ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 4, April 2016

A Partitioning Technique in MapReduce for Optimizing Large Data Analysis

Bangar Bhavna, Dangde jyoti, Priya Bhakre, Megha Godake

B.E. Students, Dept. of computer, MIT AOE, Alandi, Pune, India

ABSTRACT: In today era huge amount of data generated from various sources like Facebook, Twitter, LinkedIn, and forum and different other sources. That massive volume of both structure and unstructured data is difficult to capture, store and manipulate with traditional database and software techniques. Using big data, many companies and users started to move their data towards cloud storage so as to simplify data management and reduce data maintenance cost. In most companies the size of data is too big or it moves too fast and it exceeds current processing capacity. Other than these problems, big data has the ability to help companies improve operations and make faster and more intelligent decisions for data analysis. MapReduce is recent programming model for writing distributed application that handle huge amount of data. MapReduce divide the workload among the multiple computers in a network. Performance of MapReduce depends on how it distributes the workload among the multiple computers. This can be most challenging task in advent of data skew (i.e. data is generated in invariant capacity). In MapReduce distributions of workload depend on algorithms that partition the data. To avoid the problem of data skew data sampling techniques are used. Data sampling is a statistical analysis technique. It is used to analyze, manipulate and select a representative subset of data points in order to identify patterns and trends in the larger data set being examined. How evenly the partitions distributes the data depends on how large and representative the sample is and on how well the samples are analyzed by the partitioning mechanism. In this paper proposes an improved partitioning algorithm using custom partitioning techniques that improve the load balancing and memory consumption. To evaluate the result of proposed algorithm, performance is compared with the various existing system Proposed algorithm is more is faster, more memory efficient, and more accurate than the current implementation.

KEYWORDS: Big Data, Data sampling, Hadoop, HDFS, MapReduce, Partitioning.

I. INTRODUCTION

Now-a-days the computer technology has become increasingly ubiquitous. Personal computers, smart phones, tablets, and an ever-growing number of embedded devices can connect and communicate with each other via the internet. Computing devices have numerous uses. They are essential for businesses, scientists, governments, engineers, and the everyday consumer. These devices have the common potential to generate data. The data can come from anywhere. Sensors gathering climate data, a person posting to a social media site. The example of sources of data is GPS signal. The data is stored distributed over the network. Fetching the data across the network is the main part in the now-a-days technology. The amount of data generated can be too large for a single computer to process in a reasonable amount of time. The data is too large to store on single machine. Therefore, for reducing the time which it takes to process the data and the storage space to store the data the software engineers have to write the program and that execute data on one or more computers and distribute the workload among them. It is very difficult concept for implementation. The Google developed the Google File System (GFS)[6] which is distributed file system architecture model for large-scale data processing. The MapReduce[2] programming model is used for programming abstraction. It hides the underlying complexity of distributed data processing. Hadoop[5] is an open source software implementation of MapReduce. Hadoop is written in Java developed by Yahoo!. Hadoop is used by many universities and also it is used in many companies including Twitter, FaceBook, IBM, LinkedIn and EBay. Hadoop has continued to grow in popularity among the businesses and researchers. For improving the features which are already there, Pig, HBase, Hive, and ZooKeeper are the examples of commonly used extensions to the Hadoop framework. Similarly, this paper also focuses on Hadoop and the load balancing mechanism for small to medium-sized clusters using the Map-reduce framework. This is an important area of research because many clusters that use Hadoop are of modest size and small companies, software

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 4, April 2016

engineers and researchers do not have the resources to develop large cluster environments themselves, and often clusters of a modest size are required for certain computations.
Summary of this paper has following contributions:

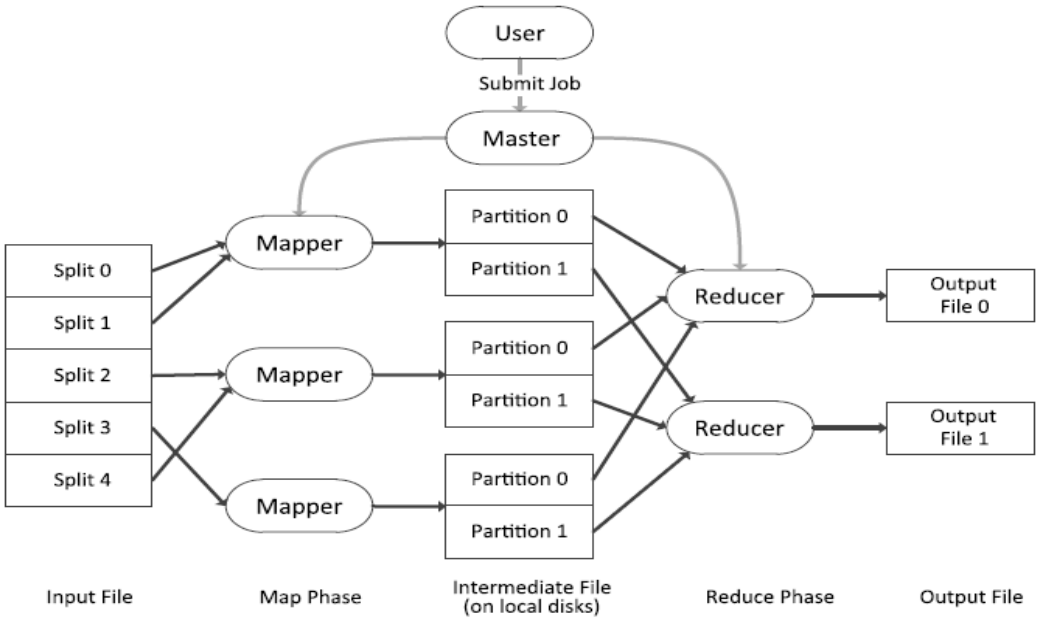
• For improving the workload distribution among the multiple machines MapReduce Framework is used.

• To avoid more memory consumption using efficient technique.

• Using custom partitioning technique time for the execution will be reduce and load balancing mechanism should be achieve.

MapReduce

MapReduce[2] is a programming model developed for programs to cope with large amounts of data. It distributes the workload among multiple machines and they parallel work on that data. MapReduce is a relatively easy way to create distributed applications. Because of that reason MapReduce has become popular. It is key technology. There are two phases in the MapReduce framework one is the "'Map" and second one is the "reduce". Each phase has key-value pairs for both input and output[12]. For implementing these phases, a programmer needs to specify two functions: a map function called a Mapper and another is reduce function called a Reducer. When a MapReduce program is executed on multiple machines or nodes. Therefore, a master node is required to run the services needed to coordinate the communication between Mappers and Reducers. An input file is then split up into fixed sized pieces. This is called input splits. These splits are then passed to the Mappers. These splits work in parallel to process data contained within each split. As the Mappers process the data and partition the output. Each Reducer then gathers the data partition from each Mapper, merges each Mappers, processes them, and produces the output file which designated for them.
An example of this dataflow of MapReduce is in Fig.1



This dataflow of Map-reduce shows the partitioning of the data that determines the workload for each reducer. In the MapReduce[2] framework, for utilizing resources efficiently the workload must be balanced in order to achieve load balancing mechanism. If there is imbalanced workload then some reducers have more work to do than others. This means that there can be reducers standing idle which is time consuming. Because of other reducers are still processing

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 4, April 2016

the workload designated to them which increases the time for completion. The job of MapReduce is not complete until all reducers finish their workload.

