An Empirical Assessment of Hadoop Cluster Performance Enhancement on Replica Management

Sujit Roy, Md. Atikur Rahman, Md. Rasel Mia

Abstract— A Hadoop cluster is a different type of arithmetic cluster designed specifically for storing, managing and analyzing vast amounts of formless data on a circulated computing system. HDFS and Map Reduce are designed for data loading and data processing. The HDFS layer delivers a virtual file system towards client application. The main feature of HDFS is data replication and simulated amongst nodes. In this paper is an Empirical Assessment of Hadoop's dividing attainment that develops the Hadoop's default block placement policy which is made by designing a rack-aware cluster to improve data obtainability. Also, An Adaptive Data Replication Algorithm based on access count prediction in Hadoop using Lagrange's exclamation is a modified sequence of events. To demonstrate investigates on a rack-aware cluster setup which significantly reduced the task completion time and the volume of the data being processed increases the c arithmetic speeds due to update cost. The threshold level for balance between the replication factor and update cost is recognized and accessible realistically.

Index Terms— Big Data, Cloud Computing, Hadoop Cluster, HDFS (Hadoop Distributed File System), Hadoop, MapReduce Framework, Replica Management.

1 Introduction

Cloud computing is an influential technology to maintain expensive computing hardware, dedicated space, to perform massive-scale and complex computing, and software. In clusters of all sizes, throughput and job completion time are important metrics for computation efficiency, which determines the cost of data centers and user satisfaction [12], [13]. Big Data, [2] is termed for a collection of data sets which are large and complex and difficult to process using old-fashioned data processing tools. The need for Big Data management is to ensure high levels of data accessibility for business intelligence, data recognize, data processing and big data analytics. Big Data is stretchy in that it can process a variety of input data sources, structured and unstructured, streaming or not, with very scared input data size limits.

Hadoop [11] is the popular open source implementation of map-reduce, a powerful tool designed for deep analysis and transformation of very large data sets. It is capable to connect and coordinate thousands of nodes inside a cluster. A distributed system is a pool of autonomous compute nodes [1] connected by swift networks that appear as a single server. In reality, solving complex problems involves division of problem into subtasks and each of which is solved by one or more compute nodes which communicate with each other by message passing. The Hadoop runtime system coupled with HDFS provides parallelism and concurrency to achieve system reliability. Applications run in Hadoop as containers, the concurrency of which affects completion time of an application as well as system resource usage concurrent containers, resource bottlenecks occur and when they're too few, system resources are underutilized.

Most important categories of the machine in a Hadoop distribution system are Client machines, Master nodes and Slave nodes. The Master nodes organize storing of data and running parallel computations on all that data using MapReduce. HDFS the Name Node organizes and coordinates the data storage function, the Job Tracker organizes and coordinates the parallel processing of data using MapReduce algorithm and Slave Nodes are the vast majority of machines and do all the cloudy work of storing the data and running the computations. Respectively slave runs a Data Node, a Task Tracker spirit that communicates with and receives information from their master nodes. The Task Tracker spirit is a slave to the Job Tracker likewise the Data Node spirit to the Name Node.

MapReduce [3] is one of many programming models available for processing large data sets in Hadoop which maintains task parallelization, job scheduling, resource allocation and data distribution in the backend. For data analysis has two major components, a mapper and a reducer in the Map-Reduce framework. A mapper maps every key/value record in the dataset by random intermediate key and a reducer generates final key/value pairs by applying computations on the combined pairs. The MapReduce framework deceits in running such simple but powerful functions with Hadoop's automatic parallelization, distribution of large-scale computations and fault tolerance features using commodity hardware. The MapReduce model has been immovable by different areas including distributed graph, graph problems, inverted index and distributed sort.

