Data Balancing Scheme for Multi-node Heterogeneous Hadoop Cluster

Indresh B. Rajwade[1], Er. Prateek Singh[2]

[1]M. Tech. (CSE) Student, Department of Computer Science & IT, SHUATS

[2]Assistant Professor, Department of Computer Sciece& IT, SHUATS

Abstract: ***Big data encompasses huge amount of information from multiple internal and external resources such as transactions, social media, enterprise content, sensors and mobile devices. It is characterized as volume, velocity, variety and veracity. MapReduce is a parallel computing framework which meets the tremendous needs for large scale data processing. Due to its simplicity, robustness and scalability MapReduce has been widely used by the companies such as Amazon, Facebook and Yahoo! to process large volumes of data on a daily basis. The MapReduce framework simplifies the complexity of running distributed data processing functions across multiple nodes in a cluster. It automatically handles the gathering of results across the multiple nodes and returns a single result or a set. Hadoop is an open source implementation of MapReduce which balances the load in a cluster by distributing data to multiple nodes based on disk space availability and processing efficiency. In this dissertation, the evaluation of data placement mechanism in a heterogeneous Hadoop cluster is performed using Grep tool and WordCount program. These are two MapReduce applications running on Hadoop clusters. A comparison has been done with Grep and WordCount through Ubuntu 14.04 LTS for three nodes in a Hadoop cluster and it is observed that the computing ratios of a Hadoop cluster are application dependent and size independent. This means that if the configuration of a cluster is updated, computing ratios must be determined again.***

Keywords: Big Data, Hadoop, MapReduce, HDFS, Grep, WordCount, Heterogeneous cluster

1. Introduction

Today, the most technological challenges in software systems research is to provide solutions for storing, manipulating, and information retrieval on large set of data. Web services and social media (Facebook, Twitter etc.) produce together a huge amount of data, reaching the scale of petabytes daily [1]. These large set of data may consist of valuable information, which sometimes is not properly discovered by existing systems. Most of this data is stored in a unstructured fashion, using different languages and formats, which, in many cases, are not compatible [2].

For example, Facebookthat initially used relational database management systems (RDBMS) to store its data. Because of the growing large volume of information generated on a daily basis (2.5 Quintillion bytes) [3]. One of the Facebook‟s largest clusters holds more than 100 PB of data, processing more than 60000 queries a day [1]. The use of traditional platform became impracticable particularlybecause, most of its data is unstructured, consisting of logs, posts, photos, and pictures [4]. Having achieved in April 2016 more than 1.59 billion globallymonthly active users [5], Facebook may be considered one of the largest and most valuable social networks. Big companies like Google, Facebook and Yahoo capturing huge amount of user data started to be evaluated not just by their applications but also by their large datasets, particularly the information that can be extracted from them. The companies have an aggregate value for their provided services and for the large amount of information stored. This data can be used for numerous future applications, such as IT infrastructure optimization, manufacturing process optimization, legal discovery, social network analysis, traffic flow optimization, web app optimization, integration of location-based information, churn analysis, natural resource exploration, weather forecasting, healthcare, fraud detection, life science research, advertising analysis etc. [6].

1.1 Big Data

The data is being generated from various sources- transactions, social media, sensors, digital images, videos, audios and click streams for domains including healthcare, retail energy and utilities. In addition to business and organizations, individual contribute to the data volume. For instance, billionsof content are being shared on Facebook every month [5].

International Data Corporation (IDC) terms this as the „Digital Universe‟ and predicts that this digital universe is set to explode to an unimaginable 8 Zeta bytes by the year 2015 [7]. The term „Big Data‟ was coined to address the massive volume of data storage and processing. The Big Data is a huge amount of unstructured data generated by companies, organisations and individual every day and these data are stored in digital form.

The "Big Data" term is used to refer to a collection of large datasets that may not be processed using traditional database management tools [8][9]. There are challenges involved to deal with large datasets such processing, storage, and analysis etc. Regarding data analysis and Big Data, the need for infrastructures capable of processing large amounts of data, within an passable time and on limited resources, is a substantial problem. Probable solutions make use of parallel and distributed computing. This model of computation has been demonstrated to be essential nowadays to extract relevant information from Big Data. Such processing is accomplished using clusters and grids, which use generally, commodity hardware to combined computational capacity at a relatively low cost.

