Journal of Computer Science

Original Research Paper

A Modified Key Partitioning for BigData Using MapReduce in Hadoop

Gothai Ekambaram and Balasubramanie Palanisamy

Department of CSE, Kongu Engineering College, Erode-638052, Tamilnadu, India

Article history
Received: 07-03-2014
Revised: 23-03-2014
Accepted: 21-03-2015

Corresponding Author:
Gothai Ekambaram
Department of CSE, Kongu Engineering College, Erode-638052, Tamilnadu, India
Email: kothaie@yahoo.co.in

Abstract: In the period of BigData, massive amounts of structured and unstructured data are being created every day by a multitude of ever-present sources. BigData is complicated to work with and needs extremely parallel software executing on a huge number of computers. MapReduce is a current programming model that makes simpler writing distributed applications which manipulate BigData. In order to make MapReduce to work, it has to divide the workload between the computers in the network. As a result, the performance of MapReduce vigorously depends on how consistently it distributes this study load. This can be a challenge, particularly in the arrival of data skew. In MapReduce, workload allocation depends on the algorithm that partitions the data. How consistently the partitioner distributes the data depends on how huge and delegate the sample is and on how healthy the samples are examined by the partitioning method. This study recommends an enhanced partitioning algorithm using modified key partitioning that advances load balancing and memory utilization. This is completed via an enhanced sampling algorithm and partitioner. To estimate the proposed algorithm, its performance was compared against a high-tech partitioning mechanism employed by TeraSort. Experimentations demonstrate that the proposed algorithm is quicker, more memory efficient and more accurate than the existing implementation.

Keywords: Hadoop, Hash Code, Partitioning, MapReduce

Introduction

Over the past decades, computer technology has become increasingly ubiquitous. Computing devices have numerous uses and are essential for businesses, scientists, governments, engineers and the everyday consumer. What all these devices have in general is the probable to produce data. In essence, data can arrive from everywhere. The majority types of data have a propensity to have their own distinctive set of characteristics over and above how that data is dispersed. Data that is not examined or utilized has small significance and can be a waste of space and resources. On the contrary, data that is executed on or examined can be of immeasurable value. The data itself may be too huge to store on a single computer. As a result, in order to decrease the time it takes to execute the data and to have the storage space to store the data, software engineers have to write down programs that can perform on 2 or more computers and dispense the workload amongst them. While abstractly the computation to execute may be straightforward, traditionally the implementation has been complicated. In reaction to these extremely same matters, engineers at Google built up the Google File System (GFS) as stated by (Ghemawat *et al*., 2003), a distributed file system design representation for major data processing and formed the MapReduce programming model by (Dean and Ghemawat, 2008).

Hadoop is an open source implementation of MapReduce, written in Java, initially developed by Yahoo. Tan *et al*. (2009) stated that Hadoop was built in response to the need for a MapReduce structure that was unfettered by proprietal licenses, in addition to the increasing need for the technology in Cloud computing. Hive, Pig, ZooKeeper and HBase are all examples of regularly utilized extensions to the Hadoop structure. Likewise, this study also concentrates on Hadoop and examines the load balancing mechanism in Hadoop's MapReduce skeleton for small-sized to medium-sized clusters.

In summary, this study presents a technique for increasing the work load distribution among nodes in the MapReduce framework, a technique to decrease the

necessary memory footprint and improved execution time for MapReduce when these techniques are performed on small or medium sized cluster of computers. The remaining part of this study is planned as follows. Section 2 discusses some basic information on MapReduce and its internal workings. Section 3 presents the related work and existing methods applied for TeraSort in Hadoop. Section 4 contains a proposed idea for an improved load balancing methodology and a way to better utilize memory. Section 5 introduces investigational results and a discussion of this study's findings. Section 6 concludes this study with a brief idea to future work.

Background

MapReduce

Dean and Ghemawat (2008) mentioned that MapReduce is a programming representation created as a method for programs to handle with huge amounts of data. It attains this objective by distributing the workload among several computers and after that working on the data in parallel. Hsu *et al*. (2007) stated that programs that perform on a MapReduce structure need to separate the work into 2 phases known as Map and Reduce. Each phase has key-value pairs for both input and output. To put into practice these phases, a programmer needs to state 2 functions: A map function called a Mapper and its equivalent reduce function called a Reducer. While a MapReduce program is performed on Hadoop, it is anticipated to be run on several computers or nodes. For that reason, a master node is necessary to run all the essential services desired to organize the communication between Mappers and Reducers. An instance of MapReduce dataflow is shown in Fig. 1. Kavulya *et al*. (2010) reported that in the MapReduce structure, the workload has to be balanced in order for resources to be utilized powerfully.

