SimpleMR

MapReduce Framework Rewritten

Team Members:

S.No	Name	Contact
1.	Akanksha Mishra	Mishra.ak@husky.neu.edu
2.	Harshali Singh	Singh.h@husky.neu.edu
3.	Saahil Singla	Singla.s@husky.neu.edu
4.	Vishal Mehta	Mehta.vis@husky.neu.edu

Table of Content

1. Abstract

SimpleMR is a framework which is able to process unstructured data in parallel across distributed cluster of instances or on a stand alone mode.
The main idea behind this project is to under the working of Apache Hadoop MapReduce and build a system similar to it which should run both in parallel on Amazon EC2 and sequentially or in parallel on the local machine. Specific MapReduce programs should be run by SimpleMR system and their output is compared with that produced by Hadoop MR and the performances of both the systems are compared. The machines running SimpleMR must be capable to run Java code.

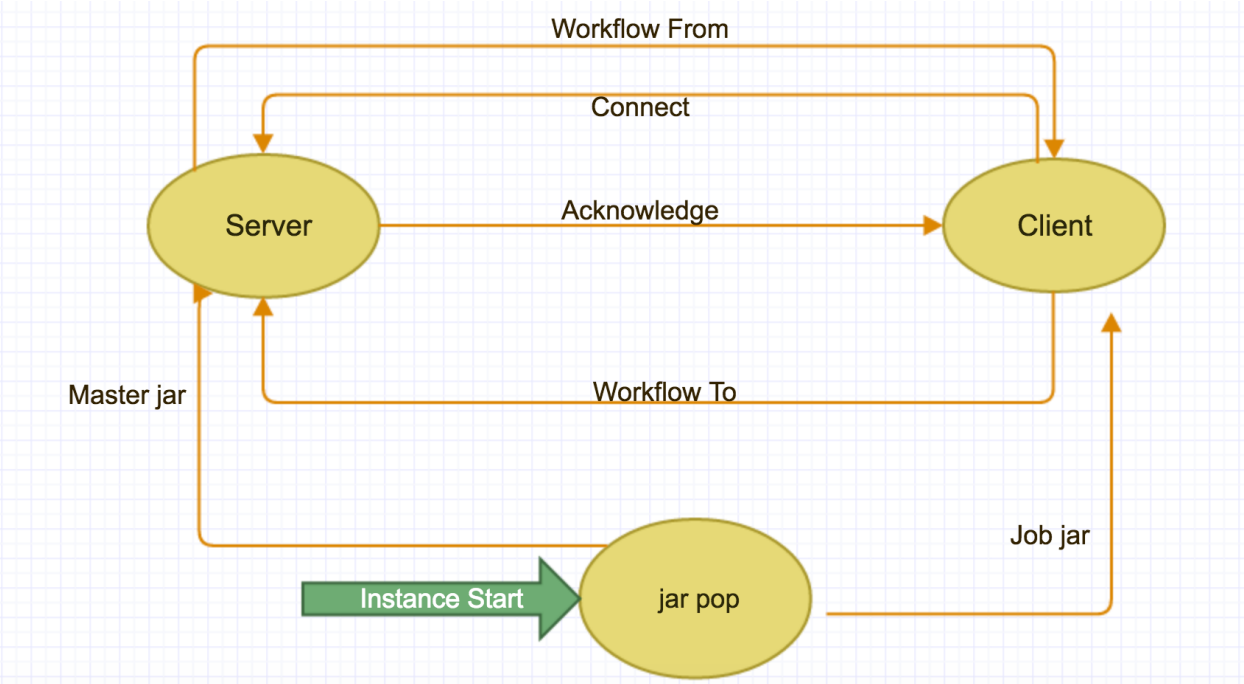
2. Introduction

MapReduce is a programming model and an associated implementation for processing and generating large datasets with a parallel, distributed algorithm on a cluster.
To the programmer, MapReduce is largely seen as an API. MapReduce is implemented in a master/worker configuration, with one master serving as the coordinator of many workers. A worker may be assigned a role of either a map worker or a reduce worker.
MapReduce tasks get pairs of keys and values as input and generate another pair of keys and values as output with the same/different types.