1.2 Hadoop

Hadoop is software framework for distributed processing of large datasets across large clusters of computers. It is a popular open-source implementation of the Google‟s MapReduce algorithm primarily developed by Yahoo [10]. It is used to generate hundreds of TB of data by Yahoo‟s servers on more than40,000 processor cores [11].Hadoop is used by most popular company Facebook to process more than 300 PB of new data every day. Other than Yahoo and Facebook, wide range of websites like Amazon and Last.fm are usingHadoop to manage large set of data per day [12]. In addition to data-intensive applications pertaining to web, scientific data-intensive applications such as seismic simulations and natural language processing take full benefits from the Hadoop system [13][12].

1.3 Hadoop Infrastructure

Essentially, there are two major layers of Hadoop system. In Figure 1, the first layer is the HadoopMapReduce engine for processing large datasets[14]. Higher levels in the software stack consists of Pig and Hive, user-friendly parallel data processing languages, Zoomkeeper a high-availability directory and configuration service, and HBase, a web-scale distributed column-oriented store designed after its proprietary predecessors. The second layer is Hadoop Distributed File System (HDFS) [15]. Currently, HDFS divides files into blocks that are replicated among several different computing nodes with no attention to whether the blocks are divided evenly. When a job is initiated, the processor of each node works with the data on their local hard disks. In the initial phase of this dissertation research, it is investigated that how Hadoop works with its parallel file system. It divides a large file into small pieces, which are evenly distributed across multiple nodes. When the large file is accessed, high aggregated I/O bandwidth can be achieved by accessing the multiple nodes in parallel. The performance of cluster can be improved by Hadoop, because multiple nodes work concurrently to provide high throughput.

1.4MapReduce

The MapReduce programming model simplifies the complexity of running parallel data processing functions across multiple computing nodes in a cluster, by allowing a programmer with no specific knowledge of parallel programming to create MapReduce functions running in parallel on the cluster. MapReduce automatically handles the gathering of results across the multiple nodes and returns a single result or set. More importantly, the MapReduce runtime system offers fault tolerance that is entirely transparent to programmers.

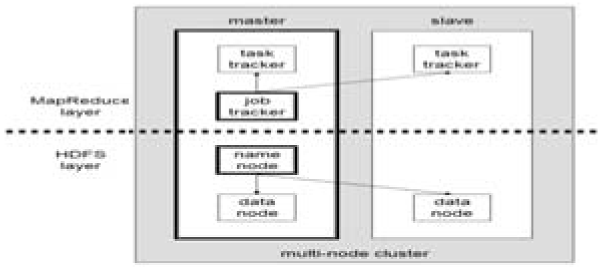

Figure 1: Hadoop Infrastructure

1.5 MapReduce Framework Architecture

The MapReduce is programming model and an associated implementation for processing and generating large data sets.

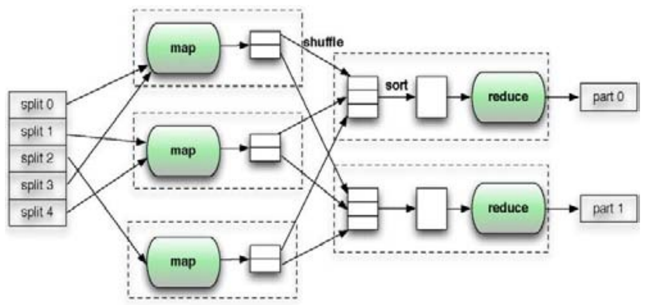

Figure 2: MapReduce Framework

Computations in MapReduce framework are divided into map and reduce phases, separated by an internal grouping of the intermediate results. The power of MapReduce is that the map and reduce functions are executed in parallel over hundreds or thousands of processors with minimal effort by the user. After user submits a job, MapReduce jobs run as follows. Firstly, input data is divided into several fixed-size splits, each of which runs a map task. Then after all map tasks are finished, the intermediate data is reassigned to reduce tasks according to different keys generated in map phase. Reduce phase can be divided into three parts, shuffle, sort and reduce function. The shuffle phase transfers intermediate data generated by map tasks to the corresponding reduce tasks. The sort phase merges all the intermediate data belonging to the reduce task and maintains sort. Then reduce function is called for each key in the sort phase, and directly writes output to distributed file system.