HashCode

HashCode Hadoop uses a default method which is hash code[1] as it partition key-value pairs. The hash code can be expressed mathematically and is represented as following: HashCode = Wn × 31n−1 + Wn−1 × 31n−2 + · · · + W1 × 310

$$= \sum_{n=1}^{TotalWord} W_n \times 31^{n-1} \quad \text{....(1)}$$

The hash code presented in Eq. (1) . It is the default hash code. It is used by a string object in Java, the programming language on which Hadoop is based. In this equation, Wn is used for representing the nth element in a string. The integer 31 is used because it is a prime number. Hash codes use prime numbers. Because it generates a unique value when multiplied with another number. A partition function uses the hash code of the key and the modulo of reducers to determine which reducer to send the key-value pair to. It is important that the evenly distribution of key-value pairs among reducers for proper workload distribution by partition function.

II. RELATED WORK

TeraSort

The evenly distribution of workload among the multiple computer which accelerate the sorting process which is done by the TeraSort. In April 2008, a Terabyte of data is sorted by using its TeraSort[3] method. The time for sorting 1 TB of data was 209 seconds that is 3.48 minutes. The first time either a Java program or an open source program had won the competition. For this data sampling and trie[4] is used. The main aim of TeraSort was to sort 1 TB of data as early as possible which is benchmark for Hadoop, it has since been integrated into Hadoop as a benchmark. Overall, the TeraSort algorithm is same as the standard MapReduce sort. The efficiency is based on how it distributes its data between the Mappers and Reducers. TeraSort uses a custom partition for achieving a good load balance. For defining a range of keys for each reducer, the custom partitioner uses a N − 1 sampled keys of sorted list. If key resides within a range then it is sent to the reducer such that sample[i − 1] <= key < sample[i]. This shows that the output of reducer i is oftenly less than the output for reducer i + 1. It samples the data and extracts keys from the input split, before the partitioning process for TeraSort begins. The keys are saved to a file in the distributed cache[11]. The keys are processed in the file using the partitioning algorithm. Since the goal of TeraSort was to sort data as early as possible. The implementation of TeraSort adopted a space for time approach. TeraSort uses a two-level trie to partition the data for achieving the above goal.
A trie[4] is also called as prefix tree. It is an ordered tree used to store strings. Throughout this paper, upto two characters trie limits strings stored in it.

III. PROPOSED ALGORITHM

This section introduces the concept of the Xtrie. It and describes how TeraSort can be modified using an Xtrie and how Xtrie can benefit TeraSort. This section describes about the Etrie and shows how it can save memory using a Remap method.

Xtrie

For improve the cut point algorithm which is generated from TeraSort[1], the Xtrie[1] algorithm is used. It uses the quicksort algorithm to handle cut points which is the problem for TeraSort. By using quick sort TeraSort needs to store all the keys it samples in memory. It reduces the accuracy of the selected cut points and this affects load balancing [3]. Another problem of TeraSort is that it only considers the first two characters of a string during partitioning. This reduces the effectiveness of the TeraSort load balancing algorithm.

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 4, April 2016

The procedure for implementing the Xtrie partitioning technique is as follows:

1. Take input file.

2. Input file samples store in trie.

3. Build twoor three level trie.

4. Maintain counter value for occurrence of each prefix 5. Determine cut point using cut point algorithm.

6. Cut points are determined by dividing the sum of counter value by number of partitions+1.

Cut points = sum of counters/number of partitions+ 1

7. Send key to appropriate reducers using cut points.

```
if (key<cutp1)
    Send key to reducer 1
else if(key>=cutp-1&&key<cutp-2)
   Send key to reducer2
else if (key>=cutp-2&&key<cutp-3)
   Send key to reducer3
else
   Send to the finished reducer
```

8. Determine if there is any slow running node by comparing the performance of each node with other

9. If there is any slow running node move the data that is processing on that node to the free node.

If number of strings in each prefix is unknown then two-level trie leads to load imbalance among the reducer. To overcome the problem of load imbalance for each word in trie it maintains a counter value. Using counter value it distributes the total number of key among the reducer. For implementation of trie array is use and it access through trie code. The equation for trie code is as follows:

$$TrieCode = W_n \times 256^{n-1} + W_{n-1} \times 256^{n-2} + \cdots + W_1 \times 256^0$$
$$= \sum_{n=1}^{TotalWord} W_n \times 256^{n-1}$$

Example:

Consider the example set of keys, {“act”, “acid”, “add”, “adult”, “born”, “good”, “grip”, “grow”, “peal”, “pearl”, “pedal”, “soft”, “stork”, “storm”}.