HashCode

Hadoop utilizes a hash code as its standard method to partition key-value pairs. The hash code itself can be depicted mathematically and is represented by (Kenn *et al*., 2013) as the subsequent equation:

$$HashCode = W_n * 31^{n-1} + W_{n-1} * 31^{n-2} + ... + W_1 * 31^0 = \sum_{n=1}^{TotalWord} W_n * 31^{n-1} \quad (1)$$

The hash code given in Equation 1 is the default hash code utilized by a string object in Java, the programming language on which Hadoop is based. A partition function normally utilizes the hash code of the key and modulo of reducers to decide which reducer to send the key-value pair to. It is essential then that the partition function uniformly distributes key-value pairs among reducers for appropriate workload distribution.

TeraSort

O'Malley (2008) stated that Hadoop ruined the world record in sorting a Terabyte of data by using its TeraSort technique. Winning first place it managed to sort 1 TB of data in 209 sec (3.48 min). This was the first occasion either a Java program or an open source program had won the contest. TeraSort was able to step up the sorting process by distributing the workload uniformly within the MapReduce framework. This was done via data sampling and the use of a Trie as stated by (Panda *et al*., 2010). Even though the main goal of TeraSort was to sort 1 TB of data as speedily as possible, it has since been incorporated into Hadoop as a standard.

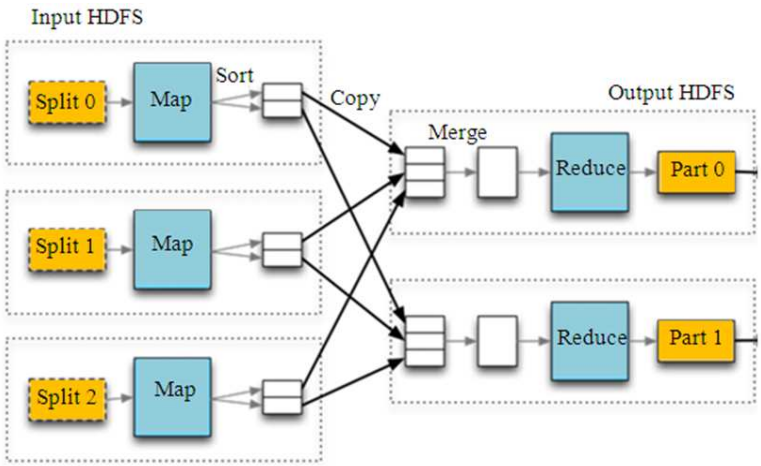

Fig. 1. MapReduce dataflow

On the whole, the TeraSort algorithm is extremely alike to the standard MapReduce sort. Its efficiencies rely on how it distributes its data between the Mappers and Reducers. To attain an excellent load balance, TeraSort uses a custom partitioner. Since the original goal of TeraSort was to sort data as speedily as possible, its implementation adopted a space for time approach. For this reason, TeraSort utilizes a 2-level trie to partition the data. Kenn *et al*. (2013) has shown that a trie which confines strings stored in it to 2 characters is known as 2-level Trie. This 2-level Trie is built using cut points extracted from the sampled data. Once the trie is constructed using the cut points, the partitioner can initiate its job of partition strings based on where in the trie that string would go if it were to be included in the trie.