- Split Input: The large input dataset is split into a number of chunks of fixed size. These chunks are read by the Mapper line by line.
- Map(): This function transforms the input split into intermediate records. Multiple Map tasks are initiated in parallel and the input is converted into <Key,Value> pair.
- Combine(): This method combines all the values for the same key in the form of an iterable list.
- Shuffle(): Data collected from different mappers is split amongst the reducers according to the keys and each reducer is provided with all the values associated with a particular key. The Partition() function decides which reducer should get any particular <Key,Value> pair.
- Reduce(): The reducer obtains the <Key,Value> pairs sorted by key and emits output for every key.

In our SimpleMR implementation, we have used Amazon S3 distributed file system. Our architecture is Client-Server based in which we determine active clients by sending and receiving heartbeat messages from master to clients. The master then assigns the jobs to the clients and combines the intermediate outputs received by them.
The speciality of our implementation is compatibility with the MapReduce programs and its ease of use.

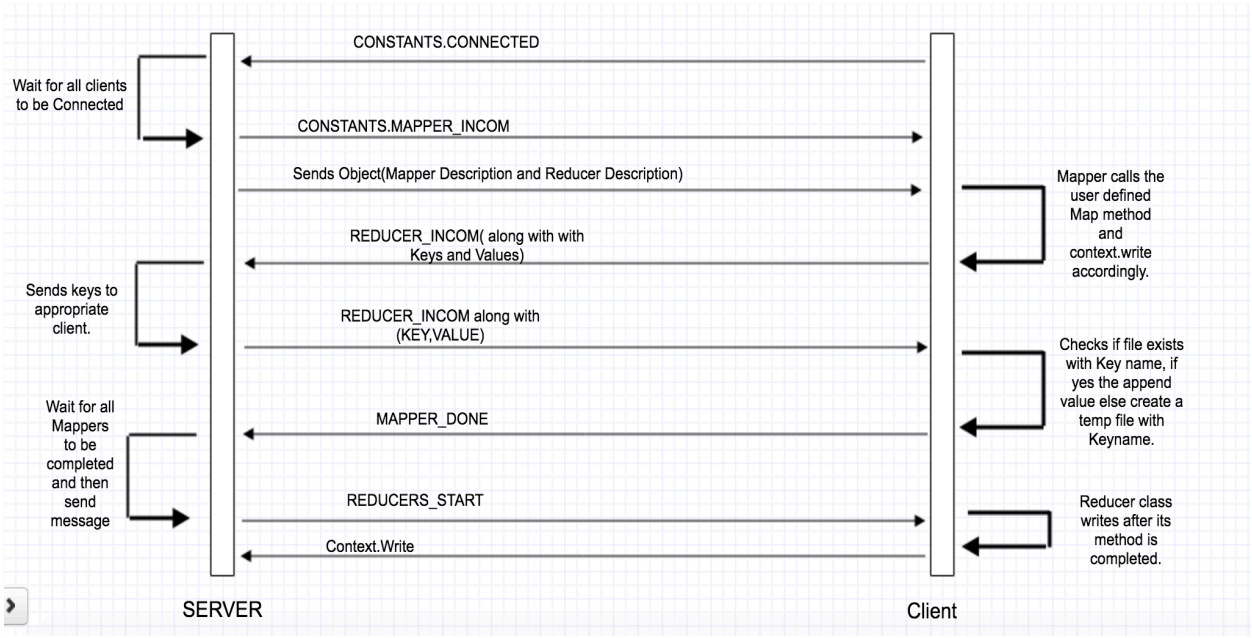
3. Architecture Diagram



Steps:

1. The script will start the EC2 instances and and make them ready.
2. The appropriate jars are popped i.e. Master jar is sent to the Server and Job jar copies are sent to all the clients.
3. Now the Master starts listening on appropriate ports and clients try to connect with the Master on the same ports.
4. After successful connection, Master sends acknowledgement to the client.
5. The job distribution and workflow is then started between the Master and the Clients.