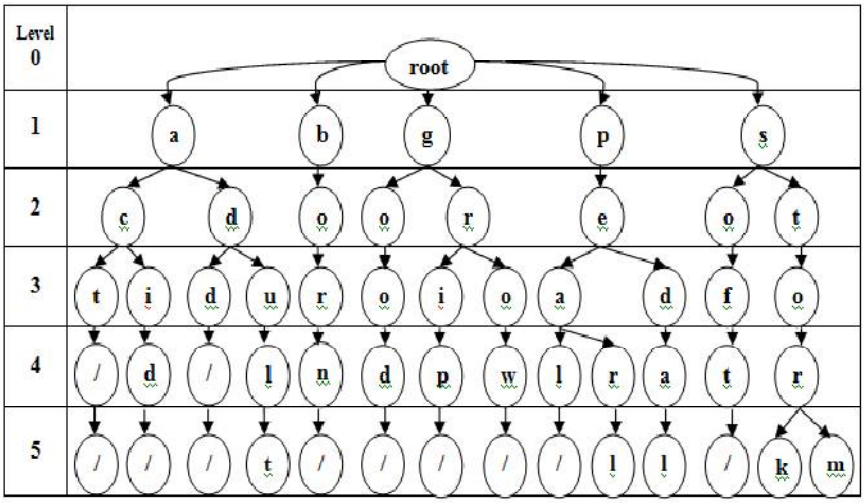
String store in trie:

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 4, April 2016



Using two level trie the above set of string is reduced to
{“ac”, “ad”, “bo”, “go”, “gr”, “pe”, “so”, “st”}.

If we consider 4 reducer then file are splits into 4 partitions. String store in trie is send to store in one reducer.

Process keys starting with {“ac”, “ad”} send to Reducer-1. Total:4 keys

Process keys starting with {“bo”, “go”} send to Reducer-2. Total:2 keys

Process keys starting with {“gr”, “pe”} send to Reducer-3. Total:5 keys

Process keys starting with {“so”, “st”} send to Reducer-3. Total:3 keys

In above example load is imbalance because of particular key value pair are not send to particular reducer.

Using Xtrie:

Prefix	Keys	Trie Code	Count	Partition
ac	act, acide	0X6163	2	
ad	add, adult	0X6164	2	1
bo	born	0X626f	1	
go	good	0X676f	1	2
gr	grip,grow	0X6772	2	
pe	pearl, pedal, pearl	0X7065	3	3
so	soft	0X736f		
st	stork,storm		1	4

