DOI: http://dx.doi.org/10.26483/ijarcs.v8i7.4256

ISSN No. 0976-5697

Volume 8, No. 7, July – August 2017

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

EFFICIENT PROCESSING OF JOB BY ENHANCING HADOOP MAPREDUCE FRAMEWORK

Ayesha Saad
Pursuing M.tech, Dept. Of Computer Science and Engineering
Integral University
Lucknow,India

M. Akheela Khanum
HOD,Dept.
Of Computer Science and Engineering
Integral University
Lucknow, India

Abstract: Cloud Computing uses Hadoop framework for processing BigData in parallel. The Hadoop Map Reduce programming paradigm used in the context of Big Data, is one of the popular approaches that abstract the characterstics of parallel and distributed computing which comes off as a solution to Big Data. Improving performance of Map Reduce is a major concern as it affects the energy efficiency. Improving the energy efficiency of Map Reduce will have significant impact on energy savings for data centers. There are many parameters that influence the performance of Map Reduce . Various parameters like scheduling, resource allocation and data flow have a significant impact on Map Reduce performance. Cloud Computing leverages Hadoop framework for processing BigData in parallel. Hadoop has certain limitations that could be exploited to execute the job efficiently. Efficient resource allocation remains a challenge in Cloud Computing MapReduce platforms. We propose a methodology which is an enhanced Hadoop architecture that reduces the computation cost associated with BigData analysis.

Keywords: Big Data, Data flow, Energy Efficiency, Map Reduce, Performance, Resource Allocation, Scheduling

INTRODUCTION

In Cloud Computing the concept of parallel processing has come out as a versatile research area as data is highly voluminous and varied. To process and understand such an enormous amount of data traditional processing methods cannot be used. There is now an availability of freely available and marketable cloud computing parallel processing platforms that have paved the way to process structured, semi-structured or unstructured data [1]. We first define what actually is BigData and Hadoop and some terms related to it.

BIG DATA CONCEPTS

The term “Big Data” refers to large voluminous data sets that is composed of a wide range of structured data as well as unstructured data which can be too big, produced at a fast rate and thus being difficult to be managed by traditional techniques.

BigData can be either a relational database that is structured, such as stock market data or non-relational database that is semistructured or unstructured, like social media data [2].

BigData is catagerized by 4V’s 1) Volume of the data, which means the data quantity 2) Velocity, which means the pace at which the data is generated 3) Variety of the data, which means the different forms of data that applications have to deal with such as numeric data or binary data. 4) Veracity of the data, which means the uncertain or imprecise data that provide meaningful information to the applications[3].

There are many challenges in Big Data[4] which can be described as technical challenges such as the challenge of storing the BigData and also minimizing redundancy present in it. Also many challenges like data integration, representation of data and data cleaning are tedious tasks because of such high volume of data. Because of these issues, BigData needs such an environment or framework that can help to work through these issues smoothly. One of such framewok called Hadoop, which works with BigData sets, are now used by most organizations to process BigData, and helps overcoming theses challenges.

Hadoop is a framework responsible for distributed storage and distributed processing, it is a freely available software framework written in java by Apache Software Foundation. It provides a file system that acts as an interface between the users’ applications and the local file system, known as Hadoop Distributed File System HDFS. The two main components of Hadoop are (i) Hadoop Distributed File System (HDFS) allowing for distributed storage (ii) MapReduce that allows for distributed processing [5] [6].

Hadoop works by dividing the data into blocks where block size is defined beforehand .The blocks will then be written with the data and replicated in the HDFS. The blocks can be replicated ato a certain fixed value based set to 3 by default [7]. HDFS is one of the major components of Hadoop cluster and HDFS is designed to have Master-slave architecture. The Master node which is called NameNode and the slaves called DataNodes. The Name Node manages and organizes data storage capacity (HDFS), A particular NameNode manages the file system and perform tasks like saving the data, it also assigns the jobs to the suitable DataNodes that stores the application data required by the job[8]. DataNodes facilitate MapReduce to process the jobs in a parallel processing environment [5, 9].

Map Reduce brings compute to the data, which is significantly cheaper than moving data towards computation. MapReduce is now heading as a standard tool[4] as it offers immense storage power and substantially high amount of computing power. Map Reduce also possess a Master-slave architecture. The Master here is Job Tracker and slaves here are the many Task Trackers.