- *Sujit Roy is working as a faculty member at Dept. of Computer Science and Engineering, Gono Bishwabidyalay, Bangladesh, PH-01746952150. E-mail: roysojib09102029@gmail.com*
- *Md. Atikur Rahman is working as a faculty member at Dept. of Computer Science and Engineering, Gono Bishwabidyalay, Bangladesh, PH-01912288599. E-mail: atikhasan.cse@gmail.com*
- *Md. Rasel Mia is working as a faculty member at Dept. of Computer Science and Engineering, Gono Bishwabidyalay, Bangladesh, PH-01738189846, E-mail: rasel376mahmud@gmail.com*

Hadoop Distributed File System (HDFS) [3] is a Java-based file system that delivers a scalable and reliable data storage system and it is built on top of the local file system and is able to support few petabytes of the big dataset to be distributed over clusters of service servers. HDFS [4] file system is designed for storing huge files with streaming data access patterns, running on clusters of commodity hardware. The HDFS is so large that replicas of files are constantly created to meet Performing and availability requirements.

A replica [5] is usually created so as the new storage location offers better Performing and availability for accesses to or from an individual location. The Hadoop architecture the replica is commonly selected based on storage and network feasibility which makes it fault tolerant soon as to recover from failing Data Node. Additional replicas are stored randomly on any rack which could be configured and dominated using scripts and the store each file as a sequence of blocks. The block size and replication factor are configurable per file. The replication factor can be specified at file creation time and can be changed far along. The Name Node makes all decisions regarding replication of blocks, it sporadically receives a Block report from each of the Data Nodes in the cluster. A Block report contains a list of all blocks on a Data Node.

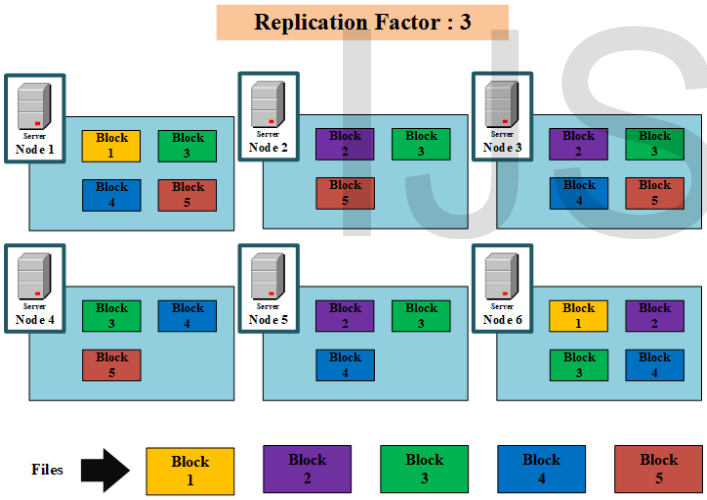

Figure 1 – Block Replication of Hadoop Architecture

The rest of this paper is organized as follows. Section II discusses related work on data replication organizations in the cluster; Section III describes the proposed work of a Hadoop cluster and the data locality problem; Section IV estimates the Performing of the system by conducting experiments on changeable data replication levels. Finally, Section V concludes and the future scope.

2 Related Work

HDFS is considered to reliably store very big files across machines in a large cluster. The tenacity of data replication in HDFS is primarily to progress the obtainability of data. Replication of a data file serves the purpose of system reliability where if one or more nodes fail in a cluster. To improve fault tolerance of data in the presence of failure and few of those are deliberated below.

Sangwon Seo, Ingook Jang; [7] suggested optimization patterns such as pre-shuffling and prefetching to resolve shared environmental harms. Both of the above schemes were implemented in a High Performing MapReduce Engine (HPMR). In an Intra block fetching an input split or an intermediate output is prefaced whereas the whole candidate data block is perfected in the inter-block prefetching. The pre-shuffling scheme reduces the amount of intermediate output to shuffle and at the time of pre-shuffling HPMR looks over an input split before the map phase begins and predicts the target reducer where the key-value pairs are separated. Prefetching and Pre-shuffling organizations to improve MapReduce Performing when physical nodes are shared by multiple users. HPMR diminishes network overhead and exploits data locality companionable with both dedicated and shared surroundings.

