ISSN (e): 2250 – 3005 || Volume, 07 || Issue, 07|| July – 2017 ||
International Journal of Computational Engineering Research (IJCER)

On The Fly Mapreduce Aggregation for Big Data Processing In Hadoop Environment

Ms. Rucha S Shelodkar[1], Prof. Himanshu U Joshi[2]

[1] Department of Computer Engineering JSPMs Imperial College of Engineering & Research, Wagholi, Pune, India,

[2] Department of Computer Engineering JSPMs Imperial College of Engineering & Research, Wagholi, Pune, India,

ABSTRACT

Big Data is term that refers to data sets whose size (volume), complexity (variability), and rate of growth (velocity) make them difficult to capture, manage, process or analyzed. To analyze this enormous amount of data Hadoop can be used. Hadoop is an open source software project that enables the distributed processing of large data sets across clusters of com-modity servers. I proposed a modified MapReduce architecture that allows data to be pipelined between operators. This reduces completion times and improves system utilization for batch jobs as well. I present a modified version of the Hadoop MapReduce framework that supports on the fly aggregation, which allows users to see early returns from a job as it is being computed. The objective of the proposed technique is to significantly improve the performance of Hadoop MapReduce for efficient Big Data processing.

Keywords: *Big Data, Combiners, Hadoop Framework, Multiplexer, On the fly Aggregation, Performance Model, System Architecture.*

I. INTRODUCTION

Over the past several years there has been a tremendous increase in the amount of data being transferred between Internet users [1]. The Big Data is a collection of large datasets that cannot be processed using traditional database management tools.

Big Data is the combination of structured, semi-structured, unstructured, homogeneous and heterogeneous data. The large amount of data is produced that can be structured and un-structured from the different sources. Such type of data is very difficult to process that contains the billions records of millions people information that includes the web sales, social media, audios, images and so on. Big data comes from the Big Companies like yahoo, Google, Facebook etc for the purpose of analysis of big amount of data.

Big data has four main characteristics as volume, velocity, variety, and value .Big data has Various challenges are such as privacy and security, access and sharing information, data management. Traditional method and technical challenge to compensate the needs of storing big data as they have high failure rates and replacement costs and less scalability. Hadoop, Google File System, No SQL and New SQL data stores have been used widely for big data storage.

This paper presents an early result estimation technique. I have proposed a simple on the fly aggregation technique implementation, which will significantly improve the performance of Hadoop framework. The main objectives from the proposed system are implementation of on the fly aggregation taking to provide the user to see early returns from a job.

The paper is organized as Section II as related work and Section III necessary background respectively. In section IV, I have discussed about motivation of the work. In section V, discussed with proposed system architecture along with mathematical model and multiplexer logic is discussed. In section VI discuss with experimental setup and results are given. Finally Section VII paper final conclusion is given.

II. RELATED WORK

1) S. Vikram Phaneendra & E. Madhusudhan Reddy[2] Illustrated that in olden days RDBMS the data was less and easily handled but Now it is difficult to handle huge data through RDBMS tools, that is called as big data. Hadoop architecture handle large data sets, scalable algorithm does log management application of big data.

2) Chris Jermaine proposes an on the fly Aggregation for Large-Scale Computing. They are interested on very large scale and data-oriented computing.

3) The EARL library [3] is an extension to the Hadoop framework and focuses on accurate estimation of final results, while providing reliable error estimations. It uses the bootstrap technique, which is applied to a single pre-computed sample.
4) Vasiliki Kalavri,Vaidas Brundza,Vladimir Vlassov [6] Here the block-level sampling technique is used to provides a random sample of the provided dataset. And provides integrated block-level sampling with the MapReduce On the fly framework is done. System provides accurate estimations of results, without requiring prior knowledge of the query.
5) Tyson Condie, Neil Conway, Peter Alvaro, Joseph M. Hellerstein [5] - The system proposes a modified version of the Hadoop MapReduce framework that supports on the fly aggregation and that allows users to see early returns from a job as it is being computed
6) The BlinkDB [7] approximate creates query engine and uses pre-computed samples of various sizes to provide fast answers. It has two types of samples: large uniform random and smaller multi-dimensional stratified. Queries are evaluated with on a number of selected samples and an initial estimate is produced and the system can give estimate results of standard aggregate queries easily.