4. Framework Workflow



4.1 Framework Initialization

We initialize the framework by starting the EC2 cluster machines and installing all the required settings and operating system on it. One EC2 instance becomes as the master while the other machines act as slaves. The master jar is fed to the Master and the job jars are fed to the slaves.

4.2 Client – Server Execution

Our design consists of one server many clients. Server starts listening for all the incoming connections. Following is the workflow explanation between client and server

4.3 Job Distribution

1. The Master or the server uses randomized approach to distribute tasks amongst the clients. To distribute the files, the server retrieves the input files from S3 and create the splits.
2. Each client send a message "CONNECTED" to the server as soon as it extracts the server IP and connects with the server on the same port.
3. The Server waits for all the clients to get connected and then sends the message "MAPPER_INCOM" to all the clients.
4. Then the server sends MapperDescription object containing the mapper class along with the file to read.
5. The Client will execute the Map method of the overridden Mapper method and sends the mapper output files via Context.write method.
Once the Client executes the Mapper, it will send the message "REDUCER_INCOM" to the server.
6. The Server after receiving this message will filter which Client should have the particular Key value pair.

7. The "REDUCER_INCOM" message is sent back to the Client by the Server along with the filtered Key-Value pair.
8. The client will first check if the file related to that Key-Value pair output exists and appends the output for that key in the same file, if the file does not exist, it creates a new file with the Keyname.
9. Once all the Key-value pairs are appended in the respective Clients, the Client will send "MAPPER_DONE" message to the Server.
10. When all the "MAPPER_DONE" messages are received by the Server, the Server will send "REDUCER_START" message to the clients at which point the Client will invoke the Reducer class and the reduce method is called. The Client then sends their respective output files to the Server and then the Server will combine all the outputs by the Clients and send them to S3 system.

4.4 Mapper

Map task is assigned to each client by the server. The client runs the mapper task to the files assigned to it by the server. The data from the files is read line-by-line to produce the intermediate output. The intermediate output is in the form of key-value pairs. The mapper is an abstract class that declares the abstract map() function. The user program e.g. WordCount can extend this class.

4.5 Reducer

The reducer takes a key and a collection of values. It then aggregates the values to produced the desired result for the given key. Just like the mapper phase, the data is distributed evenly in the reducer phase i.e. the keys are distributed evenly. The reducer is an abstract class that declares the abstract reduce() function. The user program extends this class to include his desired logic.

4.6 Fault Tolerance

The server keeps sending “HEARTBEAT” message to the clients periodically to keep track of which clients are active. If during the execution, the client becomes inactive, then the tasks which are assigned to it are distributed randomly amongst the active clients.

5. Implementation

5.1 Strategy:

Cluster Management: We have developed 3 shell scripts to manage the EC2 instances which are as described below:

- *start-cluster.sh* - This script starts N number of EC2 instances. N is specified in the argument. It parses the JSON response of *describe-instances* and retrieves the **Public-DNS** and **Public-IP** of all the running instances which writes to dns.txt & iplist.txt files respectively.
- *my-mapreduce.sh* - This script executes the job jar and passes the user's arguments.
- *stop-cluster.sh* - This script fetches instance ids of all running instances and stops those instances.
- *script.sh* – This script executes the master, slave, job jar based on whether the mode of execution is EC2 or Local.

Server implementation:

The server module of SimpleMR application is the central point of execution which gets launched with the execution of server jar copied to one of the instances. The server has the following features:

- Monitoring Heartbeat of the client
- Distributing the input to the clients
- Job scheduling
- Collecting the outputs from the client

Client implementation:

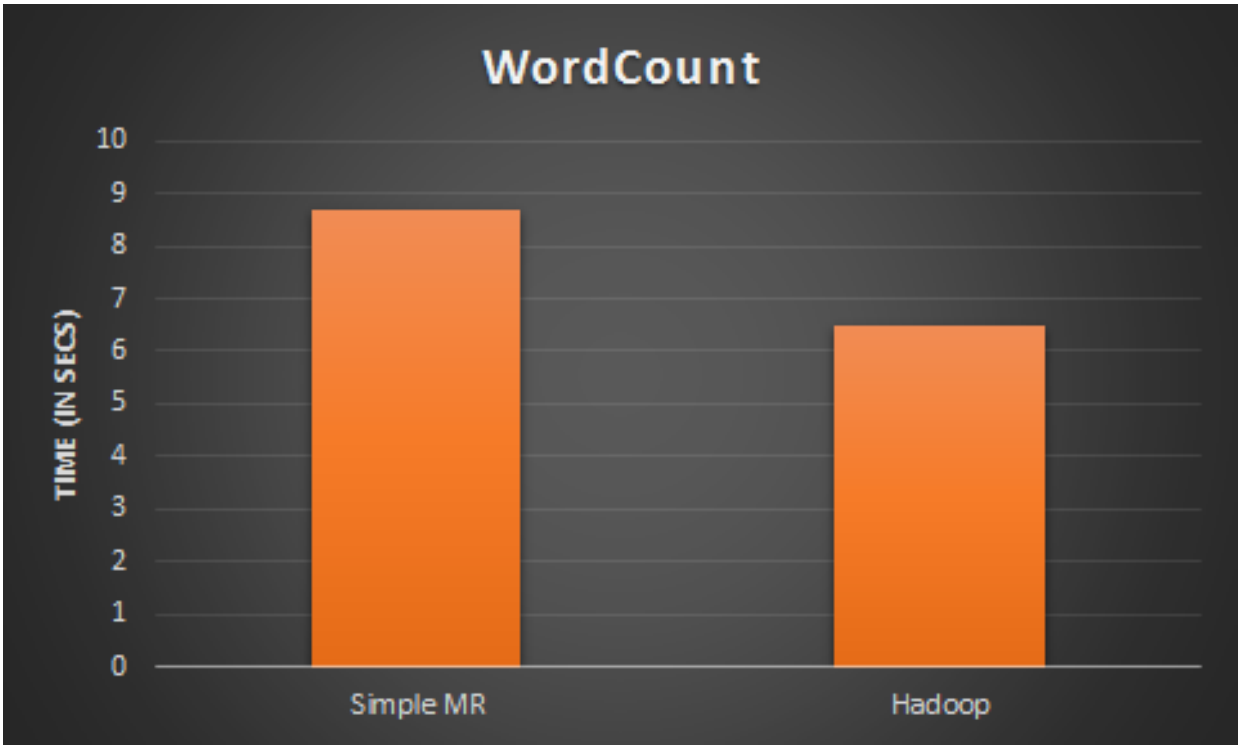
The client is responsible for executing the jobs sent by the server. A client has the following features:

- As soon as the client is up, it connects itself to the server using the server ipaddress using a TCP connection
- When the client gets a command from the server to execute the Mapper, it collects the information from MapperDescription about the input file split and the address where the fileoutput should be written. The same information is collected from ReducerDescription when the server commands to execute Reducer.

