ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 3, Issue 2, March 2014

Reducer Load Balancing and Lazy Initialization in Map Reduce Environment

S.Mohanapriya, P.Natesan

Abstract*—Big Data is revolutionizing 21st-century with increasingly huge amounts of data to store and be able to easily access it for a wide variety of purposes. It requires exceptional technologies to efficiently process large quantities of data within tolerable time. This kind processing and storage in the days before Hadoop was inefficient and pricey. Hadoop, open source data platform has become practically synonymous with the wildly popular term for storing and analyzing huge sets of information. Hadoop uses MapReduce as the programming model for operating this vast quantity of data. MapReduce model distributes the workload among various machines in the cluster equally. However, in the real world, data is often highly skewed by inefficient partitioner, which may cause the running nodes to have workload imbalance. In this paper, reduce-phase skew problem in MapReduce is addressed where reduce tasks are often assigned imbalance load. Moreover lazy initialization of reducers is proposed to improve the Mapreduce performance.

***Index Terms*— Hadoop; MapReduce; TeraSort; Partitioning; Skew; Lazy Initialization.**

I. INTRODUCTION

The amount of data in the world has been exploding and analyzing large data sets has become a key basis of competition. Big data refers to data volume, velocity and variety has emerged as a big trend because company has been gathering large amounts of data for decades. Some of the figures showing where such data is coming from are: every minute, 208,300 photos are uploaded to Facebook and 350,000 updates sent on Twitter. Businesses can use this data to understand customer's opinion about their companies.

The two biggest challenges in Bigdata are about storing the data and the insights companies can obtain from the data. Unlike traditional business insight, which analyses structured data, big data analytics tends to focus on unstructured data, such as emails, videos, photos and even posts on social media networks. To address the above challenges, Google developed the Google File System (GFS), a distributed file system architecture model and created the MapReduce programming model for large-scale data processing. The MapReduce programming model is a programming abstraction that hides the underlying complexity of distributed data processing. Therefore the complexity of parallelizing computation, distribute data and handle faults no long become an issue. Hadoop is an open source software implementation of MapReduce, written in Java, originally developed by Yahoo. It also uses a distributed file system called Hadoop Distributed File system (HDFS). The main advantage in using Hadoop is that it moves task to the place where data is residing making efficient usage of network bandwidth. Since its origin, Hadoop has continued to grow in popularity amongst businesses and researchers.

In this work, main two problems which degrade the performance of Hadoop MapReduce in Terasort application is addressed. First one is the load imbalance problem among reducers which may result due to inefficient partitioner. Second one is the early initialization of reducers which cause reducers to consume more memory. By addressing these two problems in Hadoop Terasort benchmark the performance of MapReduce is improved which results in less completion time.

The rest of the work is organized as follows: Section II deals with the background of MapReduce framework and the motivation for this work. Section III explains the Custom Partitioner for Reducer Load Balance. Section IV presents the Lazy initialization of Reducers in MapReduce framework. Section V shows the Experimental Procedure. Section VI concludes the report.

II. RELATED WORK

A. Google File System

Hadoop [9] [10] is the open source implementation of Mapreduce programming model. Mapreduce works along with a distributed file system called Hadoop Distributed File System (HDFS) in Hadoop. Both Mapreduce and

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 3, Issue 2, March 2014

HDFS in hadoop are derived from the Google's Mapreduce and Google File System. Ghemawat et al [3] presented a File System called Google File System (GFS) which was developed at Google to meet the rapidly growing demands for processing their data. This file system is widely used in Google for storing large data sets. Like other distributed file system, GFS provides high performance, scalability, reliability, and availability. This File System is taken as the base for developing the file system in Hadoop called HDFS.

B. MapReduce Model

MapReduce [7] is a programming model developed as a way for programs to cope with large amounts of data. Dean et al [2] proposed a framework for processing data in distributed manner called MapReduce. MapReduce is the model for processing and generating large data sets. Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures and managing the required inter-machine communication.