III. BACKGROUND

In this section, a briefly review of the Map Reduce programming model, the MapReduce On the fly framework and the On the fly Aggregation technique is given. Hadoop is an Apache open source framework written in Java that allows distributed processing of large dataset across cluster of computers using simple programming model. Hadoop is a framework that can run applications on systems with thousands of nodes and terabytes. Allows to distributes the file among the nodes and allows to system continue work in case of a node failure. Fig1.shows MapReduce Word Count Example:

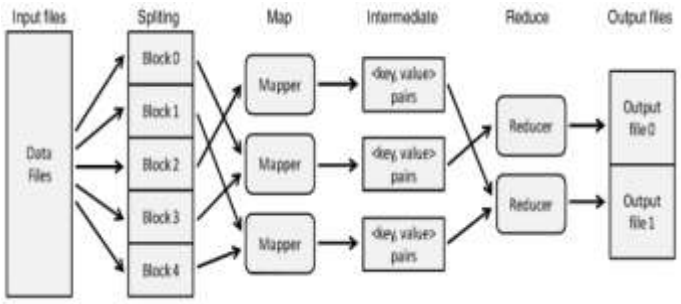

Figure 1. Hadoop mapreduce famework

3.1 Hadoop Mapreduce Framework

Mapreduce [1] [2] programming model is used for processing Big Data and It has two functions: Map and Reduce. Map tasks are apply on input data and which is partitioned into fixed sized blocks and producing the intermediate result as a output that is collection of key, value pairs. These pairs are shuffled across different reduce tasks based on key, value pairs. In Reduce task has accepts only one key at a time and process data for that key and outputs is also in the form the results as key, value pairs.

3.2 Mapreduce On The Fly

The Map Reduce On the fly is a simple freamework for Hadoop and MapReduce is a popular open-source implementation to supports On the fly Aggregation and stream processing, also im-proving utilization and reducing response time [3]. Traditional Map Reduce technique used to materialize the intermediate results of mappers and do not allow pipelining between the map and the reduce phases .In MapReduce On the fly technique has the advantage of simple recovery in the case of failures and In Traditional technique reducers cannot start executing tasks before all mappers have finished. To overcome this problem Map Reduce On the fly technique can be used by allowing pipelining between operators with preserving fault-tolerance guarantees.

3.3 On The Fly Aggregation

On the fly Aggregation [4] is a framework that used idea of batch mode to obtained approximate result .Map Reduce can be implemented using On the fly Aggregation. On the fly Aggregation technique explain partial query processing and without knowing previous output of the query specifications, such as types of operators and data structures and also, users can observe the progress of running queries and control their execution.

IV. MOTIVATION

In Big Data processing, main problems is that good performance requires both good data locality and good resource utilization. A characteristic of Big Data processing is that the amount of data being processed is typically large with compared to the amount of computation done on it. The processing can benefit from data locality, which can be achieved by moving the computation close to the data using Hadoop. Still there are few challenges with Hadoop too which are discussed in introduction. MapReduce is supposed to be for batch processing and not for on the fly transactions. The data from a MapReduce Job can be fed to a system for on the fly processing.

The current Hadoop MapReduce implementation does not provide such capabilities. As a result, for each MapReduce job, a customer has to assign a dedicated cluster to run that particular application, one at a time. Each cluster is dedicated to a single MapReduce application so if a user has multiple applications, the he has to run them in serial on that same resource or buy another cluster for the additional application. This gives the motivation to work towards improving the performance of Hadoop by implementing on the fly aggregation technique in Hadoop MapReduce framework.

V. IMPLEMENTED SYSTEM

In this Section, I have described the aggregation technique in detail. In on the fly aggregation, a database system processes a user's aggregation query in an on the fly fashion. At all times during processing, the system gives the user an estimate of the final query result, with the confidence bounds that become tighter over time. In this paper, we consider how on the fly aggregation can be built into a MapReduce system for largescale data processing.

