ISSN 2395-1621

Reducing Device Power Consumption using Hadoop and Analyzing Applications Qos

[#1]Snehal Patil, [#2]Aishwarya Pedapalliwar, [#3]Surbhi Nanoti, [#4]Abhilasha Thorat

[1]patilsnehal271@gmail.com
[2]aishwaryapedapalliwar@gmail.com
[3]surbhi.nanoti2@gmail.com
[4]thoratabhilasha67@gmail.com

SKN College of Engineering, Pune.

ABSTRACT

In the past decade the need for energy efficient approach to deal with the large amount of data being generated is daunting on the rise. With the development of new technologies, various devices and data being generated on the social media sites the computer industry has been challenged to exploit the existing techniques and explore new methods and techniques to process huge data sets to generate optimized response time. At the same time the need to address the quality data is also essential. With the ever increasing data the techniques to handle such data should also be updated to be able to handle big data and their challenged. One of the many recent processing models to process large datasets in an energy efficient manner is MapReduce technique. Distribution and parallelization of data on multiple servers is provided by Mapreduce. The paper will outline one of the best techniques to process huge datasets is MapReduce through cloud computing environment provided by Amazon Web Services. MapReduce also can help developers to carry out parallel and distributed computation in a cloud environment.

KEYWORDS: MapReduce, JobTracker, TaskTracker, HDFS, DataNode, NameNode.

ARTICLE INFO

Article History

Received: 16th March 2017

Received in revised form :

16th March 2017

Accepted: 18th March 2017

Published online :

24th March 2017

I. INTRODUCTION

Big data is a term that describes extremely large data sets that can be analyzed computationally to reveal patterns, trends and associations especially relating to human behavior and interactions. Big data is being generated by every digital process and social media exchange. The term itself describes the large volume of data both structured and unstructured. The data sets are so complex that traditional applications are inadequate to deal with them. Big data has great potential to bring huge benefits to business organizations[1]. Due to its large volume there are various challenges faced such as storage, search, managing data, quality of data and data security. Big data is characterized using 3V's model that includes volume, velocity, and variety[2]. The word 'Big' itself defines the volume in big data. The social media sites generates huge amount of data (terabytes) everyday which is complex to process using traditional systems. Velocity in big data deals with the speed of the data coming from different sources. This characteristic is not being limited to the speed of incoming data but also the speed at which the data flows[18]. For example, the data coming from sensor devices of huge amount would be constantly moving to the database store. Thus traditional systems are incapable of performing analytics on data which is constantly in motion. The data being produced is not of particular single category as it includes traditional data as well as semi structured data from various resources such as Web pages, log files, social media sites, emails etc. All the data generated is completely different including raw, structured, unstructured, semi structured which is difficult to be handled using existing traditional systems[1][2].

In today's time, huge amount of data is generated from multiple resources, that needs to be processed which remains a serious challenge. Various organizations encounter difficulties to process such large data. They are inadequate to manage, manipulate, process, share and retrieve large data using inefficient traditional software tools. To process such big data, developers require large volume of storage and multiple processing nodes to deal with

complex applications and ineffectual processing methods like sequential and centralized data processing.

One of the efficient approaches for massive data processing is to perform parallel and distributed computing in a cloud computing environment. Parallel server computing provides improved data reliability by combining Hadoop Open Source big data parallel distributed processing platform with its distributed file system. It eliminates the need for transferring data by sharing files between servers which contributes for improvement in processing performance [3][9].

Cloud computing offers infrastructure for big data and it is on-demand network access. The major reason for using cloud computing platform are hardware cost reduction and processor cost reduction The main advantage of data processing with cloud computing is capability to easily do parallel and distributed computing on large clusters [7]. HDFS is designed for rapid and reliable computations on cluster of nodes by storing multiple of data blocks , that runs in distributed environment. Hadoop uses HDFS to store data set and uses Map Reduce' s power for distributing and parallelizing this data set[2][19].