Related Works

Sorting is a primary concept and is mandatory step in countless algorithms. Heinz *et al*. (2002) stated that Burst Sort is a sorting algorithm developed for sorting strings in huge data collections. The TeraSort algorithm also utilizes these burst trie techniques as a method to sort data but does so under the perspective of the Hadoop architecture and the MapReduce framework. An essential problem for the MapReduce framework is the idea of load balancing. Over the period, several researches have been done on the area of load balancing. Where data is situated by (Hsu and Chen, 2012), how it is communicated by (Hsu and Chen, 2010), what background it is being located on by (Hsu and Tsai, 2009; Hsu *et al*., 2008; Zaharia *et al*., 2008) and the statistical allotment of the data can all have an outcome on a systems efficiency. Most of these algorithms can be found universal in a variety of papers and have been utilized by structures and systems earlier to the subsistence of the MapReduce structure stated by (Krishnan, 2005; Stockinger *et al*., 2006). As stated by (Candan *et al*., 2010), RanKloud make use of its personal uSplit method for partitioning huge media data sets. The uSplit method is required to decrease data duplication costs and exhausted resources that are particular to its media based algorithms. So as to work just about perceived boundaries of the MapReduce model, various extend or changes in the MapReduce models have been offered. BigTable was launched by Google to handle structured data as reported by (Chang *et al*., 2008). BigTable looks like a database, but does not support a complete relational database model. It utilizes rows with successive keys grouped into tables that form the entity of allocation and load balancing. And experiences from the similar load and memory balancing troubles faced by shared nothing databases. HBase of Hadoop is the open source version of BigTable, which imitates the similar functionality of BigTable. Because of its simplicity of use, the MapReduce model is pretty popular and has numerous implementations as reported by (Liu and Orban, 2011; Miceli *et al*., 2009). For that reason, there has been a diversity of research on MapReduce so as to get better performance of the structure or the performance of particular applications similar to graph mining as mentioned by (Jiang and Agrawal, 2011), data mining reported by (Papadimitriou and Sun, 2008; Xu *et al*., 2009), genetic algorithms by (Jin *et al*., 2008; Verma *et al*., 2009), or text analysis by (Vashishtha *et al*., 2010) that execute on the framework.

Occasionally, researchers discover the MapReduce structure to be too strict or rigid in its existing implementation. Fadika and Govindaraju (2011) stated that DELMA is one of such a framework which imitates the MapReduce model, identical to Hadoop MapReduce. Such a system is likely to have attractive load balancing problems, which is afar the scope of our paper. One more different framework to MapReduce is Jumbo as reported by (Groot and Kitsuregawa, 2010). The Jumbo framework may be a helpful tool to research load balancing, but it is not well-matched with existing MapReduce technologies. To work around load balancing problems resulting from joining tables in Hadoop, (Lynden *et al*., 2011) introduced an adaptive MapReduce algorithm for several joins using Hadoop that works without changing its setting. This study also attempts to do workload balancing in Hadoop without changing the original structure, but concentrates on sorting text.

Kenn *et al*. (2013) stated that the XTrie algorithm presented a method to advance the cut point algorithm derived from TeraSort. The important issue of the TeraSort algorithm is that to deal with the cut points it utilizes the Quick Sort algorithm. By using quicksort, TeraSort wants to store all the keys it samples in memory and that decreases the probable sample size, which decreases the correctness of the preferred cut points and this affects load balancing mentioned by (O'Malley, 2008). One more difficulty TeraSort has is that it only thinks the first 2 characters of a string during partitioning. This also decreases the efficiency of the TeraSort load balancing algorithm:

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

The main issue derived by TeraSort and XTrie is that they utilize an array to represent the trie. The major concern with this method is that it tends to hold a lot of exhausted space. Kenn *et al*. (2013) also stated that an Algorithm, the ReMap algorithm, which decreases the memory requirements of the original trie by decreasing the number of elements it believes. The ReMap chart

maps each one of the 256 characters on an ASCII chart to the reduced set of elements anticipated by the ETrie. Since the reason of ETrie is to imitate words found in English text ReMap relocates the ASCII characters to the 64 elements. By dropping the number of elements to think from 256 to 64 elements per level, the total memory necessary is reduced to $1/16^{th}$ of its original footprint for a 2-level Trie. So as to use the ETrie, the TrieCode offered in Equation 2 has to be customized. The EtrieCode showing in Equation 3 is alike to the TrieCode in Equation 2, but has been changed to replicate the smaller memory footprint. Even if it is superior to XTrie, the difficulty with this method is that it tends to have a lot of exhausted space. The EtrieCode equation is as follows:

$$HashCode = W_n * 64^{n-1} + W_{n-1} * 64^{n-2} + \ldots\ldots + W_1 * 64^0 = \sum_{n=1}^{TotalWord} W_n * 64^{n-1} \quad (3)$$

The Proposed Method

This section describes the key partitioning as an alternative of hash code partitioning using Horner's Rule which will be incorporated in TeraSort of Hadoop. Besides, this section discusses how memory can be saved by means of a ReMap technique. In accordance with investigational outcome of XTrie and ETrie, the irregular rate is lower, lower being improved, while a trie has more levels. This is since the deeper a trie is the longer the prefix each key symbolizes. So, in this study, full length key is considered as prefix instead of 2 or 3 and the hash value also calculated for the full key.