Master that is the Jobtrackers is the point of interaction between users and the map/reduce framework. JobTracker coordinates and deploys the applications to the DataNodes with TaskTracker services for execution and parallel processing [7]. The Job Tracker puts the MapReduce job in a queue of pending jobs, executing them on a FCFS basis, and then assigns the map tasks and reduce tasks to the tasktrackers. Slaves that is the tasktrackes will now execute tasks upon instruction from the Master that is the Jobtracker and will be handling movement of data between the map and reduce phases. Thus, the master computer has two daemons, which are NameNode in terms of HDFS and JobTracker in terms of MapReduce. Similarly, the slaves also have two daemons, which are DataNodes in terms of HDFS and TaskTrackers in terms of MapReduce.

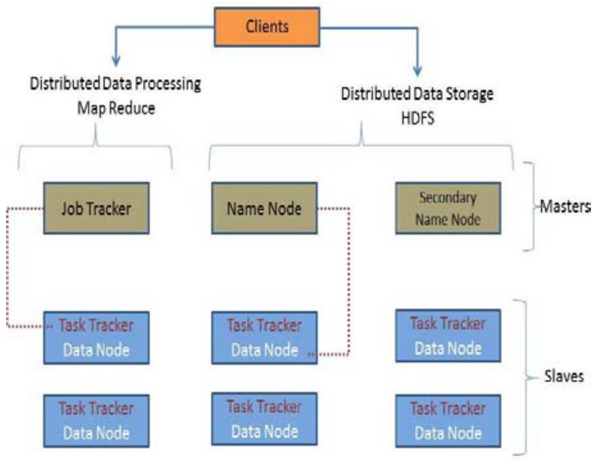

Figure 1: Concept of Master and Slaves in Hadoop MapReduce

A MapReduce job works by splitting the present input dataset into autonomous chunks that is blocks and stores them in HDFS. In the process of MapReduce, the numerous Map tasks are processed in parallel which will be followed by many Reduce tasks that is also processed in parallel. The number of maps and reduce tasks performed are application dependent and their number can vary for every application processed. Also the number of map task and reduce task are not same and be different from each other. In HDFS data can be stored in different forms [3] such as in the form of <Key,Value>, where the Key is determined and then the value of Key is resolved at the end of Job.

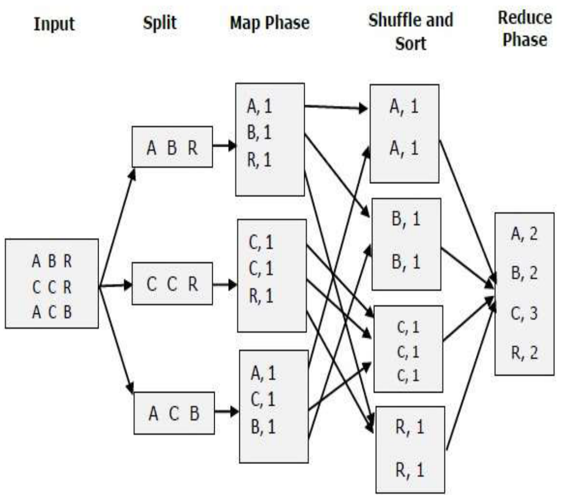

Figure 2: Example of MapReduce Processing

We try to explain MapReduce with the help of a simple example -One of the first phase is the Input Phase where a Record Reader transforms each record in an input file and forwards it to the map phase, where it is present in the form of key-value pairs. Now the next phase that is the Map phase is a user-defined function, which works by taking a sequence of key-value pairs and then processing each sequence to generate zero or more key-value pairs. These key-value pairs generated by the map phase are called intermediate keys. Then comes the wok of a combiner which is a type of local Reducer that will take similar data from the map phase and group it into exclusive sets. It is not necessarily a part of the main MapReduce algorithm and is optional. The other stage is of Shuffle and Sort. The task of Reducer starts with this very step. It copies the grouped key-value pairs onto the local machine, it is where the reducer is running. The specific key-value pairs are sorted by key into a larger data list. The data list groups the alike keys jointly so that the reduce operation is performed easily. The final main task of Reducer is that it takes the alike key-value pairs as input and finally runs a Reducer function on each pair. Now the data can be combined, categorized and filtered. After this phase is over, zero or more key-value pairs are generated to proceed towards the final step. Finally, an output formatter translates the final key-value pairs taken from the Reducer function and incorporates them onto a file using a record writer. The NameNode provides the result that has all keys and their values as the ultimate and final result.