C. Terasort Benchmark

Hadoop broke the world record by sorting terabytes of data using the TeraSort method. Malley [6] proposed sorting terabytes of strings using Apache Hadoop. Burst Sort [5] is a sorting algorithm designed for sorting strings which is as fast as Trie. TeraSort uses trie and was able to do this sorting process by distributing the workload evenly among the machines. It does this by using custom partitioner. Hadoop wrote 3 applications for terabyte sort such as TeraGen, TeraSort and TeraValidate.

D. Hadoop Distributed File System

Shafer et al [8] presented a distributed file system called HDFS. This is the file system used by Hadoop which is the clone of GFS. This file system is used to store vast amount of data by distributing load across machines. All files in the HDFS follow the write-once, read-many access rule. HDFS [1] is a popular Internet service file system that provides the right abstraction for data processing in Mapreduce frameworks.

E. Data Skew in MapReduce

Load imbalance is the major problem which degrades the performance of MapReduce job. Gufler et al [4] addressed the Data Skew problem in MapReduce. The author proposed how to efficiently process MapReduce jobs with complex reducer tasks over skewed data. For this purpose two load balancing approaches are proposed namely, fine partitioning and dynamic fragmentation.

III. REDUCER LOAD BALANCING USING CUSTOM PARTITIONER

A. Terasort

The TeraSort benchmark is the most well-known Hadoop benchmark. In 2008, Hadoop set a record by sorting 1 TB of data in 209 seconds. Basically, the goal of TeraSort is to sort 1TB of data (or any other amount of data required) as fast as possible. Hadoop was able to do this sorting by efficiently partitioning data among machines. For this Terasort uses quicksort for determining the cut-points and trie to partition the data among the reducers using the cut-points. A trie is a tree based data structure used for storing strings. In Terasort two-level trie is constructed which stores only first two characters in every string.

B. Xtrie

Terasort uses Quick sort for determining cut points whose time complexities are greater than that of Trie. However using Trie instead of Quicksort will restrict to only two-level for determining the cut-points. By using only two-level trie cut-points are determined based on number of prefix and not on the number of strings in each prefix which leads to faulty cut-points. As a result there will be a load imbalance among reducers when partitioning is made using these cut-points.

In order to overcome the load imbalance problem faced in using two-level Trie, a technique called Xtrie is proposed to partition the data. In this for each word in the trie it maintains the counter value which contains the number of occurrences in that word. This counter value is usually maintained by using array which is indexed using the trie code.

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 3, Issue 2, March 2014

Consider the example set of keys,

{“star”, “cash”, “fish”, “anger”, “aim”, “fill”, “fire”, “quick”, “man”, “steal”, “fight”, “foot”, “quiz”, “stick”, “steel”, “angel”}.

The above strings are stored in the trie as in Figure 1.

