A NOVEL APPROACH FOR PARTITIONING IN HADOOP USING ROUND ROBIN TECHNIQUE

[1]GOTHAI E, [2]BALASUBRAMANIE P

[1]Associate Professor, Department of CSE, Kongu Engineering College, Erode-638052, Tamilnadu, India

[2]Professor, Department of CSE, Kongu Engineering College, Erode-638052, Tamilnadu, India

Email: [1]kothaie@yahoo.co.in , [2]pbalu_20032001@yahoo.co.in

ABSTRACT

The Hadoop Distributed File System is constructed to store profoundly and immensely colossal data sets accurately and to send those data sets at huge bandwidth to end user applications. Hadoop gives a distributed file system and a structure for the analysis and conversion of profoundly and astronomically immense data sets utilizing the MapReduce paradigm. A paramount characteristic of Hadoop is the partitioning of data and computation across many of hosts and the execution of application computations in parallel proximate to their data. This paper recommends an enhanced partitioning algorithm utilizing round robin partitioning that advances load balancing and recollection utilization. A sequence of experimentations have exposed that given a skewed data sample, the Round Robin architecture was capable to reduce skew by distributing records on average when compared with subsisting Hash Partitioning. Experimentations demonstrate that the proposed method is efficient and more precise than the subsisting implementation.

KEYWORDS: *Hadoop, Round Robin, Partitioning, Mapreduce*

1. INTRODUCTION

The Hadoop Distributed File System (HDFS) [1] is constructed to store profoundly and immensely colossal data sets accurately and to send those data sets at huge bandwidth to end user applications. Hadoop gives a distributed file system and a structure for the analysis and conversion of profoundly and astronomically immense data sets utilizing the MapReduce paradigm. A paramount characteristic of Hadoop is the partitioning of data and computation across many of hosts and the execution of application computations in parallel proximate to their data. Today's most prosperous companies use data to their advantage [2]. The data are no longer facilely quantifiable facts, such as point of sale transaction data. Relatively these companies retain, explore, analyze, and manipulate all the available information in their purview. Ultimately, they probe for evidence of facts, insights that lead to incipient business opportunities or which leverage their subsisting strengths. This is the business value abaft what is often referred to as astronomically immense Data. Hadoop has proven to be a technology apposite to tackle astronomically Immense Data quandaries. The fundamental principle of the Hadoop architecture is to move analysis to the data rather than moving the data to a system that can analyze it. Ideally, Hadoop capitalizes on the advances in commodity hardware to scale in the way companies want.

There are two key technologies that sanction users of Hadoop to prosperously retain and analyze data: HDFS and MapReduce [3]. HDFS is a simple but astronomically potent distributed file system. It is able to store data reliably at consequential scale. HDFS deployments subsist with thousands of nodes storing hundreds of petabytes of utilizer data. MapReduce is parallel programming framework that integrates with HDFS. It sanctions users to express data analysis algorithms in terms of a minuscule number of functions and operators, chiefly, a map function and a reduce function. The prosperity of MapReduce is a testament to the robustness of HDFS - both as a system to renovate and access data, and as an application programming interface (API) for immensely colossal Data analysis frameworks. While MapReduce is convenient when performing scheduled analysis or manipulation of data stored on HDFS, it is not congruous for interactive use: it is too slow and lacks the expressive power required.

The MapReduce programming [4] model has been prosperously utilized at Google for many different purposes. This prosperity attributed to several reasons. First, the model is facile to utilize, even for programmers without experience with parallel and distributed systems, since it hides the details of parallelization, reliability, load balancing and locality optimization. Second, an immensely colossal variety of quandaries are facilely expressible as MapReduce computations. For example, MapReduce is utilized for the generation of data for Google's engenderment web search accommodation, for sorting, for data mining, for machine learning, and many other systems. Third, they have developed an implementation of MapReduce that scales to immensely colossal clusters of machines comprising thousands of machines. The implementation makes efficient utilization of these machine resources and therefore is felicitous for use on many of the astronomically immense computational quandaries encountered at Google. They have learnt several things from this work. As a first, controlling the programming model makes it too easy to parallelize and distribute computations and to create such computations reliable. Next, network bandwidth is an inadequate resource. A number of optimizations in their system were therefore embattled at reducing the amount of data thrown across the network: the locality optimization approves them to read data from local disks, and creating a single replica of the intermediate data to local disk preserves network bandwidth. Third, redundant execution can be adapted to reduce the impact of slow machines and to handle machine failures and data loss.