In the following text, an overview of existing Hadoop MapReduce workflow is described and the limitations in terms of performance of MapReduce is focused upon. The next section discusses the issue that our proposed work tries to solve. Then, we have proposed an enhanced workflow for Hadoop MapReduce. In the next section, the implementation and testing phase is discussed, and the results are evaluated and enhancement of parameters achieved are conferred. Finally the the work is concluded.

NATIVE HADOOP WORKFLOW

In the existing Hadoop MapReduce architecture, a job that is provided by the client is sent using a Query language like Hive or by creating a job source code[11] to the Name Node.

Blocks of equal size usually 64 or 128 MB of BigData are uploaded to the HDFS and are distributed among different DataNodes within the cluster. So any job that is ready for execution should have the name of the data file in HDFS, the source code file of MapReduce job that is in java, the name of the file where the final reduced results will be stored in.

In the present Hadoop MapReduce architecture, if for multiple jobs we have the same dataset, it will execute completely independent of each other. For example if any job is executed once and takes a certain amount of time for execution then if another job of same nature, that is a similar job is given by the client, it will take the same amount of time as was done by the earlier job execution.

Here we try to explain the workflow of a native Hadoop MapReduce:

Firstly, a Client "A" sends a request for job execution to the NameNode. This request will handle the copying of data files to DataNodes. The NameNode will reply with the IP address of DataNodes. Now the Client "A" will access the raw data and format it into HDFS format while dividing data blocks of fixed size.

This data block will be send to different DataNodes in form of multiple copies usually '3' is the replication factor here.

Now the Client "A" will send a MapReduce job to the JobTracker which in turn sends it to all the TaskTrackers holding the blocks of data. The TaskTracker will execute the specific task on each data block and sends final result back to the JobTracker. The JobTracker sends the final result to Client "A". If another Client"B" comes up with a similar job again the processing of job is done again using MapReduce at the TaskTrackers, and take almost similar time as the earlier executed job of Client"A".

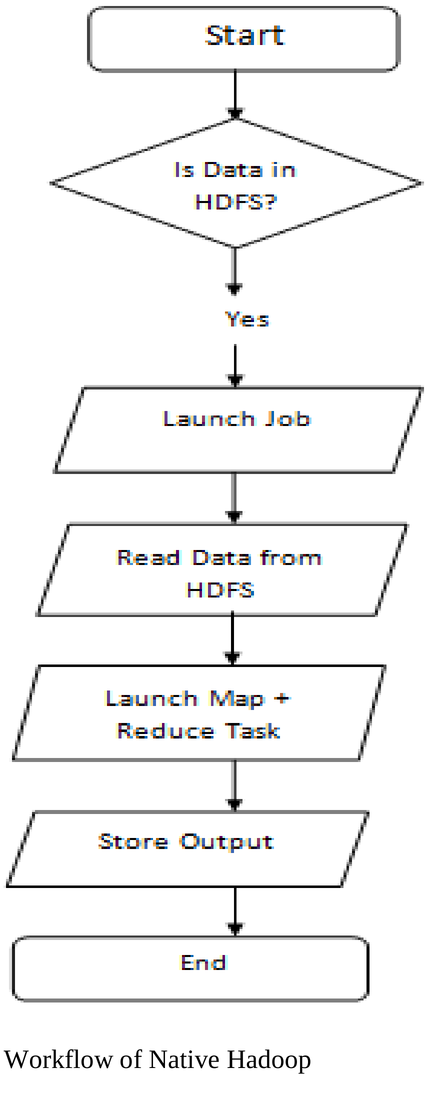

Figure 3: Workflow of Native Hadoop

A. *Native Hadoop Map Reduce Limitations*

One of the limitations we can observe here is that the job execution is independent of each other. So, every job has to go through the same process of execution even if it carries the same task to be performed as that of earlier jobs. If we have the same job executed more than once, it will read all the data every time; which may lead to declining of Hadoop performance.