Abad. C. L, Yi Lu, Campbell. R.H; [6] proposed a data replication and placement procedure (DARE) that adjusts to the fluctuations in workload. Adaptive Data Replication for Efficient resolves two harms, how many replicas to allocate for each file and where to place them, using probabilistic sampling and a competitive aging algorithm individually at each node. Adaptive Data Replication for Efficient benefits from existing remote data rescue and selects a subset of the data and creates a replica without consuming additional network and reckoning resources. The authors designed a probabilistic dynamic replication algorithm with the ensuing features: Nodes sample assigned tasks and replicate prevalent files in a distributed manner. Correlated data accesses are distributed over diverse nodes as old replicas deleted and new replicas are created. Investigates demonstrated seven times improvement in data locality and 70% improvement in cluster arranging. Reduces job turnaround time by 16% in dedicated clusters and 19% in virtualized public clouds.

Khanli. L. M, Isazadeh. A; [8] proposed a procedure to decrease access latency by expecting the future usage of files. Predictive Hierarchal Fast Spread (PHFS) pre-replicates data in a hierarchal data grid using two stages: collecting data access statistics and applying data mining techniques like clustering and association instruction mining all over the system. Files are assigned value α which is between 0 and 1 for representing relationships between files. Files are arranged according to the value of α which is called the predictive working set. PHFS utilizes the predictive working set of a file and replicates all members of a predictive working set including the file and all files on the path from source to the client. PHFS attempts to improve data locality by expecting the user's future stresses and pre-replicating them in advance thereby achieving higher accessibility with the enhanced usage of resources.

Jungha Lee, Jong Beom Lim; [9] proposed a data replication scheme (ADRAP) that is adaptive to overhead, associated with the data locality problematic. The algorithm works created on access count prediction to reduce the data transfer time and develops data locality thus reducing total processing time. The arrangement adaptively determines the required replication

factor by evaluating data access patterns and recent replication factor for a particular data file. The paper gives the following: Optimizes the replication factor and effectively circumvents overhead caused by data replication. Enthusiastically concludes data replication requirements and Minimizes processing time of MapReduce trades by humanizing the data locality.

Zaharia.M, Borthakur. D; [10] suggested a delay scheduling technique that demonstrates the conflict between fairness in scheduling and data locality by scheming a fair scheduler for a 600-node Hadoop cluster at Facebook. The procedure is applicable under a wide variety of scheduling policies beyond fair sharing such as the Hadoop Fair Scheduler. HFS has two main objectives: Fair sharing and Data locality. To accomplish the area the scheduler reallocates resources between jobs when the number of jobs changes by killing running tasks to make room for the new job and waiting for running tasks to appearance.

Delay scheduling accomplishes well in typical Hadoop workloads and is applicable beyond fair sharing. Delay scheduling in HFS is generalized to implement a hierarchical scheduling policy motivated by the needs of Facebook's users. The schedule divides slots between users based on weighted fair sharing at top-level and sanctions users to schedule their own employments using either FIFO or fair sharing.

3 Proposed Work

The perseverance of this investigation is to evaluate the Performing of the Hadoop cluster and to the organization a rack-aware Hadoop cluster. To accomplish the purpose a data replication pattern is reformed to fit the system, implemented using a rack-aware Hadoop cluster. In such a cluster task are run manually with varying levels of data replication. This research work makes a small involvement in Minimizing processing time and data transfer load between racks by improving data locality.

3.1. Rack Aware Hadoop Clusters

Hadoop apparatuses are rack-aware. In HDFS block placement will use rack awareness for fault tolerance by placing one block replica on a different rack. This provides data availability in the event of a network switch failure or partition within the cluster. The Data obtainability and locality are interrelated dominions in the realm of distributed processing which when not handle appropriately leads to Performed issues paper contracts with a similar type situation where a cluster of nodes are involved and each node belonging to the same or different rack. For the purpose of Experimental Assessment, this paper utilizes a Hadoop cluster setup with one Master node and seven slave nodes each configured automatically to define the rack number it belongs. In this paper an enhanced data placement policy to avoid data loss and improve network Depiction.

Data blocks are replicated to multiple machines to preclude data loss due to machine fiascoes. A typical block size used by HDFS is 128 MB. Thus, an HDFS file is chopped up into 128 MB chunks, and if possible, each chunk will reside on a different Data Node. A postulation that two machines in the same rack have more bandwidth and lower latency between each other than two machines in two different racks is painstaking. It is also assumed cross-rack latency is higher than in-rack latency most of the time. The system is designed for Taxation is shown in figure 2.