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

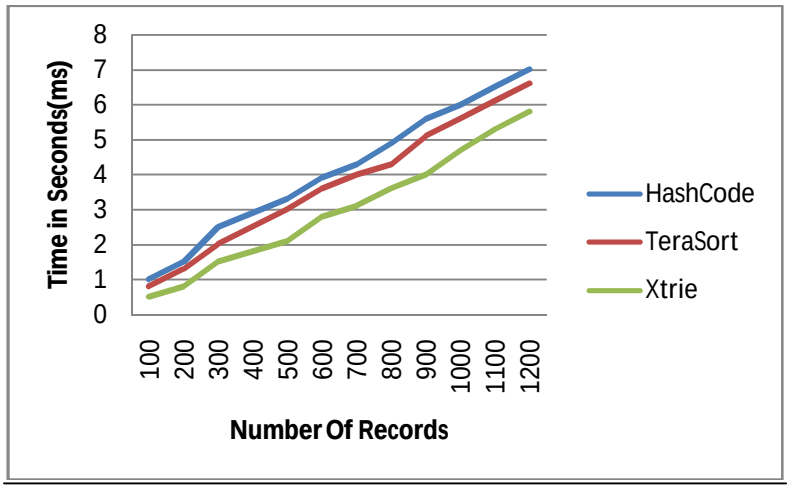
Vol. 4, Issue 4, April 2016

Etrie

One of the problem with TeraSort is that it uses array for implementation of trie. Each node in trie will contain 256 children. For single node it is not possible to have all 256 children. The problem with this it leads to wasted space in trie. In text file and log file contain white space, alphanumerical characters, line break are also filtered by mapper. So it will effect on memory consumption. To reduce memory consumption Erie[1] algorithm is used. ReMap algorithm is used which reduces the 256 characters on ASCII to 64 elements to reduce the memory requirement.

IV. RESULT

The performance of the proposed work is optimize using the given the hadoop setup. It is then compared with the existing techniques like TeraSort to show that the performance of the Xtrie partitioning technique has good load balancing among the reducers in Mapreduce framework. The figure 2 shows the time complexity comparison of TeraSort and Xtrie.



V. CONCLUSION AND FUTURE WORK

Thus we studied in this paper XTrie and ETrie, partitioning techniques, to improve load balancing for distributed applications. For improving load balancing, MapReduce Programs can become more efficient at handling tasks by reducing the overall computation time spent processing data on each machine. The TeraSort, developed by O'Malley, Yahoo, was designed based on a very large cluster of 910 nodes for randomly generated input data. In that particular computing environment and each partition generated on only one or two nodes for that data configuration by MapReduce. In contrast, our study looks for small- to medium-sized clusters this study modifies their design and optimizes it for a smaller environment. A series of experiments have shown using given a skewed data sample, the Etrie architecture was able to conserve more memory, can allocate more computing resources, and do so with less time complexity. In future, using extending this framework further research can be done so that it can use different type micro partitioning methods for applications using different input samples.

VI. ACKNOWLEDGMENT

The author would like to thank Prof. Prajakta Ugale for his help in preparing this paper.

REFERENCES

1. Kenn Slagter, Ching-Hsien Hsu, Yeh-Ching Chung and Daqiang Zhang, "An improved partitioning mechanism for optimizing massive data analysis using MapReduce", Journal of Supercomputing, 66(1): 539-555, (2013).
2. J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters", CommunACM 51:107–113, (2008)
3. O. O'Malley , "TeraByte sort on Apache Hadoop", (2008).
4. Panda B, Riedewald M., Fink D., "The model summary problem and a solution for trees", In: International conference on data engineering, pp 452-455, (2010).
5. Apache software foundation, "http://hadoop.apache.org/core", (2007).
6. S. Ghemawat, H. Gobioff, S-T Leung, "The Google file system", Proceedings of the 19th ACM symposium on operating systems

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 4, April 2016

principles (SOSP), (2003).
7. Nandhini.C, Premadevi.P IJIRCCE, "A Micro Partitioning Technique in MapReduce for Massive Data Analysis", (2014).
8. 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).
9. Papadimitriou S, Sun J, "Distributed clustering with map-reduce", In: IEEE international conference on data mining, p 519, (2008).
10. Chang F, Dean J, Ghemawat S, Hsieh WC, Wallach DA, Burrws M, Chandra T, Fikes A, Gruber RE, "Bigtable: a distributed storage system for structured data", In: 7th UENIX symposium on operating systems design and implementation, pp 205-218, (2006).
11. J. Shafer, S. Rixner, AL. Cox, ″The hadoop distributed filesystem: balancing portability and performance” , Proceedings of the IEEE international symposium on performance analysis of system and software (ISPASS), p 123, (2010).
12. Hsu C-H, Chen S-C, Lan C-Y, "Scheduling contention-free irregular redistribution in parallelizing compilers", J Supercomput 40(3):229-247, (2007).