A trie has 2 advantages when compared with the quick sort algorithm. First, the time complexity for insert and search using the trie algorithm is O (k) where k is the length of the key. In the meantime, the quick sort algorithm best and average case is O (n log n) and in the worst case O (n^2) where n is the number of keys in its sample. Next, a trie has a predetermined memory footprint. This means the number of samples moved into the trie can be enormous if so preferred. In the proposed HTrie algorithm, the HTrie is an array accessed via a HTrie code. A HTrie code is alike to a hashcode, but the codes it generates occur in chronological ASCII order using Horner's Hash Key Rule. The equation for the HTrie code is also a hash code which will use the next prime number as specified by Horner's Rule since the whole key is considered instead of a trie structure. Equation 2 and 3 used 256 and 64 respectively to get the hash code and also provided best value since only 2 or 3 prefixes were considered. So, to get the different as well as good result, the next prime number 37 instead of 31 is used. The equation is as follows:

$$HashCode = W_n * 37^{n-1} + W_{n-1} * 37^{n-2} + \ldots\ldots + W_1 * 37^0 = \sum_{n=1}^{TotalWord} W_n * 37^{n-1} \quad (4)$$

Figure 2 illustrates how the hash code works for a usual partitioner. In this illustration, there are 3 reducers and 3 strings. Each string comes from a key in a (key, value) pair. The first string 'ate' consists of 3 characters 'a', 't' and 'e' and have the equivalent ASCII values. The specific ASCII values are then supplied to Equation 4 to obtain the hash value 137186. Because of 3 reducers, a modulo 3 is used which provides a value 2. Then the value is increased by one in the illustration since there is no reducer 0, which modifies the value to 3. This moved the key-value pair to reducer 3. Using the similar technique, the 2 other strings 'bad' and 'can' are allocated to reducers 2 and 1, correspondingly.

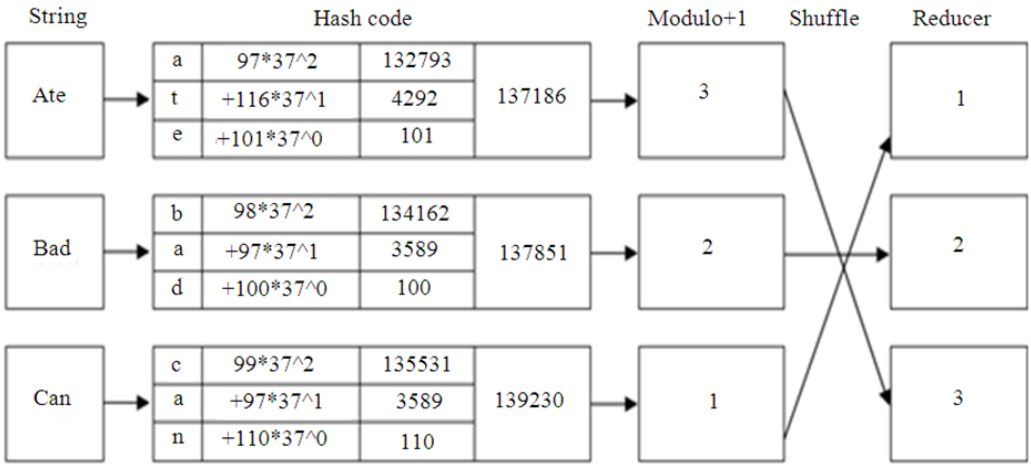

Fig. 2. Proposed Hashcode Partitioner

Results

To estimate the performance of the proposed method, this study examines how fine the algorithms dispense the workload and looks at how fine the memory is used. Tests performed in this study were completed using LastFm Dataset, with each record containing the user profile with fields like country, gender, age and date. Using these records as our input, we simulated computer networks using VMware for Hadoop file system. The tests are carried out with a range of size of dataset such as 1 Lakh, 3 Lakhs, 5 Lakhs, 10 Lakhs, 50 Lakhs and 1 Crore records. During the first experiment, an input file containing 1 lakh records is considered. As mentioned in the MapReduce Framework, the input set is divided into various splits and forwarded to Map Phase. Here for this input file, only one mapper is considered since the number of mappers is depends on the size of the input file. After mapping, partition algorithm is used to reduce the number of output records by grouping records based on Htrie value on the country attribute which is assumed as a key here. After grouping, 4 partitions are created using the procedure Gender-Group-by-Country. All the corresponding log files and counters are analyzed to view the performance. In the other 5 experiments, input files with 3 Lakhs, 5 Lakhs, 10 Lakhs, 50 Lakhs and 1 Crore records are considered. As per the above said method, all the input files are partitioned into 4 partitions.