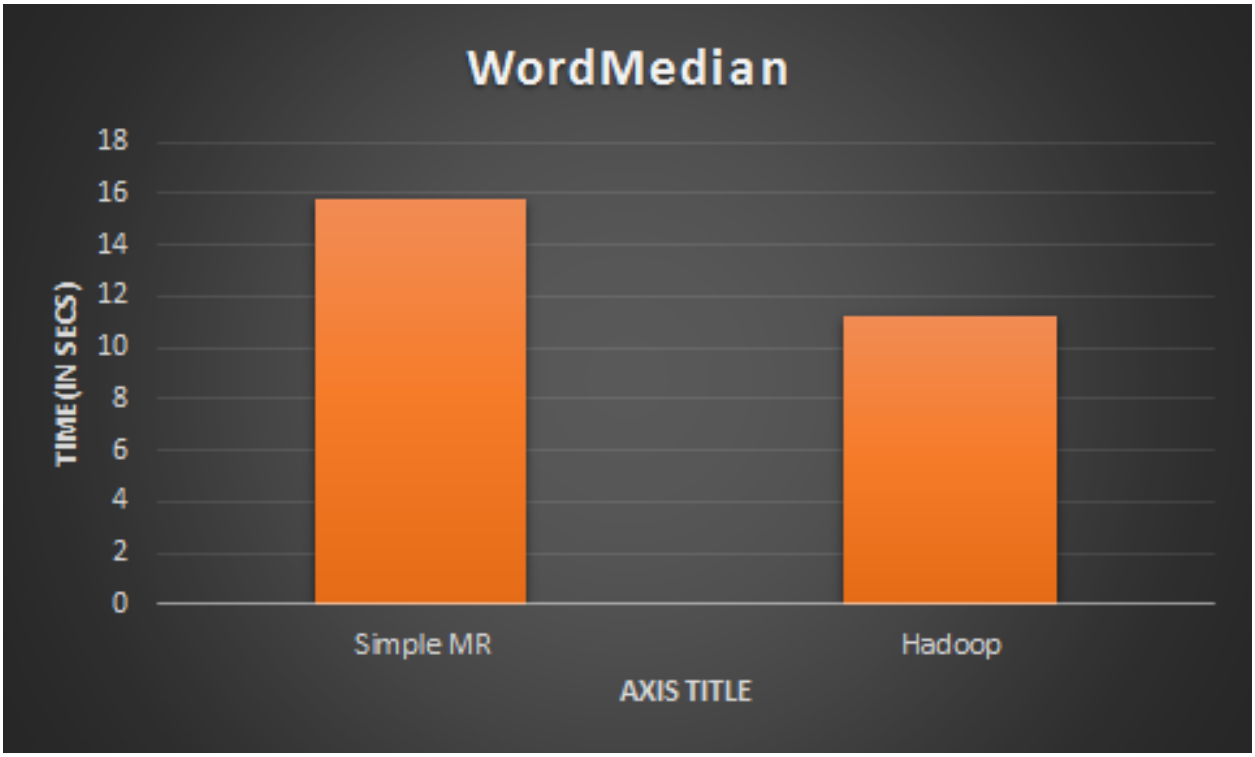
6. Performance

Program	New functionality required	Performance (vs. Hadoop)	Expected Output Produced?
WordCount	Updated Mapper and Reducer class according to our framework	Refer graph.1	Yes
WordMedian	Updated Mapper and Reducer class according to our framework	Refer graph.2	Yes
ClusterAnalysis	Updated Mapper and Reducer class according to our framework	Refer graph.3	Yes
MissedConnections	Updated Mapper and Reducer class according to our framework	Refer graph.4	Yes
Prediction & Routing	Updated Mapper and Reducer class according to our framework	NA	No

