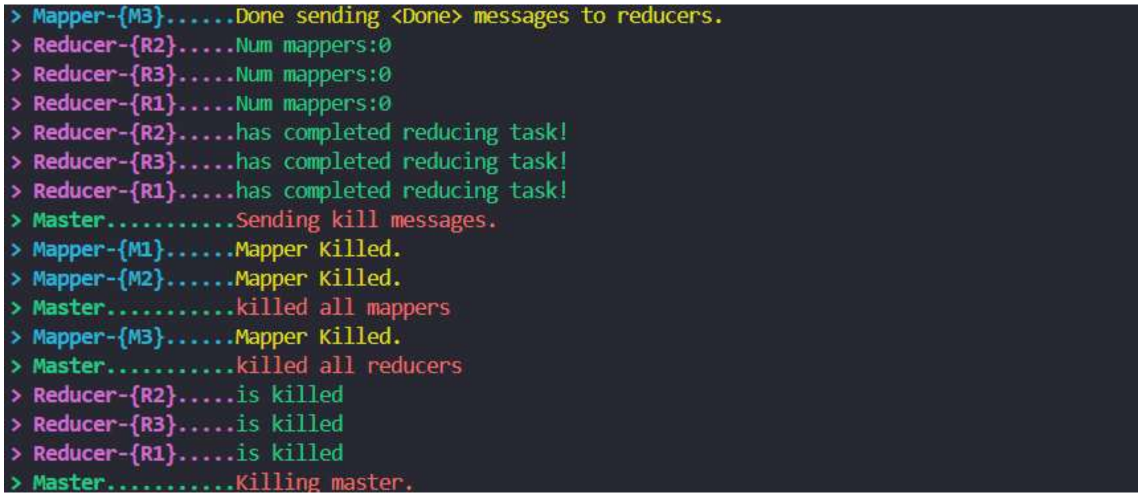


```
> Mapper-{M2}......Done sending <Done> messages to reducers.
> Reducer-{R1}.....Num mappers:0
> Reducer-{R1}.....has completed reducing task!
> Reducer-{R2}.....Num mappers:0
> Reducer-{R2}.....has completed reducing task!
> Master...........Sending kill messages.
> Mapper-{M1}......Mapper Killed.
> Mapper-{M2}......Mapper Killed.
> Mapper-{M3}......Mapper Killed.
> Master...........killed all mappers
> Mapper-{M4}......Mapper Killed.
> Master...........killed all reducers
> Reducer-{R2}.....is killed
> Reducer-{R1}.....is killed
> Master...........Killing master.
```

3.4.3 Word Count <Killing mapper M2>



```
> Master...........Spwaning mappers.
> Master...........is listening on port:localhost:5000
> Master...........Spwaning mappers done.
> Mapper-{M1}......DIESTATEmapping
> Mapper-{M1}......has started mapping task
> Mapper-{M2}......has started mapping task
> Mapper-{M2}......is listening on port:localhost:5002
> Mapper-{M3}......has started mapping task
> Mapper-{M3}......is listening on port:localhost:5003
> Mapper-{M2}......pulse started from mapper!
> Mapper-{M3}......pulse started from mapper!
> Mapper-{M3}......has completed mapping task
> Mapper-{M3}......Sent done message
> Mapper-{M2}......has completed mapping task
> Mapper-{M2}......Sent done message
> Master...........Mapper M1 has died!
> Master...........Respawned mapper M1
> Mapper-{M1}......DIESTATEmapping
> Mapper-{M1}......has started mapping task
> Mapper-{M1}......is listening on port:localhost:5001
> Mapper-{M1}......pulse started from mapper!
> Mapper-{M1}......has completed mapping task
> Mapper-{M1}......Sent done message
> Master...........Mapping tasks done.
> Master...........Spwaning reducers.
> Master...........sending request to send to mappers and waiting for reducers to finish.
> Mapper-{M1}......Mapper received RTS.
> Mapper-{M2}......Mapper received RTS.
> Mapper-{M3}......Mapper received RTS.
> Reducer-{R2}.....is listening on localhost:5005
> Reducer-{R1}.....is listening on localhost:5004
> Reducer-{R3}.....is listening on localhost:5006
> Reducer-{R2}.....Pulse started from reducer:R2
> Reducer-{R1}.....Pulse started from reducer:R1
> Reducer-{R3}.....Pulse started from reducer:R3
> Mapper-{M1}......Done sending <Done> messages to reducers.
> Mapper-{M2}......Done sending <Done> messages to reducers.
```



```
> Mapper-{M3}......Done sending <Done> messages to reducers.
> Reducer-{R2}.....Num mappers:0
> Reducer-{R3}.....Num mappers:0
> Reducer-{R1}.....Num mappers:0
> Reducer-{R2}.....has completed reducing task!
> Reducer-{R3}.....has completed reducing task!
> Reducer-{R1}.....has completed reducing task!
> Master...........Sending kill messages.
> Mapper-{M1}......Mapper Killed.
> Mapper-{M2}......Mapper Killed.
> Master...........killed all mappers
> Mapper-{M3}......Mapper Killed.
> Master...........killed all reducers
> Reducer-{R2}.....is killed
> Reducer-{R3}.....is killed
> Reducer-{R1}.....is killed
> Master...........Killing master.
```

As you can see in the log that the mapper died during the execution but it got reaspawned again.

Similarly we can modify and change the states for each mapper and reducer to see the fault tolerance.

3.5 Assumptions:

1. We assume that while doing the mapping and reducing task we have enough memory to process the chunk we are provided.
2. We assume that the master would never die and we do not have to do the fault tolerance on the master node
3. Also we assume that the ports we use are always available to be used for a mapping or a reducing node. We do not have to kill the mapper or a reducer port in order to spawn it.
4. We assume that the reducers output will somehow be combined to be made a final file which is a combination of all the reducer outputs.

3.6 Future Extensions:

1. The mapper and reducers can use HTTP protocol to communicate between the nodes. This is because the nodes can be located between multiple servers and make it more generic.
2. We can also use the centralised KV store in order to exchange the states between the mappers and reducers.
3. The fault tolerance could also be added to recover the master node if it fails while doing the execution

3.7 References:

[1] Socket.io: https://python-socketio.readthedocs.io/en/stable/
[2] Threading: https://docs.python.org/3/library/threading.html
[3] Multiprocessing: https://docs.python.org/3/library/multiprocessing.html
[4] OS module: https://docs.python.org/3/library/os.html
[5] Google paper:
https://static.googleusercontent.com/media/research.google.com/en//archive/mapreduce-osdi04.pdf
[6] Github Repo: https://github.com/jenilgandhi2111/E510-DS
[7] Fault tolerance strategies: https://www.datsi.fi.upm.es/~mperez/pub/springer2016.pdf

—---