In order to compare the different methodologies presented in this study and determine how balanced the workload distributions are, this study uses various metrics such as Effective CPU, Rate and Skew among various metrics like clock time, CPU, Bytes, Memory, Effective CPU, Rate and Skew since only the said 3 parameters shows the significant difference in outcomes. Rate displays the number of bytes from the Bytes column divided by the number of seconds elapsed since the previous report, rounded to the nearest kilobyte. No number appears for values less than one KB per second. Effective CPU displays the CPU-seconds consumed by the job between reports, divided by the number of seconds elapsed since the previous report. The result is expressed in units of CPU-seconds per second-a measure of how process or intensive the job is from each report to the next. The skew of a data or flow partition is the amount by which its size deviates from the average partition size:

$$skew\,of\,a\,data = \frac{partition\,size - average\,partition\,size}{size\,of\,largest\,partition} * 100$$

Discussion

The Tables 1-3 shows the results when using various sized input files for the comparison of the performance of ETrie, XTrie and HTrie with the parameters Skew, Effective CPU and Rate respectively. Similarly, the Fig. 3-5 shows comparison chart of the results of the above. From the tables and figures for results, it is shown that the proposed method (HTrie) is performing better than XTrie and ETrie based on all the 3 parameters said above.

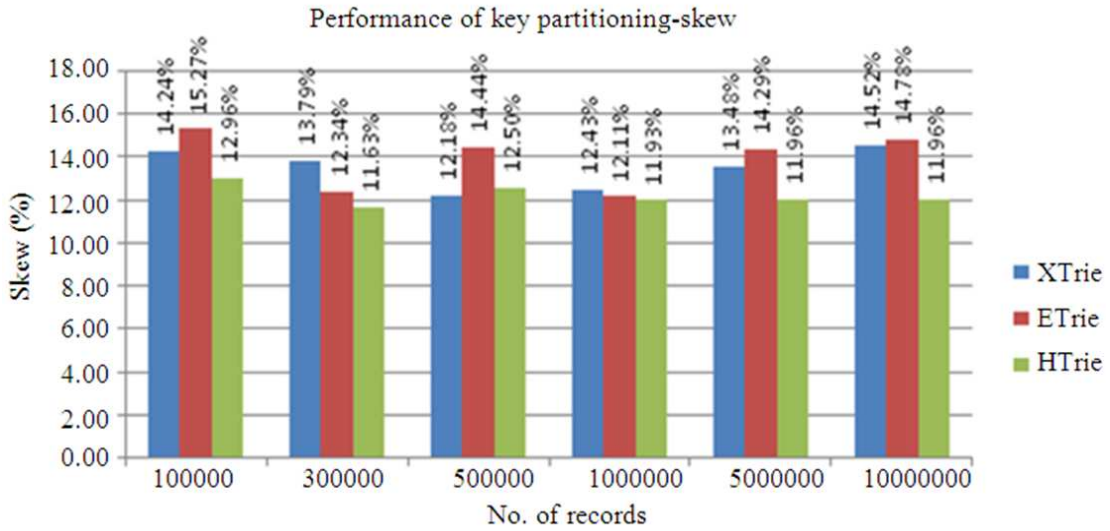

Fig. 3. Comparison chart of skew

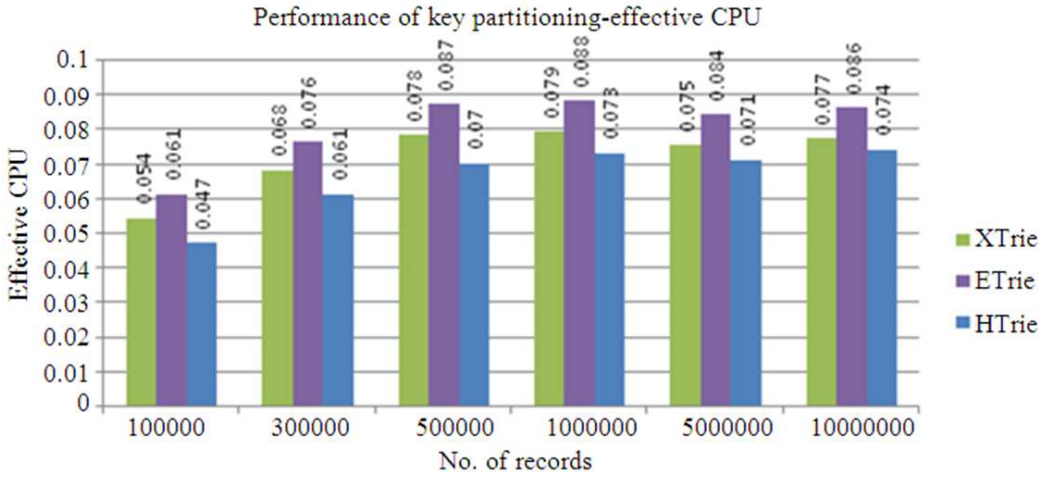

Fig. 4. Comparison chart of effective CPU

