Project Report

Team Members:

Kaushik Veluru, Habiba Shaik, Neelesh Nidadhavolu, Sudeep Vaka and Aditya Ramesha

1. Aim : The aim of this project is to develop a Map Reduce framework that replicates the Apache Hadoop Framework.

2. Goal: Since Hadoop framework is a fully functional commercial product, we would like to develop a Minimum Viable Product version of Hadoop that supports the features we have used over the duration of this course (CS6240).

3. Components:

3.1. Hadoop

A package that contains all the Classes we developed to replicate all the Hadoop features we have selected to support.

3.2. Client

A package that contains the client side functionalities. This plays a Master role in the Master - Slave architecture coordinating with all the associated slaves.

3.3. Server

A package that contains the server side functionalities. This plays a Slave role in the Master - Slave architecture by constantly updating its status to the client and listening for commands from client about what task to take up next.

3.4. S3

AWS S3 acts as HDFS for our implementation of Hadoop. Multiple S3 instances based on the parameter specified with the start-cluster script act as the cluster nodes in the HDFS architecture. All input, intermediate and final output files are written into these S3 instances.

3.5. Scripts

We have 3 steps which provide the front-end interface commands and imitate the hadoop start, hadoop run and hadoop stop commands.

3.5.1. start-cluster.sh

This script starts the servers either in pseudo mode or distributed mode based on the arguments passed to it.

3.5.2. my-mapreduce.sh

This script initiates the map-reduce functionality and produces the part-files expected from a Map Reduce program.

3.5.3. Stop-cluster.sh
This script stops all the server instances started by start-cluster.sh

4. Supported modes
We support both **Distributed** and **Pseudo** modes. The main difference between these two methods is, the server instances are created in AWS EC2 in distributed mode and the servers are created with different port numbers on the same local host in pseudo mode. Both the modes use S3 as their file system.

5. High-level Design
5.1. Start Cluster

./start-cluster.sh <No. of Instances> distributed

a. Create instances in EC2.
b. Write the instance details: instance-id and instance-ip in a file -> instances.txt

5.2. My Mapreduce

./my-mapreduce <application> <input> <output> <mode>

a. Builds the application.jar file.
b. Reads instances.txt created by start-cluster.sh and for each instance copies the required files keypair.pem, config.properties, application.jar to the server instance.
c. Executes the application.jar file in server.
d. Server opens up a connection and starts listening for the connections at this point.
e. Once all the server instances are called the client.jar is executed which starts the back and forth connections between server and client which orchestrates the flow of Map Reduce program. The flow is explained in the below steps.

5.3. Stop Cluster
./stop-cluster.sh

Reads the instances.txt to fetch the instance-id and executes aws command to stop the instances.

6. Client - Server communication

This section describes the communication between the client [Master] and server [Slave] which is the key for this implementation of our project.

Server : Opens a connection and starts listening for connections.
Client : Sends a connection request.
Server : Accepts the connection. Waits for command
Client : Sends InputBucket name
Server : Receives InputBucket name and waits for command

Client : Sends instance IP address
Server : Receives instance IP and waits for command
Client : Sends instance port
Server : Receives instance port and waits for command
Client : Sends OutputBucket name
Server : Receives S3 outputBucket name and waits for command
Client : Fetches the input data from InputBucket/inputFolder location and divides the data in to equal chunks for each server instance into S3://inputBucket/instanceIP/input location.
Client : Sends **MAPPER_START** command indicating the servers to start the Map phase of the mapreduce program
Server : Receives MAPPER_START and starts mappertask().

In mappertask() it does the below steps:

a. Fetches its corresponding input chunk from S3.
b. For each file in the input folder, invokes a map() method.

In map() it does below steps:

a. Executes the map() logic on the given input and writes the Key value pair in an intermediary file with Key as the filename and each value in a new line.
b. All the intermediary files created by map() are uploaded to s3 to instanceIp/tempFiles location
c. Sends **MAPPER_COMPLETE** signal to the client.

Client : Waits for MAPPER_COMPLETE signal from all the servers it is associated with. Once it receives MAPPER_COMPLETE from all its servers, it does below steps:

a. Downloads intermediary key files created by all the servers to a local file system.
b. Merge all the files with same filename.
c. Uploads all the merged files to S3 output folder.
d. Sends REDUCER_START signal to all the servers.

Server : Receives **REDUCER_START** signal and start reducetask()
In reducetask() it does below steps:

a. Downloads the merged intermediary key files to its local file system.
b. Invokes reduce() method for each file:
    In reduce() it performs below steps:
    a. Executes the reduce() logic for each key file and writes the output to a part-<InstanceIP> file.
c. Uploads the part-files to S3://outputBucketName/output folder
d. Sends **REDUCER_COMPLETE** signal to the client and closes the connection.

Client : Waits for REDUCER_COMPLETE signal from all the servers it is associated with and closes all the connections once it recieves REDUCER_COMPLETE signal from all the server. Also, downloads all the part files to the local file system which is similar to hdfs get method.

7. Problems
There were many problems we faced over the course of development of this project. Only the notable ones are mentioned below.

7.1. Generics
Working with Java reflection and making the methods with generic parameters and return types were most challenging. We did work around at few places to make this work as there are very few data types we are using at this point. We have spent a lot of time on this issue and didn’t have much time to proceed further.

7.2 Connectivity Issues
Client Server communication issues on local and ec2 was daunting. Sleep statements were a work around for this problem. The connection problems were drastically reduced after finding out this solution.

8. Team member contribution
Below is the list of functionalities implemented in our version of hadoop framework.

Architecture Design: Kaushik
Client-Server communication: Kaushik, Habiba
S3 I/O Operations: Sudeep
Scripts: Neelesh
DataTypes: Aditya
Byte wise file division: Neelesh
Hadoop Module: Kaushik, Habiba
Debugging and issue resolution: Sudeep, Neelesh
File locations: Habiba, Sudeep
Report: Kaushik
Generics: Habiba, Kaushik
Changes to assignments: Aditya
Integration: Habiba, Kaushik, Sudeep
Testing: Aditya, Neelesh, Sudeep
Code clean up: Kaushik, Habiba

9. Test programs
- Did this test program require any new functionality to get to work?
- How did performance compare to Hadoop?
- Did you produce the expected output?

Program Name	New functionality?	Performance	Expected Output
Word count	Changes include: - Changing the Iterable to	About the same time as Hadoop	Yes

Word Median	CustomIterable<> class in reduce method - Changing enhanced for loop to while as it is not supported by the custom iterable class	About the same time as Hadoop	Yes
a2		About the same time as Hadoop	Yes
a5		Took more time compared to other programs	Yes
a7			Yes

10. Conclusion

Our version of Hadoop implementation supports all the required functionalities we used during this course. It has been tested using Word Count, Word Median, A2, A5 and A7 programs successfully. It has been a challenging project and all of us enjoyed working on this project. Adding to the extensive learning of Hadoop architecture, it has also been interesting exploring some of the unexplored Java concepts for us. This is the first time we have used Generics extensively and Reflection concepts to some extent. Its been a great learning experience with respect to core Java. This project also gave us extensive knowledge about S3 architecture.

We really loved the idea of implementing the Hadoop framework for the main project of this course. It gave us a sense of developing a widely used product on our own even if it has just few features.