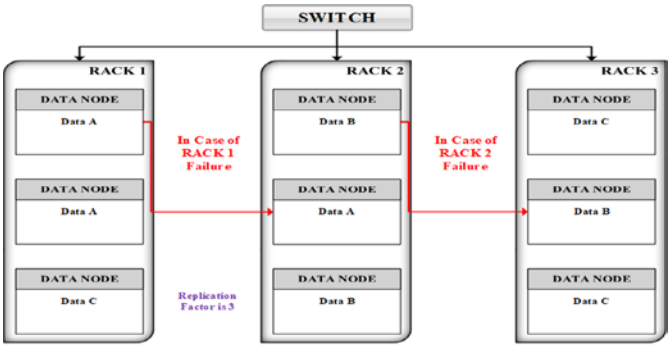

Figure 2 : Rack aware Hadoop Cluster Setup

3.2. Access Count Prediction

Data is replicated thoughtfully for if the replication factor is higher than access count for the specific file then the probability of being processed with node locality is higher than that of the contrasting case. Maintaining different replication factors per data file and haughty that a higher replication factor for a file with higher access count does not always guarantee better data locality. The current replication factor to determine the optimal replication factor.

In addition, the number of rack-off locality nodes is effectively reduced by the replica placement policy. To predict access count for individual files this work utilizes Lagrange's interpolation using a polynomial expression. The calculated method is given below:

In the equation,

Let N is the number of points, x_i are the i^{th} point and fi be the function of x_i.

To calculate the predicted access count, substitute x by time t.

$$G(x) = \frac{(x-x_1)(x-x_2)(x-x_3)\dots(x-x_n)}{(x_0-x_1)(x_0-x_2)(x_0-x_3)\dots(x_1-x_n)} f_0 + \frac{(x-x_0)(x-x_2)(x-x_3)\dots(x-x_n)}{(x_1-x_0)(x_1-x_2)(x_1-x_3)\dots(x_1-x_n)} f_1 \dots + \frac{(x-x_0)(x-x_1)(x-x_3)\dots(x-x_n)}{(x_n-x_1)(x_n-x_2)(x_n-x_3)\dots(x_n-x_{n-1})} f_n) \dots\dots\dots(1)$$

Where access time t and y by an access count at t.

3.3. Data Placement Policy

In-rack nodes have much more required network traffic than off-rack nodes. The cluster proprietor uses the configuration variable net.topology.script.file.name to choose on which rack the nodes belong to and this script is configured so that each node runs the script to determine its rack id. In a default fixing nodes are assumed to belong to the same rack with similar rack id.

In the case of a small cluster, servers are connected by a single switch with two levels of locality on-machine and off-machine. But for larger installations, it must be kept in mind that data replicas occur on multiple machines and distances multiple racks. A rack aware Hadoop file system is created with the use of scripts which allows the master node to map the network

topology of the cluster. The script is in an executable form which allows it to return the track address of each node. The script returns on a list of rack names, one for each input which is provided as arguments such as IP addresses of nodes in the cluster ordered consistently. Mapping scripts specify the key topology.script.file.name in conf/hadoop-site.xml.

3.3.1. Configuring Rack Awareness

To configure a Hadoop cluster into a rack-aware system data must be separated into multiple file blocks and store them on different machines among the cluster, failing to organize a Hadoop into a rack-aware system may result in loss of data, but rack failure is not recurrent and this can be avoided by utilizing Hadoop configuration files. Hadoop is configured using the topology things and input file to the script for rack identification. The following script performs rack identification based on IP addresses given a hierarchical IP addressing scheme enforced by the network administrator.