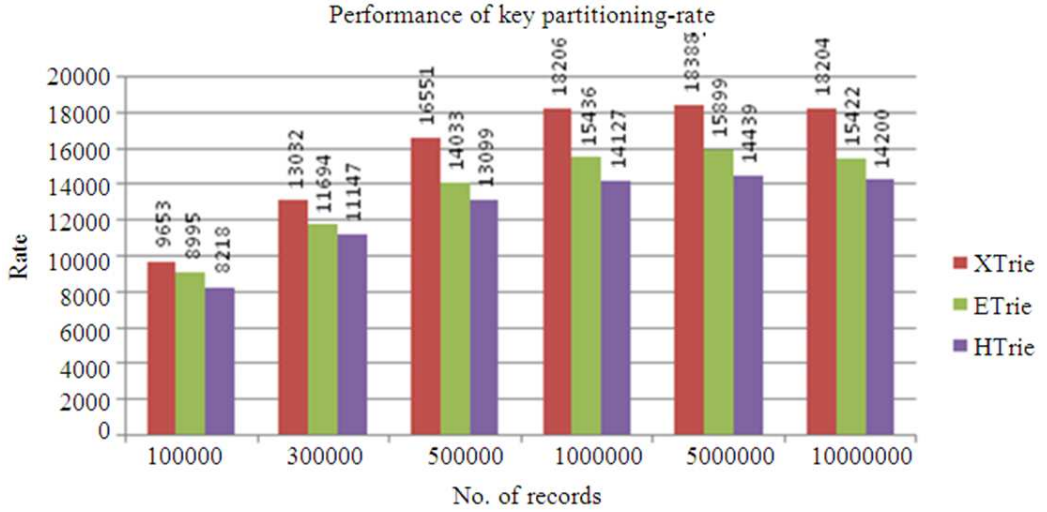

Fig. 5. Comparison chart of rate

Table 1. Comparison of skew

No. of records	XTrie (%)	ETrie (%)	HTrie (%)
100000	14.24	15.27	12.96
300000	13.79	12.34	11.63
500000	12.18	14.44	12.50
1000000	12.43	12.11	11.93
5000000	13.48	14.29	11.96
10000000	14.52	14.78	11.96

Table 2. Comparison of effective CPU

No. of records	XTrie	ETrie	HTrie
100000	0.054	0.061	0.047
300000	0.068	0.076	0.061
500000	0.078	0.087	0.070
1000000	0.079	0.088	0.073
5000000	0.075	0.084	0.071
10000000	0.077	0.086	0.074

Table 3. Comparison of rate

No. of records	XTrie	ETrie	HTrie
100000	9653	8995	8218
300000	13032	11694	11147
500000	16551	14033	13099
1000000	18206	15436	14127
5000000	18388	15899	14439
10000000	18204	15422	14200

Conclusion

This study presented HTrie, comprehensive partitioning technique, to improve load balancing for distributed applications. By means of improving load balancing, MapReduce programs can turn out to be more proficient at managing tasks by reducing the overall computation time spent processing data on each node. The

TeraSort was developed based on arbitrarily generated input data on an extremely huge cluster of 910 nodes. In that specific computing setting and for that data configuration, every partition created by MapReduce became visible on simply one or 2 nodes. But in contrast, our work concentrates at small-sized to medium-sized clusters. This study changes their model and boosts it for a smaller environment. A sequence of experimentations have exposed that given a skewed data sample, the HTrie architecture was capable to safeguard more memory, was capable to distribute more computing resources on average and do so with a lesser amount of time complexity.