MapReduce has emerged as a popular implement for distributed and scalable processing of massive data sets and is being used increasingly in e-science applications. Lamentably the performance of MapReduce systems vigorously depends on an even data distribution while scientific data sets are often extremely distorted. The ensuing load inequality which raises the processing time is even amplified by high runtime intricacy of the reducer tasks. An adaptive load balancing policy is necessary for opportune skew handling. In [5], the authors addressed the quandary of estimating the cost of the tasks that are distributed to the reducers predicated on a given cost model. Precise cost estimation is the substructure for adaptive load balancing algorithms and requires accumulating statistics from the mappers. There are some challenges such as, 1. Since the statistics from all mappers must be integrated; the mapper statistics must be diminutive. 2. The integrated statistics must capture the global data distribution albeit each mapper visually perceives only a minute fraction of the data. 3. The mappers terminate after sending the statistics to the controller and no second round is possible. Their resolution to these challenges has two components. 1. A monitoring component executed on every mapper captures the local data distribution and identifies its most pertinent subset for cost estimation. 2. An integration component aggregates these subsets approximating the global data distribution.

2. BACKGROUND

2.1. HDFS Architecture

HDFS [6] is a distributed file system that gives high throughput access to data. All the files are divided into blocks of fixed size and stored on datanodes. The block size is configurable and defaults to 64MB. Files can be written only once, i.e., updates of existing files are not allowed. The HDFS namenode keeps track of the directory structure of the file system. It also maintains a list of active datanodes as well as their data blocks in a dynamic data structure called BlockMap. Whenever a datanode starts up, it registers itself at the namenode with the list of blocks in its storage; these blocks are added to the namenode's BlockMap. Whenever the namenode detects failure of a datanode, the blocks of the failed node are removed from the BlockMap. Datanodes can both send blocks to clients upon request, but also store new blocks sent by the client. This process is coordinated by the namenode, which directs clients to the correct datanodes. HDFS can be configured to replicate files for fast recovery in the case of failures. The default replication factor is three which means that a block is stored on three separate datanodes. HDFS uses a simple data placement policy to select the datanodes that store the blocks and replicas of a file. The default policy of HDFS places the first copy of a newly created block on the local datanode at which the block is created and this called write affinity. Then HDFS attempts to select a datanode within the same rack for the second copy and a datanode in a different rack for the third copy. They have modified this data placement policy to support collocation as explained. Hadoop uses so called InputFormats to define how files are split and consumed by the map tasks. Several InputFormats are provided with Hadoop. Input formats that operate on files are based on an abstract type called FileInputFormat. When starting a Hadoop job the FileInputFormat is provided with a path containing the files to process. It then divides these files into one or more splits which constitute the unit of work for a single map task in a MapReduce program. By default the various FileInputFormat implementations break a file into 64 MB chunks. The Hadoop scheduler attempts its best to schedule map tasks on nodes that have a local copy of their splits. InputFormats provide an extensibility point that users can exploit to control the distribution of data to map tasks by assembling the customized splits.

2.2. Partitioning

There are typically three approaches to partitioning database records [7]: Range, Round-Robin and Hash.

Range partitioning places categorical ranges of table ingresses on different disks. To understand range partitioning, consider a long list of value. Range partitioning breaks the long list into several shorter manageable lists and spans them across multiple disks. Another example might be a system managing monthly operations might partition each month onto a different set of disks. In cases where only a portion of the data is utilized in a query - let's verbally express the M-P range the database can avoid examining the other sets of data in what is kenned as partition elimination. This can dramatically reduce the time to consummate a query.

Round-robin partitioning evenly distributes records across all disks that compose a logical space for the table, without regard to the data values being stored. This sanctions even workload distribution for subsequent table scans. Disk striping accomplishes identically tantamount result spreading read operations across multiple spindles but with the logical volume manager, not the DBMS, managing the striping.