Configuring rack awareness in Hadoop involves two steps:

• Configure the "topology.script.file.name" in core-site.xml

topology.script.file.name	**Rack-awareness.sh**

<property>
<name>topology.node.switch.mapping.impl</name> <value>org.apache.hadoop.net.ScriptBasedMapping </value>
</property>
<property>
<name>topology.script.file.name</name>
<value>core/rack-awareness.sh</value>
</property>
	HADOOP_CONF=/usr/local/hadoop/conf
	while [$# -gt 0] ; do
	nodeArg=$1
	ex-
	ec<${HADOOP_CONF}/topology.
	data result=""while read line ; do
	ar=($line)
	if ["${ar[0]}" = "$nodeArg"] ;
	then result="${ar[1]}"
	fi
	done
	shift
	if [-z "$result"] ; then
	echo -n "/default/rack "
	else
	echo -n "$result "
	fi
	done

Topology.data (master)Machine1.pc (slave)Machine2.pc (slave)Machine3.pc (slave)Machine4.pc	(slave)Machine5.pc /dc3/rack3 (slave)Machine6.pc /dc3/rack3 (slave)Machine7.pc /dc4/rack4 (slave)Machine8.pc /dc4/rack4

4 Experiential Assessment

Experiments were directed on the rack-aware Hadoop cluster to evaluate its Representation in terms of data availability. It involves two situations: one involving too many data files and other with no data files but complex computations. The former one is the WordCount application and the latter is Pi value calculation, also both of the above experiments were conducted in the Hadoop Framework. Machine Configuration: 8 Nodes (Similar).Nodes within a rack are connected by one Ethernet Switch and one Fast Ethernet switch is used between racks. The size of data block is set to 64Mb with an increasing replication factor starting from 1. Specifically, 8 jobs are run ranging from 1 to 8 replication levels which are not greater than a number of nodes available in the cluster. The result of the map phase only is experimented i.e. the completion time and data locality of the map phase is averaged over 8 runs. As noticed, in terms of throughput, the tasks with node locality is better than tasks with rack-off locality.

4.1. Results and Graph

Representation Assessments show that as replication levels increase the task completion time gets significantly concentrated for computation involving no data files. But for computations that encompass data files the completion time reduces and then again shoots up due to update cost. Mutually experimentations were conducted on replication levels ranging from one to eight which is not higher than a number of nodes in the cluster.

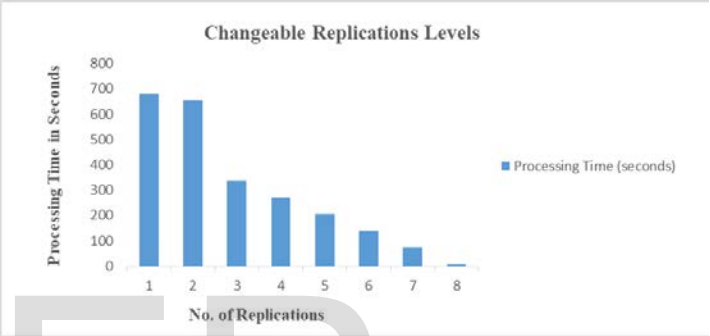

Figure 3. Replication Levels for PI Value

Figure 3 demonstrations that the data replication scheme used in PI value calculation reduces the task completion time. By comparing, with increasing replication factors there is some increase in the Performing and when the replication level is increased by 3, its completion time is 337 s, and advance reduces significantly to 8.12 s at replication level 8. This demonstration that as replication factor increases multiple map stages is announced and thus the computation speeds up.

4.1.2. Experiment for Word Count

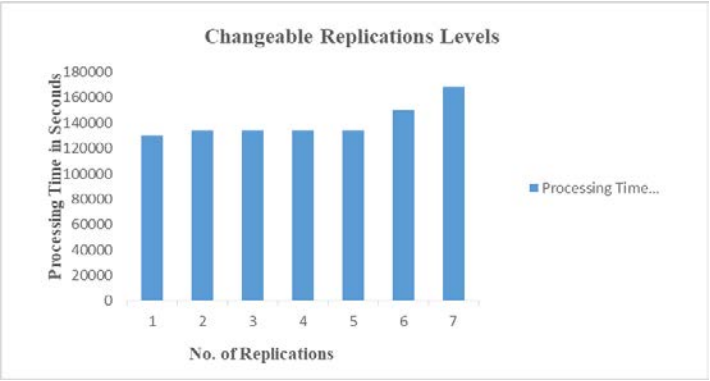

Figure 4. Replication Levels for Word Count

Figure 4 demonstrations the task completion time for Word Count application where the completion time reduces linearly

with replication factor increase but once it reaches the threshold level the Enactment starts to deteriorate. This shows that the computations involving data files do not linearly improve in Enactment as replication increases. By default, the replication level in HDFS is set to 3 which will reduce the Performing speed and thus the completion time is 132220 seconds. On increased replication levels the computation speed boosts up but once it reaches the threshold the time comes down from 1300430 s to 1608600 s.