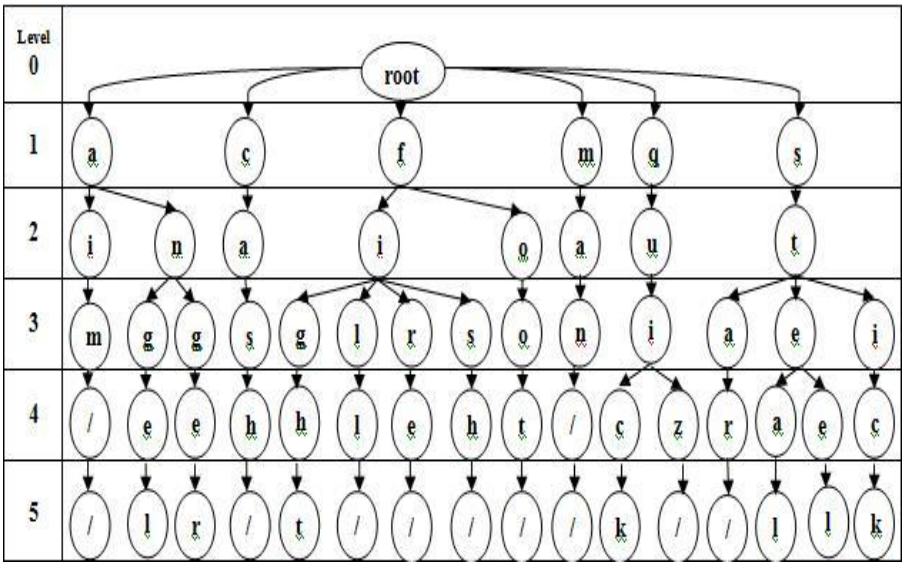

Fig 1.Strings stored in Trie

The formula for calculating the TrieCode is shown in (1) below which is used as an index to the array. Here W_n represents the ASCII value of n^{th} letter in the word and n represents the number of letters in the word.

$$\begin{aligned} \text{Triecode} &= W_n \times 256^{n-1} + W_{n-1} \times 256^{n-2} + \ldots + W_1 \times 256^0 \\ &= \sum_{n=1}^{\text{TotalWord}} W_n \times 256^{n-1} \end{aligned} \quad (1)$$

Obtaining the counter value for each prefix, the cutpoints are obtained using (2),

$$\text{cutpoints} = \frac{\text{Total number of strings in each prefix}}{\text{Number of Reducers}} \quad (2)$$

Based on this the partitions are made among reducers so that the load are equally balanced among the reducers.

By using two level trie the above set of keys shown in Figure 1 is reduced to

{“ai”, “an”, “ca”, “fi”, “fo”, “ma”, “qu”, “st”}

Consider there are 4 reducers and the above set is divided into 4 partitions. Each partition is send to one reducer. Thus,

Reducer-1 process keys starting with {“ai”, “an”}.Total:3 keys
Reducer-2 process keys starting with {“ca”, “fi”}.Total:5 keys
Reducer-3 process keys starting with {“fo”, “ma”}.Total:2 keys
Reducer-4 process keys starting with {“qu”, “st”}.Total:6 keys

This results in workload imbalance among the reducers.

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 3, Issue 2, March 2014

To overcome the above problem, in Xtrie it uses counter for each key value. Using counter, partitioner can distribute the total number of keys among the reducers evenly.
Cut-points based on counter value is,
Cut-points=16/4=4

Using these cut-points, split strings are obtained. The obtained spilt-strings are,

{“fi”, “fo”, “st”}

So strings with prefix less than fi goes to partition 1, strings with prefix equal to fi and less than fo goes to partition 2, strings with prefix equal to fo and less than st so goes to partition 3 and finally the string with prefix st goes to partition 4. The working of Xtrie is shown in Table 1.

Table 1. Xtrie Partitioner using Two-level Trie

Prefix	Keys	Trie Code	Count	Partition
ai	Aim	0X6169	1	1
an	Angel	0X616e	2	
	Anger			
ca	Cash	0X6361	1	
fi	Fight	0X6669	4	2
	Fill			
	Fire			
	Fish			
fo	Foot	0X666f	1	3
ma	Man	0X6d61	1	
qu	Quick	0X7175	2	
	Quiz			
st	Star	0X7374	4	4
	Steal			
	Steel			
	Stick			

C. Reducing Memory Requirements of Trie

The trie is represented by using array. Each node in the trie will contain a maximum of 256 children. It is not possible to have all the 256 children for a single node. This problem will make the trie to occupy lots of memory space.

To reduce the memory requirements of trie, an algorithm called ReMap algorithm is used which reduces the 256characters on ASCII to 64 elements as required by Etrie method. Using less memory allows deeper tries to be built. Deeper tries also provides the chance of distributing keys evenly among reducers.

Consider the following set of keys,
{“grab”, “gram”, “gray”, “grew”, “grid”, “grip” ,“grow”, “grub”, “gulf”}.