The MapReduce On the fly supports On the fly Aggregation and reduces response time. Traditional Map Reduce implementations materialize the intermediate results of mapper and do not allow pipelining between the map and the reduce phases. In this approach has the advantage of simple recovery in the case of failures and reducers cannot start executing tasks before all mapper have finished. That has limitation of lowers resource utilization and leads to inefficient execution for many applications. The main motivation of Map Reduce On the fly is to overcome these problems, by allowing pipelining between operators, while preserving.

5.1 System Architecture

When execution of a map task is done reduce task collect outputs by giving request to node. The reduce task does not issue this request until it has received notification that the map task has completed and its final output has been committed to disk. This means that map task execution is completely decoupled from reduce task execution. In modified map task paradigm map will push data to reducer as it is produced. Consider a job in that reducer has some different values and then the reducer applies aggregate function. Combiners allow map-side pre aggregation: by applying a reduce-like function to each distinct key at the mapper. Because this is network traffic can also be reduced Fig.2. Shows the proposed system architecture.

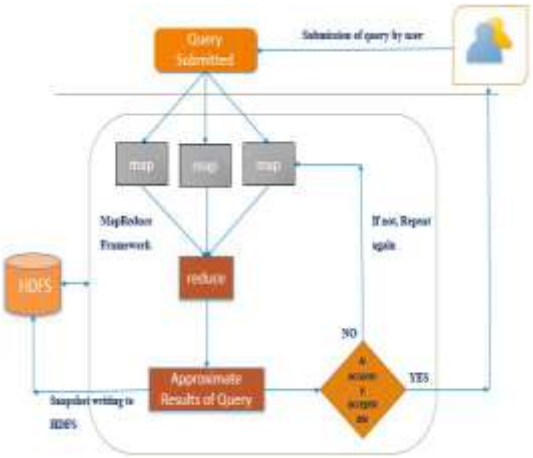

Figure 2. Architecture for on the fly aggregation of mapreduce

5.1.1 Practitioner and Combiners

Combiner can be used for aggregation .Combiner is a general mechanism to reduce the amount of intermediate data. It is thought as mini reducer. Mini reducer that processes the output of mappers. The difference between a reduce function and a combiner function is that the MapReduce library handles the output of the function. The output of a reduce function is written to the final output file. The output of a combiner function is written to an intermediate that will be sent to a reduce task. In word count example , the combiners aggregate term counts on

data processed by each map task. This results in a reduction in the number of intermediate key-value pairs that need to be shuffled across the network. An associative array (i.e., Map in Java) is introduced inside the mapper. This array tally up term counts within a single document. This version gives a key value pair for each unique term in the data instead of a key value pair for each term in the data being processed. For the same example we can consider some words appear frequently within a document. This method can save the number of intermediate key-value pairs emitted, especially for huge data. Combiner can be used for aggregation .Combiner is a general mechanism to reduce the amount of intermediate data. It is thought as mini reducer. Mini reducer that processes the output of mappers. The difference between a reduce function and a combiner function is that the MapReduce library handles the output of the function. The output of a reduce function is written to the final output file. The output of a combiner function is written to an intermediate that will be sent to a reduce task. In word count example, the combiners aggregate term counts on data processed by each map task.
This results in a reduction in the number of intermediate key-value pairs that need to be shuffled across the network. An associative array (i.e., Map in Java) is introduced inside the mapper. This array tally up term counts within a single document. This version gives a key value pair for each unique term in the data instead of a key value pair for each term in the data being processed. For the same example we can consider some words appear frequently within a document. This method can save the number of intermediate key-value pairs emitted, especially for huge data.

5.1.2 Taking Snapshots

The output of intermediate reduce operation is called as snapshot .The reducer will process the data given by mapper as input and generates the results. This result is considered as snapshot and will be shown to user .If result is acceptable then user can stop further processing of map reduce .If result is not acceptable then mapper and reducer will continue with the processing.

5.2 Algorithms and Mathematical Model

This section describes mathematical modeling of the pro-posed work. Turing machine is used to describe the processing of the system.