B. *Research Problem*

In existing Hadoop architecture, as a new job arrives it is sent by the job tracker to all the task trackers. Task Trackers execute the Map-Reduce task and return results to Job Tracker which gives it back to client. If same job comes again then the same steps are repeated again. In our proposed work, if a job has already been executed ensured by checking in the common job table then the job does not undergoes Map Reduce Processing again but the results are fetched that have already been executed and thus it saves the execution time, as we don't have to re-execute the job. The proposed system has been explained as follows.

COMMON JOB TABLE

The proposed work is almost same as the original Hadoop processing of job however some features have been added like the common job table that contains name of the already executed jobs.

The Common Job Table stores information about jobs. The job's common job name, Input path, and the Output path. The Common Job table has been built using Hive. This database forms the Common Job Table where we will save the data of all the previous jobs processed and this will help us in finding the already run jobs.

Proposed work depends on Common Job Table for efficient data analysis. Each time a new job is executed it's job name, Input path and Output path is saved in the Hive Common Job Table. Common features in the table can be compared and updated every time a client submits a new job in Hadoop.

Job Tracker will direct any new job with the common features to common job table. Now if suppose there are two jobs J1 and J2. J1 employs Map Reduce processing to process the job. If J2 contains common features of J1, the output is mapped onto J2 also and Map Reduce processing is not performed again.

If the Common Job Table is empty, the user will execute the Map Reduce job in a traditional way without getting the benefits of proposed solution.

Size of our common job table can be limited using 'Leaky Bucket' or 'Token Bucket' algorithm, which can become a topic of future discussion.

PROPOSED HADOOP MAPREDUCE WORKFLOW

Enhanced Hadoop architecture does not clash with the already existing Hadoop architecture but there are some enhancements of only the software level through or table containing the common jobs. The workflow of our proposed system has been explained as:

Firstly, a Client "A" sends a request for job execution to the NameNode. This request will handle the copying of data files to DataNodes. The NameNode will reply with the IP address of DataNodes. Now the Client "A" will access the raw data and format it into HDFS format while dividing data blocks of fixed size.

This data block will be send to different DataNodes in form of multiple copies usually '3' is the replication factor here.Now the Client "A" will send a MapReduce job to the JobTracker which in turn sends it to all the TaskTrackers holding the blocks of data. The TaskTracker will execute the specific task on each data block and sends final result back to the JobTracker, the results are returned to the client and in turn JobTracker keeps the details of the job; its Jar file Name, Input

Path, Output Path in the Common Job Table of Hive.Then, if another Client"B" submits a new MapReduce Job "Job2" to JobTracker with the same Jar File Name and Input Path as "Job1" which is found out when JobTracker checks Common Job Table. Then, MapReduce processing is not performed again and this output is fetched directly from Output Path. The result is sent back to the Client.

There is some training data present before executing the process of MapReduce so as to keep some metadata in the table of common jobs so as to observe the advantages of the new architecture.A relationship between processing of jobs, by our proposed system is made with the already existing solution and the proposed methodology comes out to be better.The flowchart given below explains the above mentioned steps of our proposed system.