After this, additional research can be made to introduce new partitioning mechanisms so that it can be incorporated with Hadoop for applications using different input samples since Hadoop file system is not having any partitioning mechanism except key partitioning.

Acknowledgement

The authors acknowledged Last.fm for providing the access to this data via their web services.

Funding Information

The authors have not approached any funding agencies for funding this work though there various funding agencies were ready to fund this work.

Author's Contributions

Gothai Ekambaram: Planned and designed all the experiments, collected all the necessary data sets, organized the study, implemented all the experiments and contributed in writing this manuscript.

Balasubramanie Palanisamy: Planned and designed all the experiments, collected all the necessary data sets, organized the study, implemented all the experiments and contributed in writing this manuscript along with Gothai Ekambaram as research supervisor.

Ethics

The authors have confirmed that there will not be any ethical issues after publication of this work.

References

Candan, K.S., J.W. Kim, P. Nagarkar, M. Nagendra and R. Yu, 2010. RanKloud: Scalable multimedia data processing in server clusters. IEEE MultiMed, 18: 64-77. DOI: 10.1109/MMUL.2010.70

Chang, F., J. Dean, S. Ghemawat, W.C. Hsieh and D.A. Wallach *et al*., 2008. BigTable: A distributed storage system for structured data. ACM Trans. Comput. Syst., DOI: 10.1145/1365815.1365816

Dean, J. and S. Ghemawat, 2008. MapReduce: Simplified data processing on large clusters. ACM Commun., 51: 107-113. DOI: 10.1145/1327452.1327492

Fadika, Z. and M. Govindaraju, 2011. DELMA: Dynamically ELastic MapReduce framework for CPU-intensive applications. Proceedings of the 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, May 23-26, IEEE Xplore press, Newport Beach, CA., pp: 454-463. DOI: 10.1109/CCGrid.2011.71

Ghemawat, S., H. Gobioff and S.T. Leung, 2003. The Google file system. Proceedings of the 19th ACM Symposium on Operating Systems Principles, (OSP' 03), New York, USA, pp: 29-43. DOI: 10.1145/945445.945450

Groot, S. and M. Kitsuregawa, 2010. Jumbo: Beyond mapReduce for workload balancing. Proceedings of the VLDB PhD Workshop, (PPW' 10), Singapore, pp: 7-12.

Heinz, S., J. Zobel and H.E. Williams, 2002. Burst tries: A fast, efficient data structure for string keys. ACM Trans. Inform. Syst., 20: 192-223. DOI: 10.1145/506309.506312

Hsu, C.H. and B.R. Tsai, 2009. Scheduling for atomic broadcast operation in heterogeneous networks with one port model. J. Supercomput, 50: 269-288. DOI: 10.1007/s11227-008-0261-6

Hsu, C.H. and S.C. Chen, 2010. A two-level scheduling strategy for optimising communications of data parallel programs in clusters. Int. J. Ad Hoc Ubiq. Comput., 6: 263-269. DOI: 10.1504/IJAHUC.2010.035537

Hsu, C.H. and S.C. Chen, 2012. Efficient selection strategies towards processor reordering techniques for improving data locality in heterogeneous clusters. J. Supercomput., 60: 284-300. DOI: 10.1007/s11227-010-0463-6

Hsu, C.H., S.C. Chen and C.Y. Lan, 2007. Scheduling contention-free irregular redistributions in parallelizing compilers. J. Supercomputing, 40: 229-247. DOI: 10.1007/s11227-006-0024-1

Hsu, C.H., T.L. Chen and J.H. Park, 2008. On improving resource utilization and system throughput of master slave job scheduling in heterogeneous systems. J. Supercomput., 45: 129-150. DOI: 10.1007/s11227-008-0211-3

Jiang, W. and G. Agrawal, 2011. Ex-MATE: Data intensive computing with large reduction objects and its application to graph mining. Proceedings of the 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, May 23-26, IEEE Xplore Press, Newport Beach, CA., pp: 475-484. DOI: 10.1109/CCGrid.2011.18

Jin, C., C. Vecchiola and R. Buyya, 2008. MRPGA: An extension of mapReduce for parallelizing genetic algorithms. Proceedings of the IEEE 4th International Conference on e-Science, Dec. 7-12, IEEE Xplore press, Indianapolis, IN, pp: 214-221. DOI: 10.1109/eScience.2008.78