Hash partitioning is a third method of distributing DBMS data evenly across the set of disk spindles. A hash function is applied to one or more database keys, and the records are distributed across the disk subsystem accordingly. Again, a drawback of hash partitioning is that partition elimination may not be possible for those queries whose performance could be amended with this technique.

3. RELATED WORKS

CoHadoop, a lightweight solution for collocating cognate files in HDFS is presented in [8]. Their approach to colocation is simple yet flexible; it can be exploited in different ways by different applications. They identified two use cases—join and sessionization—in the context of log processing and described map-only algorithms that exploit colocated partitions. They studied the performance of CoHadoop under different settings and compared it with both plain Hadoop solutions and map-only algorithms that work on partitioned data without colocation. Their experiments divine that copartitioning and colocation together provide the best performance. Both theoretical analysis and experiments suggest that CoHadoop maintains the fault tolerance characteristics of Hadoop to an immensely colossal extent.

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 [9], how it is communicated [10], what background it is being located on [11, 12, 13] 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 [14, 15]. RanKloud [16] make use of its personal uSplitmethod for partitioning huge media data sets. The uSplitmethod 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 [17] was launched by Google to handle structured data. 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 sharednothing 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 [18, 19, 20].

To work around load balancing problems resulting from joining tables in Hadoop, [21] introduces an adaptive MapReduce algorithm for several joins using Hadoop that works without changing its setting. This paper also attempts to do workload balancing in Hadoop without changing the original structure, but concentrates on distributing the data evenly among partitions using Round Robin partitioning.

4. PROPOSED APPROACH

In Hash Partitioning approach presented in the previous work of the authors, the partitioning technique that is used when the keys are diverse, large data skew can exist when the key is present in large volume and it is apt for parallel data processing. But Round Robin partition technique uniformly distributes the data on every destination data partitions and when number of records is divisible by number of partitions, and then the skew is most probably zero. For example a pack of 52 cards is distributed among 4 players in a round-robin fashion. In order to evaluate the performance of round robin partitioning in Hadoop, this partitioning is implemented in Hadoop even though this partitioning is not available in Hadoop. This section describes the round robin partitioning as an alternative of hash partitioning which will be incorporated in Hadoop. Besides, this section discusses how memory can be saved by means of a ReMap technique.

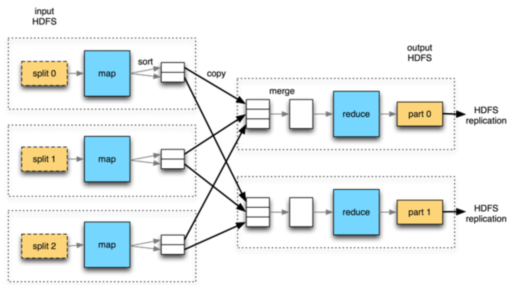

Figure 1: MapReduce Dataflow

As designated in Figure 1, the data splits are applied to the Mapper and the outcome is sorted splits. Further these splits are facsimiled to the splits of Reducer for merging. During facsimileing, the proposed round robin portioning is incorporated. The partitioning is done as designated in Figure 2. After that, the reducer does its work and engenders the final partitions.

5. RESULTS AND DISCUSSION

To estimate the performance of the proposed method, this work examines how fine the algorithms dispense the workload, and looks at how

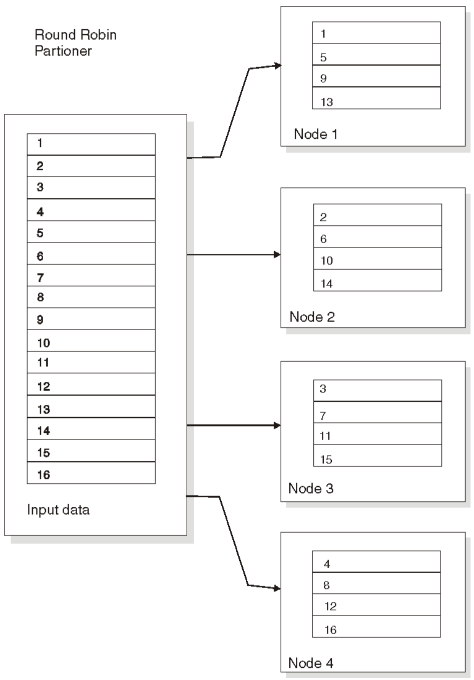

Figure 2: Round Robin Framework

fine the memory is used. Tests performed in this paper 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, they 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 in round robin fashion. 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 paper and determine how balanced the workload distributions are, this study uses various metrics such as Effective CPU, Rate and Skew among various metrics, since only these 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. Effective CPU displays the CPU-seconds consumed by the job between reports, divided by the number of seconds elapsed since the previous report. The skew of a data or flow partition is the amount by which its size deviates from the average partition size, expressed as a percentage of the largest partition.