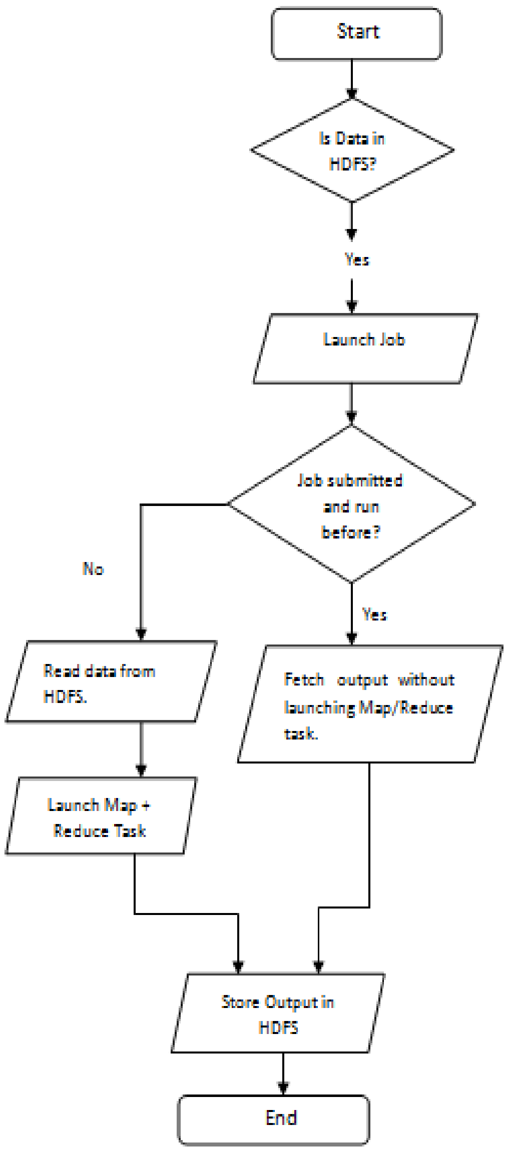

Figure 4: Workflow of Proposed Hadoop MapReduce

IMPLEMENTATION AND TESTING

In, this section we have an implementation plan for the proposed solution and its expected results.

C. Creating Common Job Table

We can create the Common Job Table using many different techniques. One of the techniques is Hive, that gives a SQL-like interface that helps in querying data stored in databases and file systems that are integrated with Hadoop. It allows for portability of SQL-based application to Hadoop. It uses a SQL like query language called HiveQL.

D. Designing User Interface

As we proposed earlier the user interface should contain user-friendly interface so that the user is receive the benefits of the enhanced design when choosing common data from lists. For example, when choosing the Common Job Name from a list of common job names that are related to the similar data files. Different forms of user interfaces can be designed based on the user's needs. One of the common user interfaces is, the command line that is commonly used when the user knows the commands and the related parameters they will use. Hadoop and Hive are controlled by the same command line, which is a shell command line in Linux. Therefore, in our work, we use the shell command line as a user interface to implement the proposed solution. The commands that are used here are the same original Hadoops' commands.

E. MapReduce Job

We executed a Job written in Java. The Job finds the maximum and minimum temperature for the day. The data was collected for 6 months. A more detailed and enormous can be worked upon in the proposed system as a future prospect.

F. Units

We have Hadoop and Hive running on shell interface of linux. Following applications are used:

- We have one Master node, that serves as the NameNode and The JobTracker.
- We have 2 slave nodes that will serve as DataNodes and TaskTrackers in different locations.
- Linux Ubuntu version 16.04 as an operating sytem on all nodes.

We executed some MapReduce Jobs. Our first Job involved temperature Input data. To check the authenticity of our work we executed the same MapReduce Job the next time. The proposed work did not perform MapReduce proceeing again as the job was already executed. Another Job with same work but with different Job Name and Input Path was saved and executed and it performed MapReduce processing again to execute the Job. Thus, we observed if same jobs are arriving again and again at Job Tracker then MapReduce processing is not performed everytime.

RESULTS AND EVALUATION

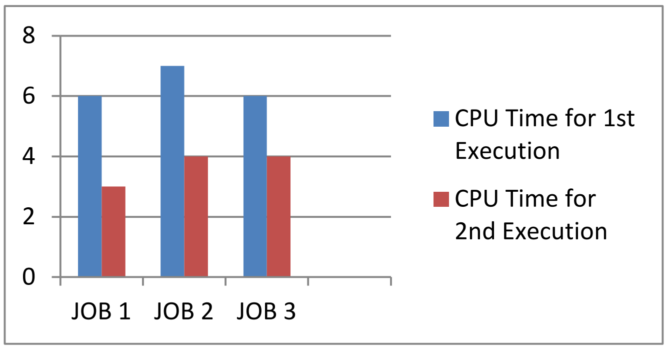

Figure 5: CPU Time for Job Execution

Figure 5 shows the difference of CPU Time taken by each Job (3 Jobs taken) when run for the first time using our proposed system that simply works like normal Hadoop framework and when another similar Job comes and is run and processed using our proposed system , it takes much less CPU time than before. As it does not perform MapReduce operation again and again.

