Big Data Analysis Using Apache Hadoop and Java

Shamini Puthooppallil Baby

Masters in Data Science
Royal Melbourne Institute of Technology
Melbourne , Australia
shaminibaby000@gmail.com

***Abstract*—We live in Digital Universe on-demand, on-command, with information propagating at a very high rate by organizations, individuals and machines. Because of its sheer volume, variety and velocity, this data is labeled as "Big Data." Almost all of these data is unstructured, quasi-structured and semi-structured and in essence heterogeneous. The size and complexity of information with the speed produced makes it difficult to handle Big Data for the current computing infrastructure. It is tedious and time consuming to use conventional DBMS techniques such as joins and indexing and other techniques such as graph search. Map Reduce is a programming model for processing and generating large data sets and a related implementation. Users define a map function that processes a key / value pair to create a set of intermediate key / value pairs, and a reduction function which merges all intermediate values associated with the same intermediate key. This model represents most real world activities. This paper suggests various methods for optimizing the problems in hand through Map Reduce framework over Hadoop Distributed File System (HDFS) and Java. Map Reduce techniques have been studied in this study which is implemented for Big Data analysis using HDFS.**

Keywords—Map Reduce; Java; big data; Apache Hadoop; Amazon web service;

I. INTRODUCTION

With the amount of data increasing at an exponential rate, it is necessary to develop tools able to take use of that data and extracting value from it. When implementing any algorithm on large datasets, the need for large data analytics frameworks can be seen. Any such process in a local system uses a single core of the CPU. GPUs which contain multiple cores are increasingly being used to improve performance as the data size increases. Because of their distributed architecture, parallel processing can be easily done.

The primary purpose of this study is to efficiently handle data of a large magnitude.Resources and maintenance costs will change as required in a distributed data system and the proposed system is scalable enough to support this. The frame used is also smart enough to handle errors completely. We can implement a faster data retrieval and analysis if paired with the advantages of parallel processing. The problem of parallel processing could be overcome by using Apache Hadoop which is designed to boost the number of machines each offering local computation from a single server. This paper explores a more efficient and robust platforms, Apache Hadoop which is integrated with Amazon Web Services and Java development environments

II. RELATED WORK

Study of this literature start with the intention to analyze past work in these areas and understand their limitations in order to engineer a system capable of handling these limitations.

The architecture and utility of Apache Hadoop was presented to the community by the authors of [1]. It gives a brief overview regarding the programming model, which includes Java, parallel computing, Processing Efficiency etc. It also introduces a few implementations and testing in the environment.

Map Reduce framework and implementation using Big Data is introducing by the authors of [2].It carefully analyses the Problem solving ability and scalability of mapReduce and Apache hadoop.

Map Reduce techniques have been studied in [3] which is implemented for Big Data analysis using HDFS. It suggest various methods for catering to the problems in hand through Map Reduce framework over Hadoop Distributed File System (HDFS)

III. PROPOSED FRAMEWORK

A framework implementation on Amazon Web Services is proposed based on the MapReduce requirements; it is shown in Figure 1. This framework is designed to operate in a cloud computing environment[1].

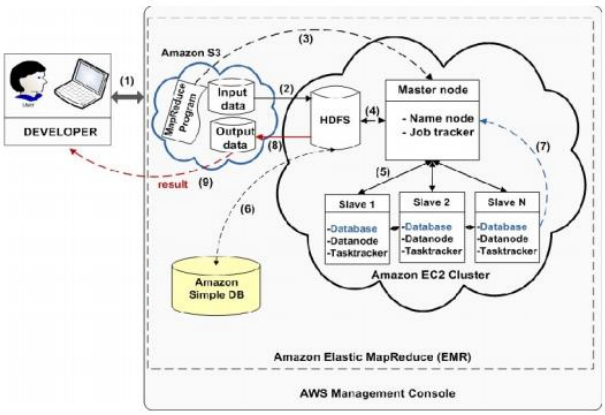

Fig. 1. Amazon Web Service running a MapReduce Program

Legends:

(1.) Connect to AWS console, upload MapReduce program and input dataset on Amazon S3 service.
(2.) Input file is uploaded into HDFS
(3.) MapReduce program is located at the master node.
(4.) Master node gets the input file from HDFS.
(5.) Master node distributes data blocks to slaves for processing.
(6.) Master node checks and updates the state of the running jobs.
(7.) Results from slaves are backed to master node and stored on HDFS.
(8.) Final output file is transmitted from HDFS to Amazon S3.
(9.) Final output file is transmitted from Amazon S3 to the local machine.

Figure 1 shows a working model of implementing the proposed framework in Amazon Web Services, for massive data processing. It shows how the Amazon Web Services uses cloud infrastructure to work with MapReduce for running a MapReduce program for large-scale data processing.

IV. EXPERIMENTATION

The overall aim of experimentation is to perform a qualitative and quantitative analysis on the performance of the framework proposed in this paper.A MapReduce program was written in Java to process a large dataset. The program determines the trip duration of taxis in New York city which travelled more than one hour [9]. The program was executed based on the proposed framework in a cloud computing environment hosted by Amazon Web Services. The proposed framework, which is one contribution of this work, was tested by running the MapReduce program on Amazon Elastic Compute Cloud (EC2) cluster. In the experiment, datasets which are comma separated text files with varying sizes from 660 MB to 1.5 GB, Were used independently and then combined to create a large data set of size 5 GB and stored on Amazon Simple Storage Service (S3) for being processed.

