Introduction:
The report contains the design and implementation of the distributed mapReduce library. In this distributed mapReduce system, the client provides the map and reduce functions and mapreduce does the mapping and reducing tasks accordingly by distributing the data into workers. The final output is then stored at a location requested by client

Design Decisions
The mapper and reducer functions are taken as inputs from the client.
Client then calls the mapreduce library by passing these functions and input data to the mapreduce.
The architecture is as follows:

1. KVStore for persistent data storage. This server is started before the mapreduce application starts. The KVStore will be discussed in more detail in the following section. KVStore is served on a different process/application
2. After KVStore application is started, mapreduce application starts by spawning or creating a master node for the map reduce application. Client will interact with this node for the mapreduce tasks which are exposed using a rpyc library in python.
   APIs exposed by KVStore:
   set(key,value)
   get(key)
   append(key)

3. Client calls init_cluster and provides with the ip addresses on which the master spawns the worker processes
4. These are general purpose workers and can be used as mappers or reducers. These workers after being spawned starts their own servers and exposes APIs through RPC which will be used by the master to communicate with these workers.
   APIs exposed by workerNode:
   execute(func, input_data)

   MAP PHASE

5. After the data is passed, the master distributes the input data equally among all the workers and assigns them the mapper tasks. Hence these workers will be called mappers in the map phase
6. The mappers are running as independent nodes and will process the data parallely.
7. The mappers are also passed the mapper function received from client and will perform or execute the task accordingly.
8. After each mapper has done processing the data, they will write the intermediate into persistent storage using key value store (***KVStore***)
9. Each mapper will append the value to the already existing value of the key. In this way the aggregation of the data is managed on the fly and thus the task will be reduced for the master.

INTERMEDIATE DATA PHASE

10. Since the mappers have already aggregated the data according by the key, master only has to distribute the keys and give them to reducers for the reducer tasks.
11. This is done by mapping each key to a certain reducer based on the hash function so that the keys are distributed uniformly.

REDUCE PHASE

12. Since the map phase is done all the workers will be free and hence can be used as reducers.
13. The reducers are then given the list of keys as input and are asked to execute according the reducer function provided by the client.
14. Note that the functions are passed by marshalling and are passed over the network using RPC
15. The reducer then sends the output back to the master.

FINAL OUTPUT

16. The master then merges the result from all the reducers and writes the final output to the location requested by client

Data Partitioning

MAP PHASE
The input data for word count is simply divided equally amongst all the workers for map phase. This is done by calculating the total number of words in the file and then dividing equally among the workers

For the inverted index client, the length of input data is calculated as the total number of files in the directory.

REDUCE PHASE
For the reduce phase the master is aware of all the keys which it gets through the KVStore(discussed later). It then divides the keys equally amongst all the reducers by using a hash function

Data Storage
For persistent data storage I have made use of KVStore. The KVStore has exposed following APIs through RPC

1. set(key,value) : sets the value for the key
2. get(key): returns value
3. append(key, value): appends the value to the already existing value for that key
4. getAllKeys(): returns all the unique keys

5. setupPersistentStorage: This method sets up a directory for the map reduce client by making use of the cluster id. Unique Cluster id is generated and hence this storage will be unique for a given mapreduce instance.
6. removePersistentStorage: This method removes all the files and intermediate data. This is triggered from destroy cluster. Intermediate data can be seen if destroycluster is not called from the client

Note that I have used key as the filename and hence the data can be appended directly into the file by the mappers. This saves efforts for the master to group the data by keys

Dynamic membership

Since worker node is generic it can be used as mappers or reducers

Implementation assumptions

1. No node failure
2. Reliable connection
3. Reliable persistent storage
4. No special characters for the word count client as I have used a direct split
5. We have to manually stop the servers by Ctrl+C

Communications between master and workers

The workers are API servers and the master is the client in RPC context. Master communicates with the worker nodes using worker APIS exposed through RPC

Test Cases

I have tested the application on several files and have attached the results in the output folder.

How to run?
Path : Server/
First start KVStore:

1. Start KVStore - *make START_KVSTORE*
2. Open a new console and start mapreduceMasterNode - make START_MR_MASTERNODE
3. Open a new console and start client - START_WC_CLIENT or START_INV_IDX_CLIENT

Note: I have added a config file which is causing some error in running the code. I will upload the new submission as soon as it gets fixed.