The MapReduce framework operates on (key, value) pairs, that is, the framework views the input to the job as a set of (key, value) pairs and produces a set of (key, value) pairs as the output of the job, conceivably of different types.

1.6 HDFS

The Hadoop Distributed File System or HDFS is a distributed file system designed to run on commodity hardware. HDFS is the primary distributed storage used by Hadoop applications on clusters. Although HDFS has many similarities with existing distributed file systems, the differences between HDFS and other systems are significant. HDFS is highly fault-tolerant and is designed to be deployed on cost-effective clusters. HDFS offers high throughput access to application data and is suitable for applications that have large data sets.

1.7 HDFS Architecture

Figure 3 represents a diagram that describes the architecture ofHDFS. Diagram describes the master-slave architecture, in which a master is called NameNode and slaves are referred to as DataNodes [16]. Basically, an HDFS cluster consists of a single NameNode, which manages the file system namespace and regulates access of clients to files. In addition, there are a number of DataNodes. Usually, each node in a cluster has one DataNode that manages storage of the node on which tasks are running. HDFS exposes file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks stored in a set of DataNodes. The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. The NameNode also determines the mappings of blocks to DataNodes. The DataNodes not only are responsible for serving read and write requests issued from the file system‟s clients, but also perform block creation, deletion, and replication upon instructions from the NameNode [17].

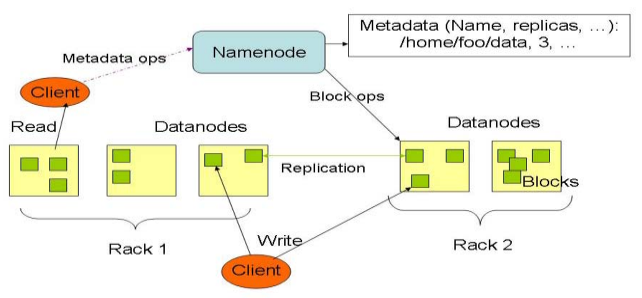

Figure 3: HDFS Architecture

HDFS is designed to support very large files, because Hadoop applications are dealing with large data sets. These Hadoop applications write their data only once but read the data one or more times and require these reads to be performed at streaming speeds. HDFS supports write-once-read-many semantics on files. A typical block size used by HDFS is 64 MB/128MB, thus an HDFS file is chopped up into 64 MB/128MB chunks. If it is possible, chunks are residing on different DataNodes.

2. Related Work

Zaharia et al. proposed the FAIR scheduler, optimized for multi-user environments, in which a single cluster is shared among a number of users. The FAIR algorithm is used in the data mining research field to analyze log files. The FAIR scheduler aims to reduce idle times of short jobs, thereby offering fast response times of the short jobs. The scheduler in Hadoop organizes jobs into pools, among which resources are shared. Each pool is assigned a guaranteed minimum share, which ensures that certain users or applications always get sufficient resources. Fair sharing can also work with job priorities, which are used as weights to determine the fraction of total compute time allocated to each job. Fair scheduling assigns resources to jobs so that all jobs consume, on average, an equal share of resources. They also address the problem of speculative execution of straggling tasks while still concerned with performance [18].

Leo and Zanetti implemented a solution to make Hadoop available to Python programmers called Pydoop. A Python package based on CPython provides an API for MapReduce and HDFS. This works as an alternative to Hadoop Streaming or Jython. Hadoop Streaming uses a communication protocol to execute a Python script as the Mapper or Reducer via the standard input and output [19]. Therefore, it cannot process arbitrary data streams, and the user directly controls only the Map and Reduce parts, except for HDFS operations.

Shvachko et al. proposed an Hadoop Distributed File System is the block storage layer that Hadoop uses to keep its files. HDFS was designed to hold very large datasets reliably using data replication [20]. This allows HDFS to stream large amounts of data to user applications in a reasonable time. Its architecture is composed of two main entities: NameNode and DataNodes, which work in a master-slave fashion. NameNode is responsible for keeping the metadata about what and where the files are stored in the file system. DataNodes are responsible for storing the data itself. HDFS works as a single-writer, multiple-reader file system.

Vernica et al. describes solutions to improve Hadoop's performance. They focus on the interaction of Mappers, introducing an asynchronous communication channel between Mappers. In the current implementation of Hadoop, Mappers are completely independent. Using a transactional distributed meta-data store (DMDS), Mappers can post metadata about their state and check the state of all other Mappers [21].