5.2.1 Algorithms for Mapper & Reducer

There are two main methods which actually carried outs the result, Map and Reduce. While processing the data number of instances of these methods runs in parallel. So, only Map and Reduce methods are analyzed as a analysis of complete system to find out which class the problem belongs to i.e. if it is P, NP, NP–complete, NP-Hard problem.
Combiner provides a general mechanism within the MapReduce framework to reduce the amount of intermediate data generated by the mappers. The combiner can be understood as mini-reducers that process the output of mappers. The combiner aggregate term counts across the documents processed by each map task. These results in a reduction in the number of intermediate key-value pairs that need to be shuffled across the networkfrom the order of total number of terms in the collection to the order of the number of unique terms in the collection. Algorithm for word count Mapper using Combiner and for Reducer are given above (Algorithm 1 & 2)[3].

Algorithm1: Algorithm for Word Count Mapper using the Combiner [3]

Input: Split Data
Output: Count of words for respective splits
1: class Mapper
2: method Initialize
3: H: new Associative Array
4: method Map (docid a,doc d)
5: for all term t doc d do
6: H (t) H (t) + 1
7: end for
8: method close
9: for all term t from H do
10: Emit (term t,count H(t))
11: end for

Algorithm2: Algorithm for Word Count Reducer [3]

Input: Intermediate results
Output: Final count of words
1: class Reducer

```
2:      method Reduce (term t, counts (c1, c2 ...))
3:      sum = 0
4:      for all count c from counts (c1, c2 ...) do
5:      sum      sum + c
6:      end for
7:      Emit (term t,count sum)
```

5.2.2 Multiplexer Logic

In addition to mathematical model, the multiplexer logic also discussed here. As the fig.3.indicates, the data from the HDFS is given to mapper as input. The mapper processes the data and the result is then sent to the reducers. The number of mappers and the reducers will be depending on the programming logic used and the number blocks to be processed. As figure indicates, the result of reducers is written to the HDFS and same is then sent to the mapper for further processing. The whole process is executed continuously until the desired output is not achieved. The difference between the traditional Hadoop and the proposed system is that, the intermediate results of mapper are forcefully sent to the reducers to obtain the early results. Where as in traditional system the reducer cannot start until the mapper finishes it processing completely.

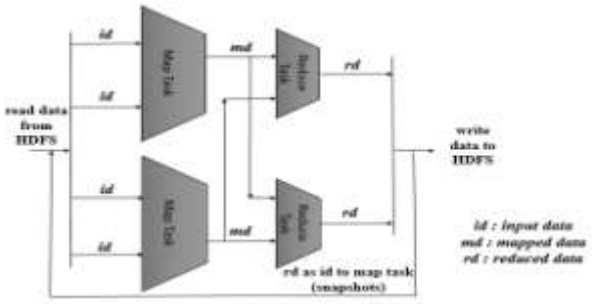

Figure 3. Multiplexer logic for aggregated mapreduce framework

5.2.3 Performance Model

The time taken for a Hadoop to execute a job on given data set can be described as

$T = ov + (ps+pa)\ h$ (1)

Where, **ov** = the fixed overhead in running a job **ps** = processing time for Hadoop system **pa** = processing time in map and reduce **h** = time over which data is collected. Here, the dynamic part of the processing time is divided into in to two parts, one part related to Hadoop system and the other is with the application logic. There is a fixed overhead cost in running any Hadoop job, which is represented by ov. The overhead is mostly associated with the work involved in job. It is independent of the data size. The dynamic part of the time has two components. The first component represented by ps, contains all the time taken by Hadoop system for processing a job. This part is mostly disk and network IO bound. Bradley speaking it constitutes the following:

a) Input and output for map
b) Shuffle
c) Input and output for reduce

The second component represented by pa. It is the time taken by the application logic i.e. time spent in the map and reduces functions. It contains the following parts:

a) Execution of map
b) Execution of reduce

The overhead is fixed and not much can be done about it. The focus is on the dynamic part of the time. We can get the best performance when pa dominates over **ov** and **ps**.

VI. EXPERIMENTAL RESULT

The results obtained from the traditional Hadoop framework and implemented proposed systems are taken on different sets of data. Both results are then compared to find out the conclusion. For both traditional and proposed system the same environment is used. The setup used for Hadoop and the results obtained are discussed below.