The tables 1, 2 and 3 shows the results when using various sized input files for the comparison of the performance of existing Hash partitioning and proposed Round Robin partitioning with the parameters Skew, Effective CPU and Rate respectively. Similarly, the figures 3, 4 and 5 shows comparison chart of the results of the above. From the tables and figures for results, it is shown that the proposed method is performing better than Hash Partitioning based on a parameter skew but not in other 2 parameters said above.

Table 1: Performance Comparison of Skew

No. of records	Hash	Round Robin
100000	12.96%	0.50%
300000	11.63%	1.74%
500000	12.50%	1.49%
1000000	11.93%	1.79%
5000000	11.96%	0.85%
10000000	11.96%	0.54%

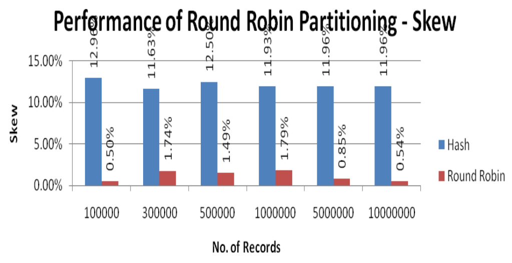

Figure 3: Comparison Chart of Skew

Table 2: Performance Comparison of Rate

No. of records	Hash (in kb)	Round Robin (in kb)
100000	8218	9040
300000	11147	12596
500000	13099	15064
1000000	14127	15822
5000000	14439	16460
10000000	14200	15620

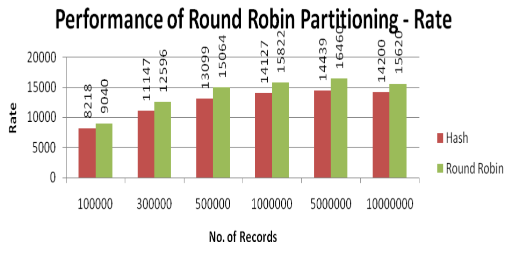

Figure 4: Comparison Chart of Rate

Table 3: Performance Comparison of Effective CPU

No. of records	Hash (in sec)	Round Robin (in sec)
100000	0.047	0.052
300000	0.061	0.069
500000	0.07	0.081
1000000	0.073	0.082
5000000	0.071	0.081
10000000	0.074	0.083

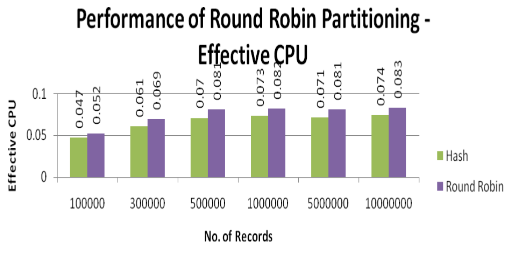

Figure 5: Comparison Chart of Effective CPU

6. CONCLUSION

This paper presented Round Robin, a 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. Our work concentrates at small-sized to medium-sized clusters rather than large clusters. This study changes existing model of hash partitioning and boosts it for a smaller environment with round robin partitioning. A sequence of experimentations have exposed that given a skewed data sample, the Round Robin architecture was capable to reduce skew by distributingrecords on average when compared with existing Hash Partitioning. After this, additional research can be made to introduce few other 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 hash key partitioning.

REFERENCES:

[1] www.aosabook.org/en/hdfs.html

[2] www.gopivotal.com/sites/default/files/Hawq_WP_042313_FINAL.pdf

[3] www.hadoophdfs.com/tutorials

[4] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters", *Proceedings of the 6th OSDI Symposium*, 2004.

[5] B. Gufler, N. Augsten, A. Reiser, and A. Kemper, "Load balancing in MapReduce based on scalable cardinality estimates", *Proceedings of the 28th ICDE Conference*, 2012.