Ahmad et al. proposed MaRCO (MapReduce with communication overlap), which is directed to the overlapping of the Shuffle with the Reduce computation. The original Hadoop data flow was modified allowing the operation of Reduce tasks on partial data. MaRCO breaks Reduce into many smaller invocations on partial data from some map tasks, and a final reducing step re-reduces all the partial reduce outputs to produce the final output [22].

Lin et al. have proposed an overlapping model between map and shuffle phases. The approach is based on two complementary scheduling algorithms called MaxSRPT and SplitSRPT. MaxSRPT minimizes the average response time of the queue, while SplitSRPT addresses the poor performance of MasSRPT when jobs are more unbalanced. Moreover, this study presents an analytical model proving that the problem of minimizing response time in the proposed model is strongly NP-hard [23].

Xie et al. uses a pre-shuffling approach to reduce the network overload imposed by shuffle intensive applications. To accomplish this, a push model using in-memory buffer and a 2-stage pipeline in the pre-shuffling scheme to exchange partial data between map and reduce tasks are implemented [24].

Costa et al implemented a new Hadoop version incorporating Byzantine fault-tolerance to MapReduce. Initially, the approach runs f þ1 map tasks, f being the maximum number of faulty replicas. This was the minimum number of replicas the authors reached by considering the expected low probability of arbitrary faults [25]. The model also achieves better performance by using speculative execution of reduce tasks. Although the resources used practically doubles in this approach, this cost may be acceptable for a large number of applications handling critical data.

After focusing on the above research works, we have evaluated an approach which helps to develop and implement an algorithm for balancing the data blocks to the nodes on heterogeneous Hadoop cluster. We have also evaluated theperformance of heterogeneous Hadoop cluster using the balancing scheme.

3. Proposed Work

3.1Balancing Data in Heterogeneous Hadoop Cluster:

In a heterogeneous cluster, the computing capacities of nodes may significantly vary. A most-efficient node can finish processing data stored in a local disk of the node much faster than its least-efficient nodes. After a fast node completes the processing of its local input data, the fast node must perform load sharing by handling unprocessed data located in one or more slow nodes in cluster. This can be achieved by a data balancing scheme that distributes and stores data across multiple heterogeneous nodes based on their computing capacities. Data placement overheads can be reduced if the number data blocks placed on the disk of each node is proportional to the node‟s data processing capacity.

In our data balancing method, we designed a algorithm and incorporated the algorithms into Hadoop‟s HDFS. The algorithm is to initially place data blocks to heterogeneous nodes in a cluster to balance the data load. When all data blocks of an input file required by computing nodes are available in a node, these data blocks are distributed to the computing nodes.

3.2Data Balancing Algorithm:

INPUT: N, Data, t

```
/* N: Number of Nodes, Data: Files of large Data, t: data intensive job/task */
1. DB ← Divide a large file into a number of even-sized data blocks
2. <Nme, Nle, r>←GetInfo(N, t) /* Get the information of nodes in cluster according to their processing capacity after comparing with least-efficient nodes, most-efficient nodes are expected to store and process more data blocks. An N: node number and the r: rack number of the task t. Nme: Most-efficient node, Nle: Least-efficient node */
3. For each db in DB do
     If a replica of db already exists in the Nme(r)/Nle(r) then continue;
     Else
     (Nme ,Nle, r)←db /* The number of data blocks will be distributed to high-efficient nodes and least-efficient nodes respectively according to their processing speed in rack r. */
      End if
      Done
4. Execute the Task/Job : Grep, WordCount
5. End
```