II. LITERATURE SURVEY

Before we go any further, it is necessary to define certain definitions that are related to BigData and Hadoop.

Big data is a major information, it is a gathering of vast data sets that can't be handled using conventional processing procedures[1]. It is not only relational database means Structured database but also non-relational database such as Semi-structured or Unstructured. But large amount of data cannot use in traditional process[1][13].

Ms. Min Chen describe 4V's of big data they are as fallow: Volume, Velocity, Veracity and Variety[3][1]. Hadoop is a Framework that takes into consideration the distributed processing of substantial data sets over clusters of PCs . It is intended to scale up from single servers to many machines, each offering nearby calculation and capacity[4][1].

There are three main things are in hadoop development Client machine, Masters, Slaves. The Master nodes supervise the two key practical pieces that make up Hadoop: storing large amount of data (HDFS), and running parallel calculations on all that data (Map Reduce). The Name Node manages and organizes data storage capacity (HDFS), while the Job Tracker administers and arranges the parallel processing of information utilizing Map Reduce. slave means both a Data Node and Task Tracker which is use to communicate with and accept the command from their master nodes. The Task Tracker work under the Data node and job tracker works under the Name Node[7]. "Write once and read-many" is an approach utilized as a part of HDFS and after that permits it to be read many times over regarding the quantities of allocated jobs. As the writing procedure, Hadoop partitions the data into blocks with a predefined block estimate. The blocks are then composed and copied in the HDFS. The blocks can be copied various times in view of a particular esteem which is set to three times by default[15].

The MapReduce function is distributed file system. Basically a large file is distributed into block of identical size and they are split across the cluster for storage[19][21]. In MapReduce implementation there are three stage : Map, Shuffle, and Reduce .

The map stage concern as map function to all input. it is utilized to handle the blocks in the input file that are keep into the PCs nearby capacity. calcuations are done where the data is really put away. Since there is no any conditions in various mappers, all mappers do their work in parallel and they can work in parallel and separately to each other. In cluster if one computer fails then result can be recomputed on another computer[18][8]. A mapper procedures the substance of a piece line by line, translating every line as a key-value match. The real map function is called separately for each of these sets and makes a self-assertively expansive file of new key-value sets from it: A mapper procedures the substance of a piece line by line, translating every line as a key-value match[14]. The real map function is called separately for each of these sets and makes a self-assertively expansive file of new key-value sets from it:

Map (key, value) -> List(key', value')

After Map function finish its process it will pass the result to Shuffle function to arrange the resulting pair with their keys then pass it to Reducer as per their keys. The structure ensures all sets with a similar key are appointed to a similar reducer[17].

Now all pair of key is gather by reducer gather and creates a sorted list from the values. Input for the reduce function is key and the sorted list of values .To make a size of list very small, reduce function compact the list of values . it returns a single value as its output. Reduce function creates an list of key-value pairs, just like the map function[21]:

Reduce (key, List(values)) -> List(key', value')

III. HADOOP FRAMEWORK

Hadoop comes with a distributed file system which is Hadoop distributed file system(HDFS). It is designed for storing large files. Block size of HDFS is much larger than normal file system. HDFS cluster has two types of nodes i.e name node(master node) and data node(slave nodes). The name node manages the file system space and metadata for all the files and directories. The data node stores and retrieves blocks as per the instructions of namenode. User submits a job to the master node. Master node divides the input dataset into many pieces of data into HDFS and forms the several copies of datasets in clusters. HDFS makes multiple copies of datasets and put them over working machines in the form of clusters. Then the MapReduce function starts processing over the blocks of data. The master node partitions a job into map and reduce tasks. These tasks are performed by mapper and reducer functions. The master node then chooses the idle mapper and reducer function and allocates mapper or reducer tasks to the particular data sets. The mapper function receives a map task, read the content of the available dataset and performs map function. Then the map function produces the

intermediate key and value pairs for the given input datasets. The master node receives the physical memory addresses of the buffered data and the reducer function is performed. When the reducer reads all the intermediate key and value pairs, it sorts them by the intermediate keys so that all the data of the same intermediate key are classified together. The reducer sends each unique key and its consequent set of intermediate values to reduce function. The final output is available in the reducer and then it is stored to the distributed file system (HDFS)[4].