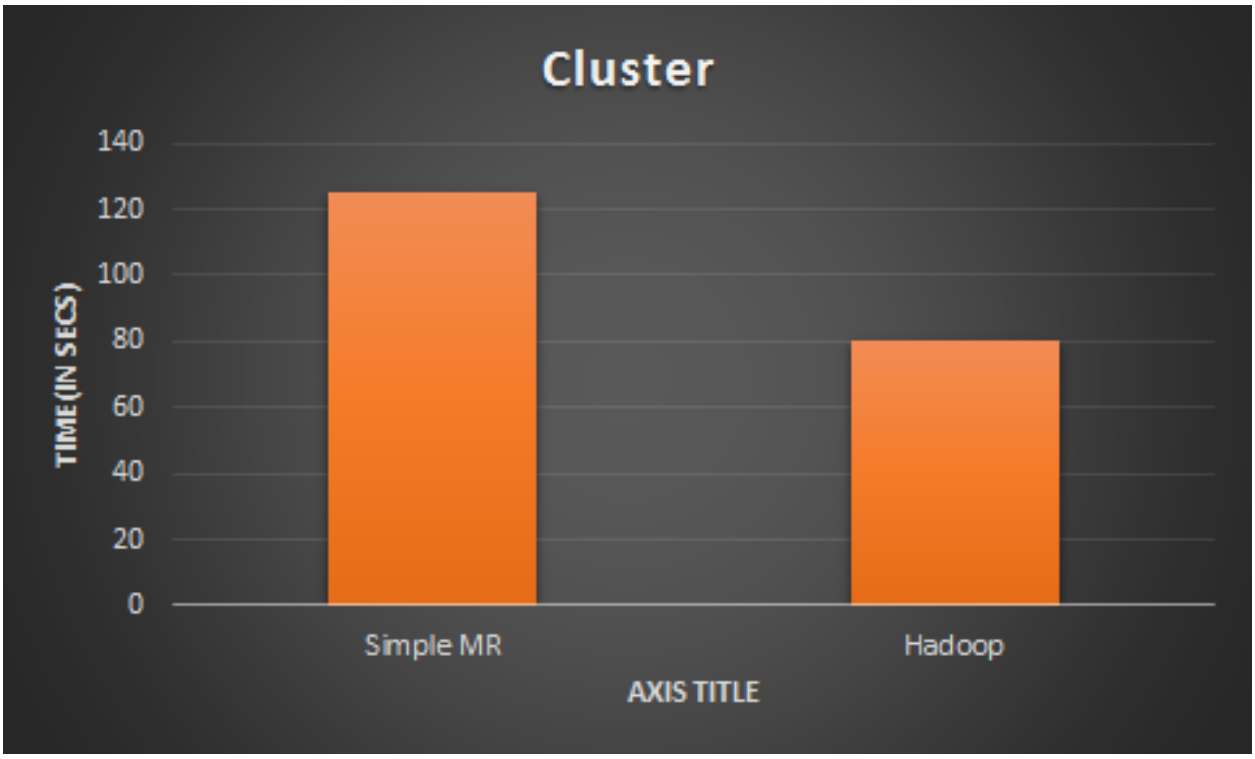
Graphs:



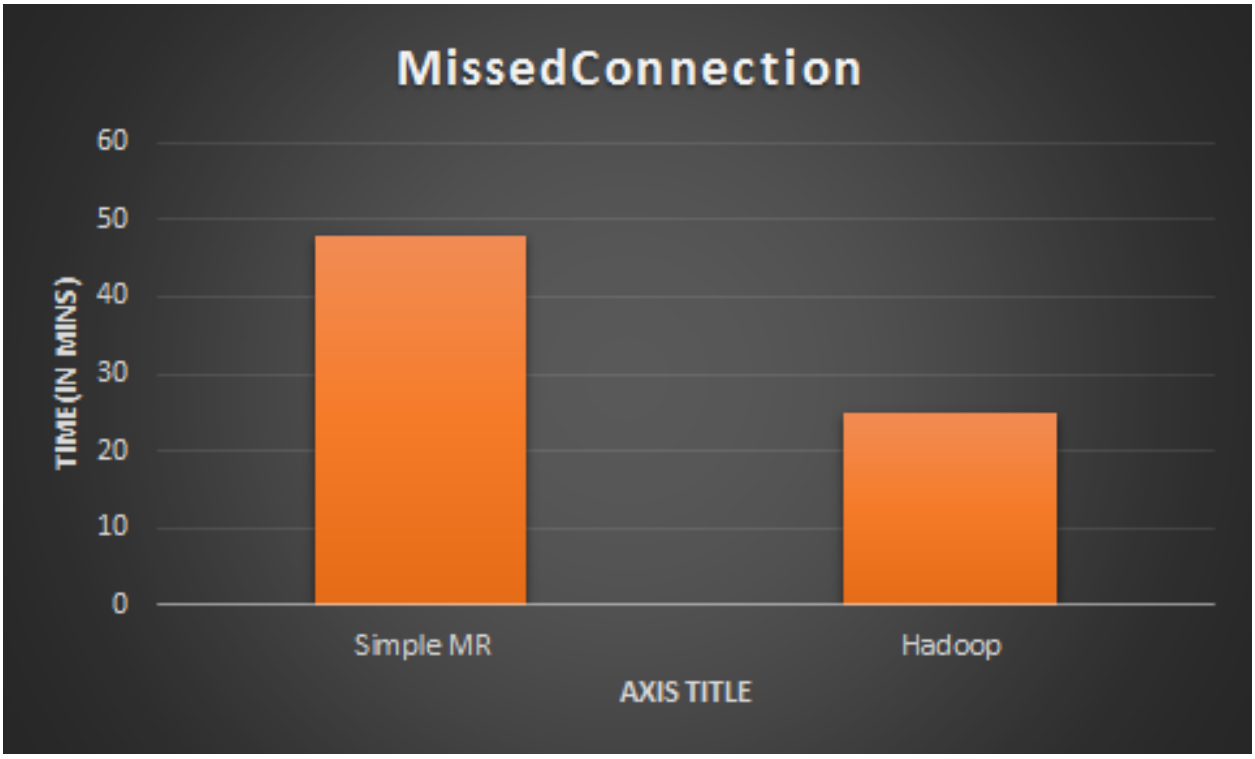
Graph.1



Graph.2



Graph.3



Graph.4

These graphs depicts time took for Pseudo Mode with same input size.

Configuration of machine on which these programs were run are:

Processor: 2.6GHZ Intel Core I5
Memory: 6 GB DDR3
OS: Linux
Make: Dell

7. Future Scope

The project can be further extended to achieve the following features:

- Job progress of Mapper and Reducer
- Extending the Mapper and Reducer to take any kind of datatypes as their keys and values
- Implementing a shared file system or adapting to HDFS for sharing files instead of relying on S3 to make the system fast

8. Challenges Faced

- We were partially able to execute Assignment 7 (Prediction & Routing) using our framework as we were not supporting Custom Writable classes.
- Initially we had issues with dependencies between Master and Slave as we built them as one project and making them to run independently was a challenge.
- Our framework's architecture supports one master- one client communication easily, however we faced multiple issues while implementing one master and multiple clients architecture.
- Executing shell scripts within java code was a strenuous task.

9. Individual Contribution:

Vishal Mehta: Worked on building the scripts to start, stop the cluster and my-mapreduce script. Also, managed the instances on ec2 and worked on the logic for Master.
Harshali Singh: Worked on developing the infrastructure to implement master-slave communication. Also contributed in developing the scripts.
Saahil Singla: Worked on logic for Slave and job scheduling. Also contributed in testing the job examples on our framework.
Akanksha Mishra: Worked and contributed in building the Network Manager and also spent lot of time in building the Master-Slave architecture.

Team Contribution: Project design and architecture, Readme, Report, Running and testing code on EC2 instances, Resolving the challenges faced during implementation of entire project.

10. Conclusion

This project SimpleMR is developed to understand the Hadoop MapReduce framework and rewrite its APIs so that we are able to run MapReduce programs. It is a much smaller prototype of Hadoop MapReduce framework and can be improved in terms of performance, functionalities and accuracy. We realized that the distributed system is not enough to increase the performance, and our standalone system worked better than the distributed system.

11. Acknowledgement:

The SimpleMR project consumed a lot of research and work. We would like to thank Prof. Nat Tuck without whose guidance throughout the semester, the implementation would not have been possible.
We would also like to thank our TAs', Yogendra Miraje, Shreyas Mahimkar, Monisha Singh for clearing our doubts all the time and lastly our classmates as it was due to them we learnt great and innovative ideas and it was a good experience learning with such great minds.

12. References:

- https://hadoop.apache.org
- Hadoop the definitive guide: Tom White
- AWS Documentation:
  http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/authorizing-access-to-an-instance.html

  http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/TroubleshootingInstancesConnecting.html#TroubleshootingInstancesConnectingMindTerm

  http://docs.aws.amazon.com/codedeploy/latest/userguide/how-to-create-iam-instance-profile.html#getting-started-create-ec2-role-cli

- Stackoverflow: www.stackoverflow.com
- https://wiki.apache.org/hadoop/HadoopMapReduce