Kavulya, S., J. Tan, R. Gandhi and P. Narasimhan, 2010. An analysis of traces from a production mapreduce cluster. Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, May 17-20, IEEE Xplore Press, Melbourne, VIC., pp: 94-103. DOI: 10.1109/CCGRID.2010.112

Kenn, S., C.H. Hsu, Y.C. Chung and D. Zhang, 2013. An improved partitioning mechanism for optimizing massive data analysis using mapReduce. J. Supercomput., 66: 539-555. DOI: 10.1007/s11227-013-0924-9

Krishnan, A., 2005. GridBLAST: A globus-based high-throughput implementation of blast in a grid computing framework. Concurr Comput., 17: 1607-1623. DOI: 10.1002/cpe.906

Liu, H. and D. Orban, 2011. Cloud mapReduce: A mapreduce implementation on top of a cloud operating system. Proceedings of the 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, May 23-26, IEEE Xplore Press, Newport Beach, CA., pp: 464-474. DOI: 10.1109/CCGrid.2011.25

Lynden, S., Y. Tanimura, I. Kojima and A. Matono, 2011. Dynamic data redistribution for mapReduce joins. Proceedings of the IEEE 3rd International Conference on Cloud Computing Technology and Science, Nov. 29-Dec. 1, IEEE Xplore press, Athens, pp: 717-723. DOI: 10.1109/CloudCom.2011.111

Miceli, C., M. Miceli, S. Jha, H. Kaiser and A. Merzky, 2009. Programming abstractions for data intensive computing on clouds and grids. Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, May 18-21, IEEE Xplore Press, Shanghai, pp: 478-483. DOI: 10.1109/CCGRID.2009.87

O'Malley, O., 2008. TeraByte sort on Apache Hadoop.

Panda, B., M. Riedewald and D. Fink, 2010. The model-summary problem and a solution for trees. Proceedings of the IEEE 26th International Conference on Data Engineering, Mar. 1-6, IEEE Xplore Press, Long Beach, CA, pp: 449-460. DOI: 10.1109/ICDE.2010.5447912

Papadimitriou, S. and J. Sun, 2008. DisCo: Distributed co-clustering with map-reduce: A case study towards petabyte-scale end-to-end mining. Proceedings of the 8th IEEE International Conference on Data Mining, Dec. 15-19, IEEE Xplore Press, Pisa, pp: 512-521. DOI: 10.1109/ICDM.2008.142

Stockinger, H., M. Pagni, L. Cerutti and L. Falquet, 2006. Grid approach to embarrassingly parallel CPU-intensive bioinformatics problems. Proceedings of the 2nd IEEE International Conference on e-Science and Grid Computing, Dec. 4-6, IEEE Xplore Press, Amsterdam, Netherlands, pp: 58-58. DOI: 10.1109/E-SCIENCE.2006.261142

Tan, J., X. Pan, S. Kavulya, R. Gandhi and P. Narasimhan, 2009. Mochi: Visual log-analysis based tools for debugging hadoop. Proceedings of the USENIX Workshop on Hot Topics in Cloud Computing (HotCloud), (TCC' 09), USENIX, San Diego, CA. DOI: 10.1.1.149.881

Vashishtha, H., M. Smit and E. Stroulia, 2010. Moving text analysis tools to the cloud. Proceedings of the 6th World Congress on Services, Jul. 5-10, IEEE Xplore Press, Miami, FL., pp: 107-144. DOI: 10.1109/SERVICES.2010.91

Verma, A., X. Llora, D.E. Goldberg and R.H. Campbell, 2009. Scaling genetic algorithms using mapReduce. Proceedings of the 9th International Conference on Intelligent Systems Design and Applications, Nov. 30-Dec. 2, IEEE Xplore Press, Pisa, pp: 13-18. DOI: 10.1109/ISDA.2009.181

Xu, W., L. Huang, A. Fox, D. Patterson and M.I. Jordan, 2009. Detecting large-scale system problems by mining console logs. Proceedings of the 22nd Symposium on Operating Systems Principles, Oct. 11-14, New York, pp: 117-132. DOI: 10.1145/1629575.1629587

Zaharia, M., A. Konwinski, A.D. Joseph, R. Katz and I. Stoica, 2008. Improving mapReduce performance in heterogeneous environments. Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation, USENIX, San Diego, California, USA, pp: 29-42.