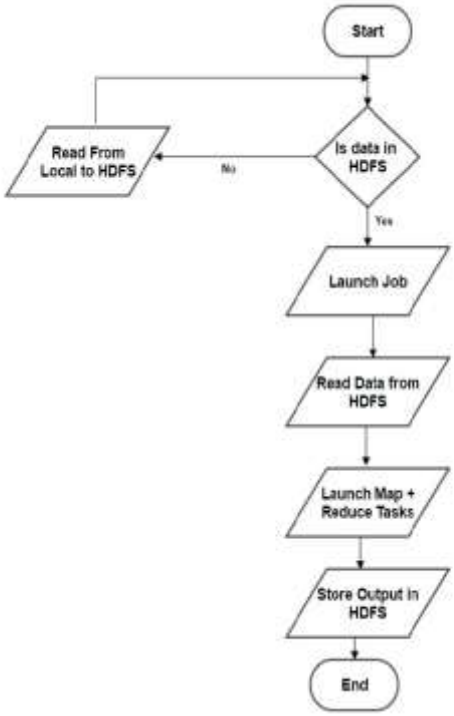

Fig 1. Algorithm for HDFS

DEMERITS

1. Redundancy observed in performance.
2. Meeting the need for high speed processing.
3. Addressing the quality of data.

IV. PROPOSED SYSTEM

To overcome flaws in existing system there is another approach using MapReduce framework in which, we use HDFS as storage for distributed parallel processing.

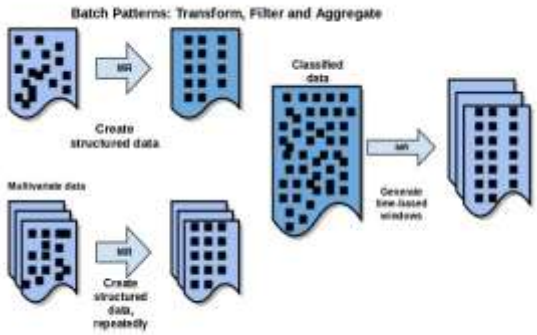

Fig. 2 System Architecture

In the Hadoop Distributed File System the data is written once on the server and then subsequently read and reused many times thereafter. Compared to other file systems repeated read/write operations are well performed by HDFS than the other file systems. The working of HDFS has a main name node and multiple data nodes on commodity hardware cluster. Data is then broken down into separate blocks that are distributed among the various data nodes for storage. These blocks may also be replicated across the nodes to reduce the likelihood of failure. The NameNode is the node in the cluster that knows exactly which data node contains which blocks and the location of data nodes in the cluster. The NameNode manages the access to files, including read,write,create and delete operations. The data nodes constantly communicates with the name node to see if they need to complete a certain task. If any data node is not functioning properly name node is able to re-assign its task to the other data node. Hadoop MapReduce is developed and maintained to break down the data into smaller pieces, process the data in parallel on the distributed cluster and combine it into the desired result[5].

1. Splitting the Data:

The way HDFS has been set up it breaks down very large files into large blocks and stores the copies of these blocks on different nodes in the cluster. When the MapReduce job is started the master node or the name node look after the lifecycle of the job and determines which blocks are needed for processing. If we have n nodes, the HDFS will distribute the file over all these n nodes. Once the data in each block is recognized they are assigned specific key value pairs. This key value pair is given as an input to the mapper class[6].