6.1 Environment

The figure 4 describes the experimental setup of proposed system. The hadoop-1.2.1 is used for setup. Ubuntu 12.04 operating system is installed on all the nodes. Total 3 nodes are configured, one as master node and remaining two as slave nodes. The processors and memory allocated to the nodes is given in table 1.

	Description	Total Virtual Machines
Machine	Processor: Intel Core i5-4210U CPU @ 1.70 GHz 2.40 GHz RAM: 4.00 GB	1- Master 2- Slaves
Hadoop Version	Hadoop-1.2.1	
Virtual Machine	VMware Workstation Version 11.0.0	

Figure 4. Experimental environment

Sr. No.	Machine	CPU	Memory	Disk
1	Master	4	2 GB	20 GB
2	Slave1	2	1 GB	20 GB
3	Slave2	2	1 GB	20 GB

Table 1. Specification for each node

6.2 Results

Simple word count job for 1 GB and 2 GB data is executed on the traditional Hadoop framework and the results were obtained. The time taken by the traditional Hadoop system is shown in fig.5. Then the same job is executed on modified Hadoop framework. The results are obtained for 1 GB and 2 GB data sets. The time taken by modified Hadoop framework is shown in fig.6. After comparing both the figures, it shows that the time taken by the modified Hadoop paradigm is less with compared to traditional Hadoop paradigm.

Figure 7 shows the time taken by the system to find out the intermediate results. For any particular job performed on big data, the results are obtained for few initial records from the input data. And the same process is followed until user accepts the results. According to the proposed system, whole input data will not be processed at one glance. For example for the input data of size 1 GB, system will process first 250 MBs of data and will produce some result. Then next 250 MB of input data will be processed and both the results are combined. Likewise the system processes the input data and intermediate results are given to user.

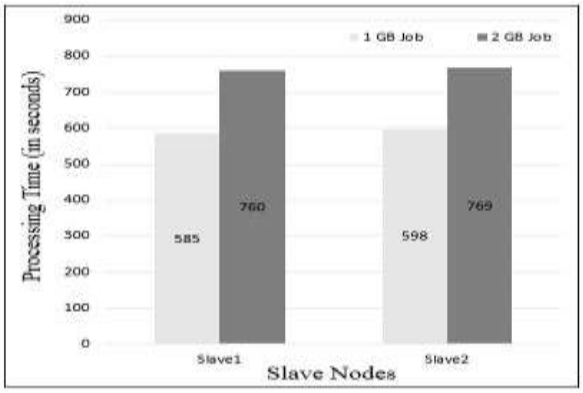

Figure 5. Execution Time of wordcount Job

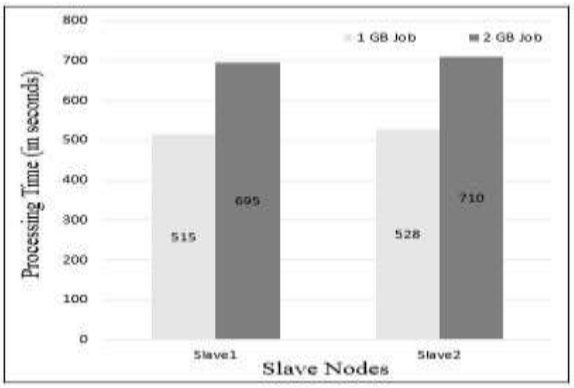

Figure 6. Impact of Combiners Strategy on Execution Time of WordCount Job

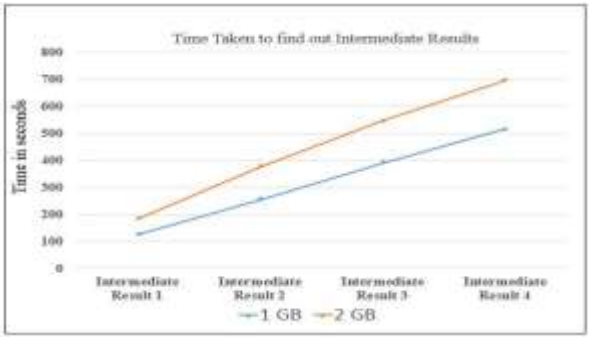

Figure 7. Time taken to find out intermediate results

VII. CONCLUSION

The proposed system is based on implementation of on the fly Aggregation of MapReduce in Hadoop for efficient big data processing. Traditional Map Reduce implementations materialize the intermediate results of mappers and do not allow pipelining between the map and the reduce phases. However, reducers cannot start executing tasks before all mappers have finished. The limitation of traditional MapReduce lowers resource utilization and leads to inefficient execution for many applications. The main motivation of Map Reduce on the fly is to overcome these problems, by allowing pipelining between operators.