3.3 Flow Chart of the above Algorithm:

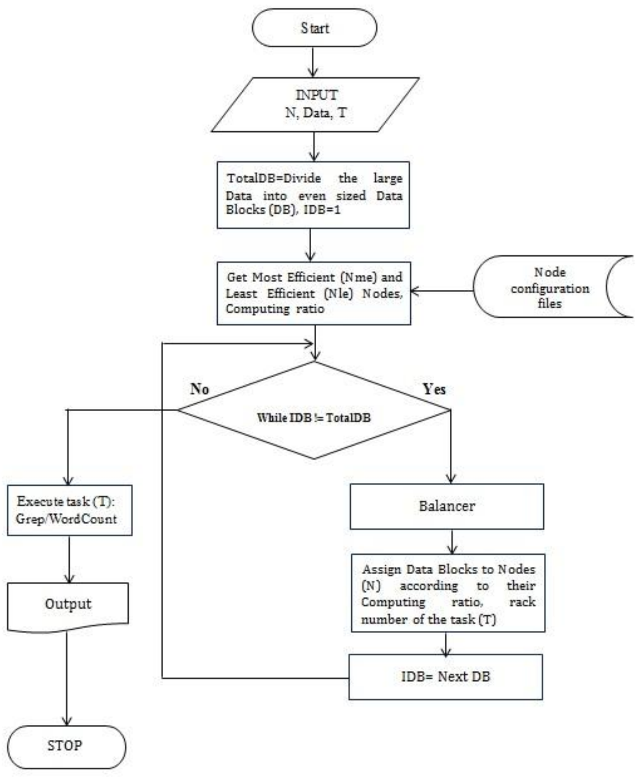
Start
INPUT
N, Data, T
TotalDB=Divide the large Data into even sized Data Blocks (DB), IDB=1
Get Most Efficient (Nme) and Least Efficient (Nle) Nodes, Computing ratio
Node configuration files
No
Yes
While IDB != TotalDB
Execute task (T): Grep/WordCount
Balancer
Assign Data Blocks to Nodes (N) according to their Computing ratio, rack number of the task (T)
Output
IDB= Next DB
STOP

Figure 4: Flow chart

In figure 4, flow chart takes large files (Data Set), number of nodes (N) and task/job (T) as input in the hadoop cluster. Data is divided into even sized blocks (DB) and total number of data blocks are assigned to TotalDB for further use. IDB refers to initial data block which is assigned to one. Efficiency of nodes (N) is calculated according to their computing ratio and response time (Table 2) using node configuration files. Balancer (program) distributes all the data blocks to the most efficient and least efficient nodes as per their performance along with rack number of the task (Grep/WordCount). Finally job (Grep/WordCount) is executed and output is stored in the file on the storage of computer.

4. Performance Evaluation

We have used Ubuntu 14.04 LTS, Java (jdk1.8), Eclipse (Mars) to implement the algorithm.

4.1 Ubuntu 14.04 LTS

It has important features such as stability, speed, open source and reliability which motivated us to use it for running the Hadoop.

4.2 Java (Jdk 1.8)

It is designed to be simple and architecture neutral, so that it could be executed on a variety of hardware. Its main characteristics such as easy learning, object oriented, and platform independent inspired us to write programs and execute them on Hadoop platform.

4.3Eclipse Mars 1

Eclipse provides IDEs and platforms for nearly every language and architecture. It is famous for Java IDE, JavaScript and PHP IDEs built on extensible platforms for creating desktop, web, and cloud IDEs to write and deploy variety of programs easily and speedily.

4.4 Proposed Method

To address the limitation of imbalanced data load in heterogeneous hadoop cluster we propose a data balancing mechanism in the HDFS to initially distribute a large data set to multiple nodes in accordance to the computing capacity of each node.

4.5Hardware and Software used

Technologies used for evaluating the performance of heterogeneous multinodehadoop cluster is described in table 1.

Table 1: Hardware and Software Specification

SN.	Specifications	Master Node (A)	Slave Node1 (B)	Slave Node2 (C)
1	Hard Disk	60GB	50GB	40 GB
2	RAM	4GB	2GB	1 GB
3	Processor	Intel Core i5-4200U	Intel Pentium Dual Core, E-6700	Intel Core 2 Duo E-7500
4	CPU	2.30GHz	3.20GHz	2.93GHz
5	Operating System	Ubuntu 14.04 LTS	Ubuntu 14.04 LTS	Ubuntu 14.04 LTS

4.6 Measurement of Heterogeneity

Before implementing the data balancing algorithm, we need to calculate the heterogeneity of a Hadoop cluster in terms of data processing speed. Such processing speed highly depends on data-intensive applications. Thus, heterogeneity measurements in the cluster may change while executing different MapReduce applications. We have used a technique to measure each node‟s processing speed I a heterogeneous cluster called computing ratios. Computing ratios are determined by a describing technique carried out in the following steps [17].