Attempting to partition keys evenly using a two-level trie is impossible in this case. Using a two-level trie would create set of prefixes {“gr”, “gu”}. Although if identified that the prefix “gr” represents 8 keys and prefix “gu” represents 1 key, the prefix themselves are indivisible, which creates a load imbalance.

This problem can be overcome using deeper tries. By using deeper tries, the length of the prefix can be increased. This increases the number of prefix and reduces the number of keys per prefix. Using this even distribution of keys among the reducers can be created. The above set of keys using three-level is represented using prefix {“gra”, “gre”, “gri”, “gro”, “gru”, “gul”} thus dividing the 8 keys represented by prefix “gr” into five smaller categories.

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 3, Issue 2, March 2014

IV. LAZY INITIALIZATION OF REDUCER

In MapReduce job Reducers are initialized with Mappers at the job initialization, but the reduce method is called in reduce phase when all the maps had been finished. So in large jobs where Reducer loads data (>100 MB for business logic) in-memory on initialization, the performance can be increased by lazily initializing Reducers i.e. loading data in reduce method controlled by an initialize flag variable which assures that it is loaded only once.

By lazily initializing Reducers which require memory on initialization, number of maps can be increased (controlled by mapred.tasktracker.map.tasks.maximum).

For example,
Consider that each node in the cluster will have an 8GB RAM and the memory required for each Map and Reduce task is shown in the Table 2.

Table 2. Memory Requirements for MapReduce Job in a Node

Total Memory per Node	Max. Memory for each Map Task	Reducer Memory During Initialization	Max. Memory for each Reduce Task	No. of Reducers	Memory for Data Node + Task Tracker
8GB	400MB	200MB	400MB	4	2GB

A. Before Lazy Initialization of Reducers

The number of Map tasks that can be run in a node before lazy initialization of Reducer with the given system requirement is shown in the Table 3.

Table 3.Maximum Map Tasks that can run in a Node Before Lazy Initialization of Reducers

Max. Memory Required by all Reduce Job	Number of Map Tasks that can be run in a Node
4*400MB=1600MB	(8-2-1.6)/400MB=11 Map tasks

B. after Lazy initialization of Reducers

The number of Map tasks that can be run in a node after lazy initialization of Reducer with the given system requirement is shown in the Table 4.

Table 4.Maximum Map Tasks that can run in a Node After Lazy Initialization of Reducers

Max. Memory Required by all Reduce Job	Number of Map tasks that can be run in a Node
4*(400-200)=800MB	(8-2-0.8)/400MB=13 Map tasks

So by lazily initializing Reducers, 2 more Map tasks can be run in a node.
Therefore by lazy initialization of reducers less memory is consumed by reducers and more map tasks can be executed in a node which results in increase in performance of the MapReduce job.

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 3, Issue 2, March 2014

V. EXPERIMENTAL PROCEDURE

The experiment is conducted by using Hadoop-1.2.1. Hadoop is installed in fully-distributed mode among the multiple VMs created in Virtual Box. The system requirements are 4GB RAM, 100GB hard disk, java version 1.6 or any other versions above version 6, Ubuntu Linux OS.

A. Hadoop-1.2.1

Hadoop is the open source implementation of mapreduce. Many versions of hadoop are available. The Hadoop version 1.2.1 is one of the latest releases when the work is carried out and is considered as the stable one among various Hadoop versions available.

B. Results

Using the above experimental setup, the work is carried out with a small data set. As a result the loads will be equally balanced among the reducers using custom partitioner and thus improve the completion time of MapReduce. Table 5 shows the Execution Time of MapReduce job at three different reducers without using the Xtrie partitioner and Table 6 shows how the Execution time get decreased when using Xtrie partitioner.