Representation Assessments demonstration that as replication levels increase the task completion time gets significantly reduced for computation involving no data files. But for computations that involve data files, the completion time reduces and then again shoots up due to update cost. Both experiments were conducted on replication levels ranging from one to eight which is not higher than a number of nodes in the cluster.

5 Conclusion and Future Scope

Data Replication in Hadoop framework to investigate the data locality problem was experimented and proved that replication improves Performing and also decreases it as the threshold limit is crossed. Supplementary the replication factor was supported by an access count prediction algorithm for data files using Lagrange's interpolation which optimizes the replication factor per data file. Enactment Assessment indicated that our data replication organization reduces the task completion time. The task completion time for Pi value calculation started with 680s and came down to 8.12s and similarly, the Word Count application does start with a completion time of 134300 s and came down to 130430 s. But once the threshold limit is reached the completion time shoots up to 168600 s.

6 Reference

[1] Dean, J., Chang, F., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M...& Gruber, R. E. (2008). Bigtable: A distributed storage system for structured data. ACM Transactions on Computer Systems (TOCS), 26(2), 4.

[2] Agrawal, Divyakant, Sudipto Das, and Amr El Abbadi. "Big data and cloud computing: current state and future opportunities." In Proceedings of the 14th International Conference on Extending Database Technology, pp. 530-533. ACM, 2011.

[3] J. Dean and S. Ghemawat, MapReduce: a flexible data processing tool, Commun. ACM, Vol. 53 (1), pp. 7277, 2010.

[4] Hadoop DFS User Guide. http://hadoop.apache.org/.

[5] Fadika, Zacharia, Madhusudhan Govindaraju, Richard Canon, and Lavanya Ramakrishnan. "Evaluating Hadoop for data-intensive scientific operations." InCloud Computing (CLOUD), 2012 IEEE 5th International Conference on, pp. 67-74. IEEE, 2012.

[6] K. Elissa, Abad, Cristina L., Yi Lu, and Roy H. Campbell. "DARE: Adaptive data replication for efficient cluster scheduling." In Cluster Computing (Cluster), 2011 IEEE International Conference on, pp. 159-168. IEEE, 2011. "Title of paper if known," unpublished.

[7] Seo, Sangwon, Ingook Jang, Kyung Chung Woo, Inkyo Kim, Jin-Soo Kim, and Seungryoul Maeng. "HPMR: Prefetching and pre-shuffling in shared MapReduce computation environment." In Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on, pp. 1-8. IEEE, 2009.

[8] Khanli, Leyli Mohammad, Ayaz Isazadeh, and Tahmuras N. Shishavan. "PHFS: A dynamic replication method, to decrease access latency in the multi-tier data grid." Future Generation Computer Systems 27, no. 3 (2011): 233-244.

[9] Lee, Jungha, JongBeom Lim, HeonchangYu Daeyong Jung, KwangSik Chung, and JoonMin Gil. "Adaptive Data Replication Scheme Based on Access Count Prediction in Hadoop." The World Congress in Computer Science, Computer Engineering, and Applied Computing, 2013.

[10] Zaharia, Matei, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. "Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling." In Proceedings of the 5th European conference on Computer systems, pp. 265-278. ACM, 2010.

[11] J. Liu, F. Liu, and N. Ansari, "Monitoring and analyzing big traffic data of a large-scale cellular network with Hadoop," Network, vol. 28, issue 4, pp. 32-39, 2014.

[12] G. Ananthanarayanan, S. Agarwal, S. Kandula, A. Greenberg, I. Stoica, D. Harlan, E. Harris, "Scarlett: Coping with skewed popularity content in MapReduce clusters", Proc. Eur. Conf Comput. Syst. (EurcSys), 2011.

[13] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, I. Stoica, "Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling", Proc. Eur. Conf. Comput. Syst. (EuroSys), 2010.