First, the data processing operations of a given MapReduce application are separately performed in each node. To compare processing capability, we ensure that allthe nodes process the same amount of data. For example, in our experiments the input file size is set to 2GB.

Second, we noted down the response time of each node performing the data processing tasks.

Third, the smallest response time is used as a reference to standardize the response time measurements. The standardized values are called computing ratios which is used by the data balancing algorithm to allocate input data blocks for the given MapReduce application.A small computing ratio of a node in cluster implies that the node has high speed, indicating that the node should process more data blocks than other nodes.

An instance to demonstrate how to calculate computing ratios that leads the data balancing process.Suppose there are three heterogeneous nodes (i.e. Node A, B and C) in Hadoop cluster. After running a Hadoop application on each node, we record that the response times of the application on node A, B and C are 10, 20, and 30 seconds respectively. The response time of the application on node A is the shortest. Therefore, the computing ratio of node A with respect to this particular application is set to 1, which becomes a reference used to determine computing ratios of node B and C. Thus the computing ratios of node B and C are 2 and 3, respectively. Table 2 shows the response times, computing ratios for each node and data blocks to be distributed in a Hadoop cluster. Naturally, the fast computing node (i.e. node A) has to handle 60 data blocks whereas the slow node (i.e. node C) only needs to process 20 data blocks.

Table 2: Response time and computing Ratio

Node	Response Time (Sec)	Computing Ratio	No of Data Blocks	Speed
A	10	1	60	Faster
B	20	2	30	Average
C	30	3	20	Slowest

The data block distribution is governed by a data distribution server, which constructs network and calculates disk space utilization. The server generates and maintains a configuration file containing a list of computing-ratio information. The data distribution server applies the round-robin policy to assign input data blocks to heterogeneous nodes based on their computing ratios.

5. Results

We have used the commodity hardware configuration showed in table 1 to implement and evaluate the data balancing algorithm in a heterogeneous Hadoop cluster. The cluster consists of three heterogeneous nodes (Table 1). The Grepand WordCount are two Hadoop applications running on the cluster. Grep is a searching tool for a regular expression in a text file; whereas WordCount is a program written in java and used to count the number of words consisting set of numbers, letters, special symbols etc. in text file.

Computing Ratios of the Three Nodes with respect to Grep and WordCount Applications and number of data blocks allotted. (File size 2 GB) large data sets are taken from website of National Centers for Environmental Information, National Oceanic and Atmospheric Administration (NOAA) for analysis purpose [26].

Table 3: Computing Ratio

Node	Ratio for Grep (sec)	Ratio for WordCount (sec)	Number of data blocks
A	1	1	9
B	2	2	6
C	3	4	3

The data allocating server follows the approach of measuring heterogeneity to obtain computing ratios of the three nodes with respect to Grep and WordCount applications.

Figure 5 and 6 show the response times of the Grep and WordCount applications running on each node of the Hadoop cluster when the input file size is 2 GB. The results plotted in figures suggest that computing ratios are independent of input file size, because the response times of two applications are proportional to the file size. Figures also depict that the input file of same size given, Grep"s response times are shorter than those of WordCount. Consequently, the computing ratios of Grep are different from those of WordCount.