The main goal of the project was to design and implement performance optimizations for big data frameworks. This work contributes methods and techniques to build tools for easy and efficient processing of very large data sets. It describes ways to make systems faster, by inventing ways to shorten job completion times successfully. Naturally, processing more data requires more time and also, performing more computations slows down analysis. Thus, in order to increase the speed of analysis, I had defined four main corresponding goals and which are achieved successfully.

VIII. FUTURE ENHANCEMENT

Future enhancement of the project work will be to enhance the map reduce functionality to achieve pipelining between jobs. Another direction would be to investigate alternative Aggregation techniques and explore the possibility for integration with large-scale processing frameworks. Another chance for future scope would be to improve the storing of data into HDFS so that processing can be benefited. There can be a limitation on how many intermediate results can be stored in Red is while processing the data. The storing capacity of the Red is can be increased to store more intermediate results.

References

[1] S. Vikram Phaneendra and E. Madhusudhan Reddy, Big Data- solutions for RDBMS problems- A survey, In 12th IEEE/IFIP Network Operations and Management Symposium (NOMS 2010) (Osaka, Japan, Apr 19, 2013).
[2] Kiran kumara Reddi & Dnvsl Indira, Different Technique to Transfer Big Data: survey, IEEE Transactions on 52(8) (Aug.2013) 2348 2355.
[3] Jimmy Lin and Chris Dyer, Data-Intensive Text Processing with MapRe-duce, University of Maryland, College Park, and Manuscript prepared April 11, 2010.
[4] Jimmy Lin MapReduce Is Good Enough?, The control project. IEEE Computer 32 (2013).
[5] Jiawei Han and Micheline Kamber, Classification and Prediction in Data Mining: Concepts and Techniques, 2nd ed., San Francisco, CA The Morgan Kaufmann, 2006.
[6] N. Laptev, K. Zeng, and C. Zaniolo, Early accurate results for advanced analytics on mapreduce, vol. 5, no. 10. VLDB Endowment, 2012, pp. 10281039.
[7] G Report from Pike research, http://www.pikeresearch.com/research/ smartgrid-data-analytics.
[8] National Climate Data Center [On the fly]. Avail-able:http://www.ncdc.noaa.gov/oa/ncdc.html.
[9] D. Borthakur, The Hadoop Distributed File System: Architecture and Design, 2007.
[10] J. Hellerstein, P. Haas, and H. Wang,On the fly aggregation, In SIGMOD Conference, pages 171182, 1997.
[11] C. Jermaine, S. Arumugam, A. Pol, and A. Dobra, Scalable approximate query processing with the dbo engine, In SIGMOD Conference, pages 725736, 2007.
[12] The apache hadoop project page, http://hadoop.apache.org/, 2013, last visited on 1 May, 2013.
[13] J. Dean and S. Ghemawat, Mapreduce: simplied data processing on large clusters, Communications of the ACM, vol. 51, no. 1, pp. 107 113, 2008
[14] S. Agarwal, A. Panda, B. Mozafari, S. Madden, and I. Stoica, Blinkdb: Queries with bounded errors and bounded response times on very large data, in ACM EuroSys 2013, 2013.
[15] N. Pansare, V. R. Borkar, C. Jermaine, and T. Condie, On the fly aggregation for large mapreduce jobs, vol. 4, no. 11, 2011, pp. 11351145.

International Journal of Computational Engineering Research (IJCER) is **UGC approved** Journal with Sl. No. 4627, Journal no. 47631.

Ms. Rucha S Shelodkar. " On The Fly Mapreduce Aggregation for Big Data Processing In Hadoop Environment." International Journal of Computational Engineering Research (IJCER) 7.7 (2017): 36-42.