[6] The Apache Software Foundation, "HDFS architecture guide", http://hadoop.apache.org/hdfs/docs/current/hdfs_design.html.

[7] www.infonitive.com

[8] Y. Mohamed, Eltabakh, Yuanyuan Tian, Fatma Ozcan, Rainer Gemulla, Aljoscha Krettek, and John McPherson, "CoHadoop: Flexible Data Placement and Its Exploitation in Hadoop", *Proceedings of the 37th International Conference on Very Large Data Bases*, August 29th September 3rd 2011, Seattle, Washington. Proceedings of the VLDB Endowment, Vol. 4, No. 9, 2011.

[9] C-H. Hsu and S-C. Chen, "Efficient Selection Strategies towards Processor Reordering Techniques for Improving Data Locality in Heterogeneous Clusters", *Journal of Supercomputing*, Vol. 60, 2012, pp.284–300.

[10] C-H. Hsu and S-C. Chen, "A Two-level Scheduling Strategy for Optimizing Communications of Data Parallel Programs in Clusters", *International Journal of Ad Hoc UbiqComputing,* Vol.6, 2010, pp. 263–269.

[11] C-H. Hsu and B-R. Tsai, "Scheduling for Atomic Broadcast Operation in Heterogeneous Networks with One Port Model", *Journal of Supercomputing*, Vol. 60, 2009, pp. 269–288.

[12] C-H. Hsu, T-L. Chen and J-H. Park, "On Improving Resource Utilization and System Throughput of Master Slave Jobs Scheduling in Heterogeneous Systems", *Journal of Supercomputing*, Vol. 45, 2008, pp. 129–150.

[13] M. Zaharia, A. Konwinski, AD. Joseph, R. Katz and I. Stoica, "Improving MapReduce Performance in Heterogeneous Environments", *Proceedings of 8th USENIX Symposium on Operating Systems Design and Implementation,* 8-10 December 2008, San Diego, California, USA, USENIX, pp. 29-42.

[14] A. Krishnan, "GridBLAST: A Globus-based High-throughput Implementation of BLAST in a Grid Computing Framework", *Concurrent Computing,* Vol. 17, 2005, pp. 1607–1623.

[15] H. Stockinger, M. Pagni, L. Cerutti and L. Falquet, "Grid Approach to Embarrassingly Parallel CPU Intensive Bioinformatics Problems", *Proceedings of IEEE International Conference on eScience and Grid Computing*, December 2006, Amsterdam, The Netherlands, pp. 58.

[16] KS. Candan, JW. Kim, P. Nagarkar, M. Nagendra and R. Yu, "RanKloud: Scalable Multimedia Data Processing in Server Clusters", *IEEE MultiMed*, Vol. 18, 2010, pp. 64–77.

[17] F. Chang, J. Dean, S. Ghemawat, WC Hsieh, DA. Wallach, M. Burrows, T. Chandra, A. Fikes and RE. Gruber, "Bigtable: A Distributed Storage System for Structured Data", *ACM Transactions on Computer Systems*, Vol. 26, 2008.

[18] H. Liu and D. Orban, "Cloud MapReduce: A MapReduce Implementation on Top of a Cloud Operating System", *Proceedings of IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, May 2011, CA, USA, IEEE/ACM, pp. 464–474.

[19] A. Matsunaga, M. Tsugawa and J. Fortes, "Programming Abstractions for Data Intensive Computing on Clouds and Grids", *Proceedings of IEEE Fourth International Conference on eScience*, 7-12 December 2008, Indiana, USA, IEEE, pp. 489–493.

[20] C. Miceli, M. Miceli, S. Jha, H. Kaiser and A. Merzky, "Programming Abstractions for Data Intensive Computing on Clouds and Grids", *Proceedings of IEEE/ACM International Symposium on Cluster Computing and the Grid*, 18-21 May 2009, USA, IEEE/ACM, pp.480–483.

[21] S. Lynden, Y. Tanimura, I. Kojima and A. Matono, "Dynamic Data Redistribution for MapReduce Joins", *Proceedings of IEEE International Conference on Cloud Computing Technology and Science*, 29 November – 1 December 2011, Athens, Greece, IEEE, pp. 713–717.