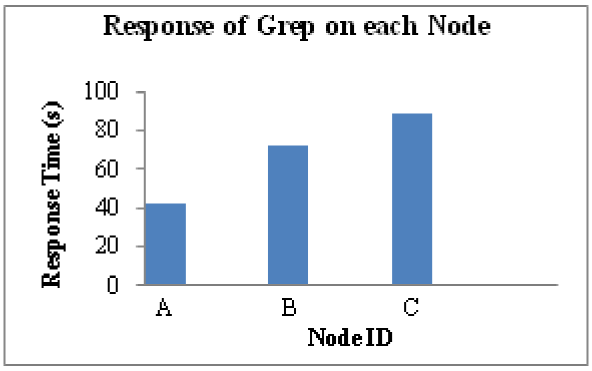

Figure 5: Response time of Grep application on each node

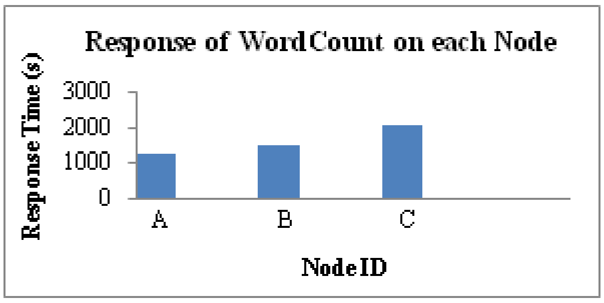

Figure 6: Response time of WordCount application on each node

Now we are positioned to evaluate the impacts of data balancing decisions on the response times of Grep and WordCount. Table 4 shows five demonstrative data balancing decisions, including two optional data balancing decisions (D1-2-3 and D1-2-4) offered by the data balancing algorithm for the Grep and WordCount applications. The data blocks of input file are balanced and distributed on the three nodes based on fie different data balancing decisions, among which two optional decisions (ie. D1.2-3 and D1-2-4) are made by our data balancing scheme as per computing ratios stored in configuration file.

5.1 Data Balancing Schemes

Table 4: Data Balancing Schemes

Notation	Data Balancing Schemes
D1-2-3	Allocating data blocks under the computing ratios of the Grep. This is the optimum data balance for Grep
D1-2-4	Allocating data blocks under the computing ratios of the WordCount. This is the optimum data balance for WordCount
All-in-A	Allocating all the data blocks to node A.
All-in-B	Allocating all the data blocks to node B.
All-in-C	Allocating all the data blocks to node C.

Figure 7 shows the impacts of data balance on the response times of the Grep application. The first (from left) bar in represent the response time of the Grep application after distributing data blocks based on Grep"s computing ratios. Figure also shows the response time of Grep on the 3-node cluster with the other four data balancing decisions. We observe from figure 7 that data balancing decision (denoted as D1-2-3) leads to the best performance of Grep, because the input data blocks are distributed according to the computing ratios of nodes. If all the data blocks are given to node C decision (All-in-C), Grep performs extremely poorly. Grep"s response time is unsatisfactorily and long under the „All-in-C" decision, because all the data blocks are placed to node C. Node C is the slowest node in the cluster.

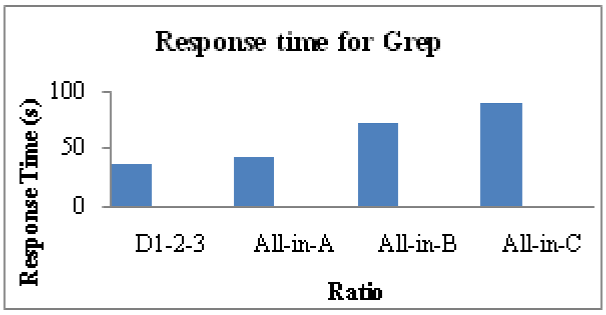

Figure 7: Impact of Data balance on performance of Grep

Figure 8 demonstrates the impacts of data balancing decisions on the response times of WordCount application. The second bar from left in figure depicts the response times of the WordCount application on the cluster under and best data balancing decision. In this optimum data balancing case, the input data blocks are placed according to the computing ratios decided and managed by the data distributed server. The result plotted in figure 8 indicate that the response time of WordCount under the optimal „D1-2-4" data balancing decision is the smallest compared with the other data balancing decisions. The performance of „All-in-C" data balancing is unacceptably poor. The „D1-2-4" data balancing decision is proved to be the best, because this data balancing decision is made on the heterogeneity measurement i.e. computing ratios.

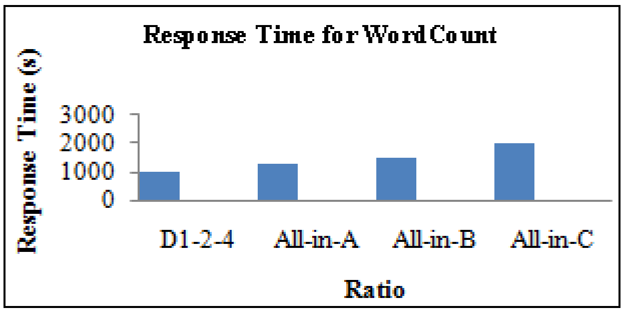

Figure 8: Impact of Data balance on the performance of WordCount

Our results reported in figures 7 and 8 that our data balancing scheme can increase the performance of Grep and WordCount by up to 11.9% and 20.87 % respectively.