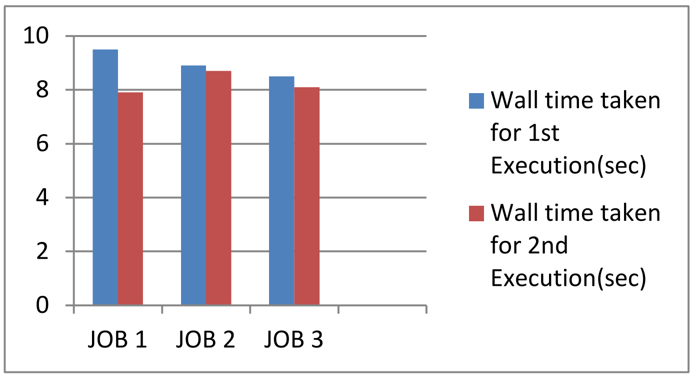

Figure 6: Wall Time for Job Execution

The another parameter considered to compare the performance is Wall Time for Jobs executed using our proposed system. As the above figure shows the Wall time for the execution of Job run using proposed architecture for first time and then second time observes that Wall time Using Common Job table requires less time than native Hadoop Architecture.

CONCLUSION

The proposed work presented here focuses on the software aspect of Native Hadoop Architecture and brings out some enhancements in the same. The work allows for the NameNode to identify the already run Jobs using a Common Job Table to store the Jobs with common name and Input Path and saving the time spent in MapReduce processing of similar Jobs by fetching the already present result. The proposed architecture's workflow was discussed and the time taken by first execution where MapReduce processing has been performed to the execution of similar Job arriving where MapReduce is not performed again. Thus achieving significant gain in Hadoop performance.

REFERENCES

[1] Eugen Feller,Lavanya Ramakrishnan,Christine Morin," Performance and energy efficiency of big data applications in cloud environments: A Hadoop case study", Journal of Parallel and Distributed Computing,Elsevier (2015)

[2] Mukhtaj Khan, Yong Jin, Maozhen Li, Yang Xiang and Changjun Jiang, "Hadoop Performance Modeling for Job Estimation and Resource Provisioning", IEEE Transactions on Parallel and Distributed Systems.

[3] Javier Conejero, Omer Rana, Peter Burnap, Jeffrey Morgan, Blanca Caminero, Carmen Carrión," Analyzing Hadoop power consumption and impact on application QoS", Future Generation Computer Systems 55 (2016)

[4] Jacob Leverich, Christos Kozyrakis" On the energy (in)efficiency of Hadoop clusters", Volume 44 Issue 1,January2010, Pages61-65 ,ACM New York, NY, USA

[5] Rini T. Kaushik, Milind Bhandarkar" GreenHDFS: Towards An Energy-Conserving, Storage-Efficient, Hybrid Hadoop Compute Cluster", HotPower'10 Proceedings of the 2010 international conference on Power aware computing and systems, Article No. 1-9, Vancouver, BC, Canada

[6] Yanpei Chen, Archana Ganapathi" GreenHDFS: Towards An Energy-Conserving, Storage-Efficient, Hybrid Hadoop Compute Cluster", HotPower'10 Proceedings of the 2010 international conference on Power aware computing and systems, Article No. 1-9, Vancouver, BC, Canada

[7] Zhuo Tang, Lingang Jiang, Junging Zhou, Kenli Li, Keqin Li "A self-adaptive scheduling algorithm for reduce start time", Future Generation Computer System, Volumes 43–44, Pages 51–60 (2015)

[8] Weikuan Yu, Yandong Wang, Xinyu Que, Cong Xu "Virtual Shuffling for Efficient Data Movement in MapReduce", IEEE Transactions on Computers, Volume: 64, Issue: 2 (2015)

[9] Kumar KA, Konishetty VK, Voruganti K, Rao GVP. CASH: Context Aware Scheduler for Hadoop. In: Proceedings of the International Conference on Advances in Computing, Communications and Informatics. New

[10] CloudSuite 1.0, Web page at

[11] http://parsa.epfl.ch/cloudsuite/cloudsuite.html (Last access: 26.06.14).

[12] Tian C, Zhou H, He Y, Zha L. A dynamic MapReduce scheduler for heterogeneous workloads. In: 8th International Conference on Grid and Cooperative Computing. 2009. p. 218–24.