New York Taxi Data set is has the information of Taxi Number, Pick Up Time and Drop off Time. Once he data is feed to the mapper, it calculates the travel duration and creates a key value pair of <Taxi Number, Travel Duration>.This research is conducted with an implementation of petitioner and combiner in between mapper and reducer. Combiner does the same function of reducer that is, it filter out the records with travel duration more than one hour which is coming from each mapper .The output of the program gives the taxi number and corresponding travel time in a sorted order.

The experiment is conducted in two different aspects. Initially, the number of nodes made constant (10) and the input data size changed. Secondly, the same set of experiments repeated after increasing the nodes from 10 to 16 [4].

A. Processing Efficiency

The speed of processing depends on the how big is the scale of the Amazon EC2 cluster in a cloud environment. This experiment uses 3 different ways of process optimization

TABLE I. PROCESSING SPEED-I

No. of Records	Data Size	Time Taken(In min)
19808095	1.3 GB	4 min 8 sec
35878057	1.6GB	5min 42 sec
61144039	4 GB	8 min 53sec

a. No of Nodes :16,without Implementing Combiner and Reducer

TABLE II. PROCESSING SPEED-II

No. of Records	Data Size	Time Taken(In min)
19808095	1.3 GB	3 min 6 sec
35878057	1.6GB	3 min 32 sec
61144039	4 GB	6 min 7sec

a. No of Nodes :10,Implemeted combiner and Reducer

TABLE III. PROCESSING SPEED-III

No. of Records	Data Size	Time Taken(In min)
19808095	1.3 GB	1 min 2 sec
35878057	1.6GB	1 min 22 sec
61144039	4 GB	3 min 34sec

a. No of Nodes :16, Implemented combiner and Reducer

- Local Aggregation: Table I and Table III represent the test outcomes with same number of instances and nodes. Efficiency of the data retrieval increased dramatically when introduces local aggregation by adding combiner and custom practitioner in the input java program.
- Increasing nodes: Initially, the experiment was conducted without any task node. Eventually 10 task nodes added and it increased to 16 nodes. Table II and Table III compares the respective processing efficiency.
- Data size: The experiment repeated with 3 different input sizes starting from 1.3 GB to 5.3 GB.As the data size increases, the processing efficiency is falling down considerably.

B. Cost Efficiency

In Amazon EC2 cluster, we pay only for what we use by the hour. In this pay-as-you-go system, it allows users to lease services. This feature provides a cost-effective service for quick processing of large amounts of distributed data at minimal and affordable cost, especially if the dataset size is too big. We run a cost-effective Hadoop cluster in the AWS cloud, which can be expanded dynamically by leasing more instances. Because this experiment uses medium instances[1].

TABLE IV. AMAZON EC2 PRICING

No. of Instances	Data Size
2	$0.32
4	$0.64
6	$0.96
8	$1.28

V. CONCLUSIONS

This paper conducted a novel framework for the analysis of big data using Amazon Web Services and Hadoop Frame work. The proposed framework combined two widely popular tools, namely Apache Hadoop and Java, under different input conditions. This research helped to conduct big data analysis with higher accuracy from a new perspective using different optimization techniques in map reduce and Hadoop storage platform. Real time datasets from the Amazon Web Service storage is accessed and compared the efficiency of the application and data retrieval.

VI. SCOPE FOR FUTURE WORK

There were some serious challenges while conducting the research on this paper. One key challenge was using the framework for large datasets available in AWS. Since, the available set of clusters was limited, this paper narrow down the search area to the available dataset in the S3 store.

REFERENCES

[1] S. Daneshyar and M. Razmjoo, "*Large-Scale Data Processing Using Mapreduce In Cloud Computing Environment".* IEEE, 21-Oct.-2019.

[2] A. Gupta and H. K. Thakur, "*A Big Data Analysis Framework Using Apache Spark and Deep Learning*." IEEE, 11-Aug.-2019

[3] S. G. Manikandan and S. Ravi, *.Big Data Analysis Using Apache Hadoop*.IJARIAINT, 21-Sep.-2018

[4] *AWS*. [Online]. Available: https://aws.amazon.com/. [Accessed: 1-Oct-2019].

[5] *Advanced MapReduce*. [Online]. Available: http://hadooptutorials.co.in/tutorials/mapreduce/advanced-map-reduce-examples-2.html. [Accessed: 16-Oct-2019].

[6] *CodeJava*. [Online].Available: https://www.codejava.net/coding/how-to-configure-log4j-as-logging-mechanism-in-java. [Accessed: 16-Oct-2019]

[7] *GitHub*.[Online].Available:https://github.com/deshpandetanmay/mapreduce-examples/blob/master/trunk/cdr-analytics/CDRAnalytics/src/in/co/hadoop/tutorials/STDSubscribers.java. [Accessed: 10-Aug-2019

[8] *blog*. [Online]. Available: https://www.loggly.com/blog/an-introduction-to-amazon-elastic-mapreduce-emr-logging/. [Accessed: 16-Oct-2019].

[9] *New York Taxi*. [Online]. Available: https://data.cityofnewyork.us/Transportation/2018-Yellow-Taxi-Trip-Data/t29m-gskq. [Accessed: 16-Oct-2019].

SUPPORTING DOCUMENTS

The supportig documents for the experiment is listing below: All of these documents are available at:

https://drive.google.com/file/d/1EaoTEZlSfGWmow44Xt5v7C7xBMsifon7/view?usp=sharing

[1] Sample source Data
[2] Java Program code
[3] Runnable Jar File
[4] Sample utput File
[5] Log details
[6] Program execution flow