6. Conclusion

We mentioned a performance problem in HDFS on heterogeneous cluster. Motivated by the performance decrease caused by heterogeneity, we designed and implemented a data balancing mechanism in HDFS. The new algorithm balances the data blocks of an input file to heterogeneous computers according to their processing capabilities. Our approach significantly increases the performance of Hadoop heterogeneous clusters.

References

[1] Facebook. Under the hood: Scheduling MapReduce jobs more efficiently with Corona: 2012
[2] Bakshi K. Considerations for beg data: architecture and approach. In: Aerospace Conference. IEEE: 2012. P 1-7.
[3] http://www.vcloudnews.com/every-day-big-data-statistics-2-5-quintillion-bytes-of-data-created-daily/, 2015
[4] IvaniltonPolato, Reginaldo Re, Alfredo Goldman, Fabio Kon"A comprehensive view of Hadoop research – A systematic literature review" 2014
[5] Statista, 2016a
[6] http://download.microsoft.com/download/C/2/D/C2D-2D5FA-768A-49AD-8957-1A434C6C8126/ Microsoft_Modern_Data_Warehouse_white_paper.pdf, 2016
[7] https://www.emc.com/collateral/analyst-reports/idc-extrac -ting-value-from-chaos-ar.pdf, 2011
[8] Zikopoulos P, Eaton C. Understanding big data: analytics for enterprise clasHadoop and streaing data. McGraw-Hill; 2011
[9] White T. Hadoop: The definitive guide, 3rd edition. O‟Reilly Media.Inc; 2012
[10] Apache Software Foundation. Hadoop.http://hadoop.apache.org/hadoop.
[11] Yahoo.Yahoo! launches world‟s largest hadoop production application. htttp://tinyurl.com/2hgzv7.
[12] R.Pike, S.Dorward, R.Griesemer, and S.Quinlan. *Interpreting the data: Parallel analysis with Sawzall*, volume 13.IOS Press, 2005.
[13] C.Olston, B.Reed, U.Srivastava, R.Kumar, and A.Tomkins. Pig latin: a not-so-foreign language for data processing. In SIGMOD ‟08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data, pages 1099–1110. ACM, 2008.
[14] Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *OSDI '04*, pages 137–150, 2008.
[15] D. Borthakur. *The Hadoop Distributed File System: Architecture and Design*. The Apache Software Foundation, 2007.
[16] Apache Software Foundation. Hadoop.http://hadoop.apache.org/hadoop.
[17] https://etd.auburn.edu/bitstream/handle/10415/2962/dis-sertation.pdf?sequence=2
[18] ZachariaFadika and MadhusudhanGovindaraju.Lemo-mr: Low overhead and elastic mapreduce implementation optimized for memory and cpu-intensive applications. In *CloudCom*, pages 1–8, 2010.
[19] Leo S, Zanetti G. Pydoop: a Python MapReduce and HDFS API for Hadoop. In proceedings of the 19th international symposium on high performance distributed computing. New York, NY, USA: ACM; 2010. P. 819-25.
[20] Shavachko K, Kuang H, Radia S, Chansler R. The Hadoop distributed file system. In: proceedings of the 26th symposium on mass storage systems and techonologies. Washingtom, DC, USA: IEEE. P. 1-10.
[21] Vernica R, Balmin A, Beyer KS Ercegovac V. Adaptive MapReduce using situation-aware mappers. In: proceedingsof the 15thinternationalconference on extending database technology . New York, NY, USA: ACM; 2012 p. 420-31
[22] Ahmad F., Lee S., Thottethodi M. and Vijaykumar T. (2013). MapReduce with communication overlap (MARCO). J Parallel DistribComput 2013:73(5):608-20.
[23] Lin M, Zhang L, Wierman A, Tan J, Joint optimization of overloading phases in MapReduce. Perform Eval 2013;70(10):720-35 2013
[24] Xie J, Tian Y, Yin S, Zhang J, Ruan X, Qin X, Adativepreshuffling in Hadoop clusters, Procedia, Computer Science 2013: 18(0):2458-67 , 2013
[25] Costa P, Pasin M, Bessani A, Correia M. O the performance of byzantine faulttolerantMapReduce. IEEE Trans Dependable Secure Computing 2013:10(5):301-13
[26] https://www.ncdc.noaa.gov/orders/qclcd