Table 5. Execution Time at Different Reducers without Xtrie Partitioner

Number of Records	Number Of Partitions	Execution Time(sec)		
		Reducer1	Reducer2	Reducer3
7,00,000	3	19	16	13
10,00,000		21	21	13

Table 6. Execution Time at Different Reducers using Xtrie Partitioner

Number of Records	Number Of Partitions	Execution Time(sec)		
		Reducer1	Reducer2	Reducer3
7,00,000	3	17	16	13
10,00,000		21	20	13s

Figure 2 shows Execution time (in seconds) of MapReduce job with and without using Xtrie partitioner. The graph shows that the Execution time is reduced when using Xtrie partitioner thus improving job performance.

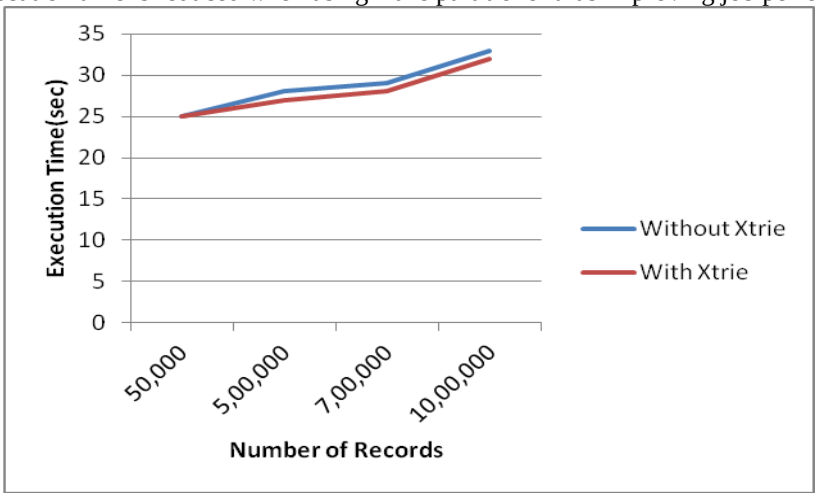

Fig 2. Execution Time with and without Xtrie Partitioner

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 3, Issue 2, March 2014

VI. CONCLUSION

Hadoop is the distributed processing environment which is used for processing huge volume of data. It is the partitioning of the data which determines the workload of the reducers. In order to overcome the problem of skew, the data should be partitioned efficiently among the reducers. If the data are partitioned equally among the nodes then the execution time for overall MapReduce job is decreased. The proposed techniques can be used to efficiently eliminate the problem of data skew in reducers and thereby decreasing the processing time of MapReduce job.

Moreover, by lazy initialization of reducers less memory will be utilized by reducers which will cause the more number of map tasks to be executed in a node. As a result load balancing and lazy utilization of reducers improve the performance of MapReduce job in Hadoop.

REFERENCES

[1] D.Borthakur, "The Hadoop Distributed File System: Architecture and Design", the Apache Software Foundation, (2007).

[2] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters", Commun. ACM, pages:107–113, (2008).

[3] S. Ghemawat, H. Gobioff, S-T Leung, "The Google file system", Proceedings of the 19th ACM symposium on operating systems principles (SOSP), (2003).

[4] B. Gufler, N. Augsten, A. Reiser and A. Kemper, "Handling Data skew in Mapreduce", Proceedings of the international conference on Cloud Computing and Services Science(CLOSER), pp. 574-582, (2011).

[5] S. Heinz, J. Zobel, H.William,"Burst tries: a fast, efficient data structure for string keys", ACM Trans Inf Syst 20(12):192–223, (2002) .

[6] O. O'Malley, "Tera Byte sort on Apache Hadoop", (2008).

[7] A. Matsunaga, M. Tsugawa, J. Fortes, "Programming abstractions for data intensive computing on clouds and grids", In: IEEE fourth international conference on escience, pp 489–493, (2008).

[8] J. Shafer, S. Rixner, AL. Cox, "The hadoop distributed file system: balancing portability and performance", Proceedings of the IEEE international symposium on performance analysis of system and software (ISPASS), p 123, (2010).

[9] Hadoop, http://hadoop.apache.org/core.

[10] http://developer.yahoo.com/hadoop/tutorial/