2. Mapping Phase:

Once the data is in the form acceptable to map, each key-value pair of data is processed by mapping function. The map tasks produce a sequence of key-value pairs from the input and this is done according to the code written for map function[6]. These value generated are collected by master controller and are sorted by key and divided among reduce tasks. The sorting basically assures that the same key values ends with the same reduce tasks.The mapping function is done by segregating the similar keyvalue pairs[8].

3. Reducing Phase:

The Reduce tasks combine all the values associated with a key working with one key at a time. Again the combination process depends on the code written for reduce job. The Master controller process and some number of worker processes at different compute nodes are forked by the user. Worker handles map tasks and reduce tasks but not both[8][6].

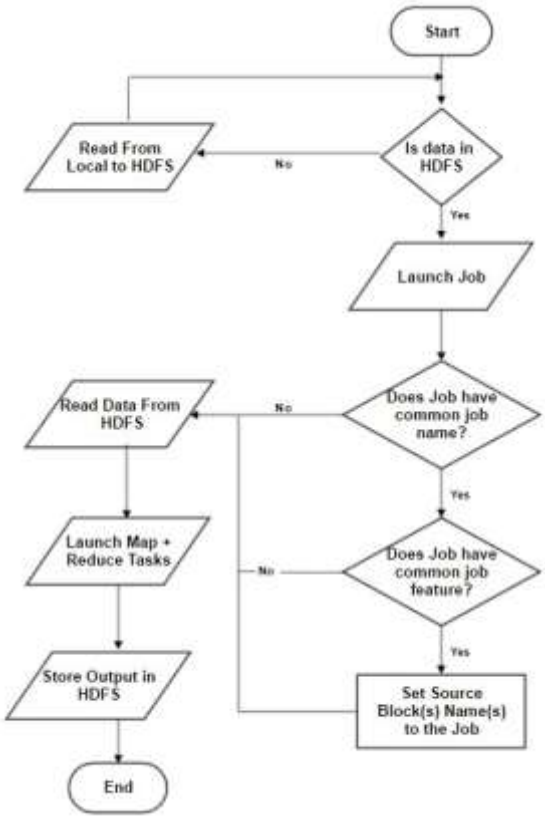

Fig 3.Proposed System Algorithm

V. PROPOSED ALGORITHM

1. Algorithm 1

MapReduce workflow in native Hadoop has been explained as follows:

Step 1: Client " A" sends a request to NameNode. The request includes the need to copy the data files to DataNodes.

Step 2: NameNode replays with the IP address of DataNodes. In the above diagram NameNode replies with the IP address of five nodes (DN1 to DN5).

Step 3: Client " A" accesses the raw data for manipulation in Hadoop.

Step 4 : Client "A" formats the raw data into HDFS format and divides blocks based on the data size. In the above example the blocks B1to B4 are distributed among the DataNodes.

Step 5: Client "A" sends the three copies of each data block to different DataNodes.

Step 6: In this step, client "A" sends a MapReduce job (job1) to the JobTracker daemon with the source data file name (s).

Step 7: JobTracker sends the tasks to all Task Trackers Holding the blocks of the data.

Step 8: Each TaskTracker executes a specific task on each block and sends the results back to the Job Tracker.

Step 9: Job Tracker sends the final result to Client "A". If client "A" has another job that requires the same datasets it repeats the set 6-8.

Step10: In native Hadoop client "B" with a new MapReduce job (job2) will go through step 1-5 even if the datasets are already available in HDFS. However, if client "B" knows that the data exists in HDFS, it will send job2 directly to JobTracker.

Step 11: JobTracker sends job2 to all Task Trackers.

Step12: TaskTrackers execute the tasks and send the results back to the JobTracker.

Step 13: JobTracker sends the final result to Client "B". We can see that there is independency between jobs because there are no conditions that test the relationship between jobs in Native Hadoop. So, every job deals with the same data every time it gets processed. In addition, if we have the same job executed more than one time; it reads all the data every time, which can cause weakness in Hadoop performance.

2. Algorithm 2

Step 1 to Step 8: remain in the same workflow as native Hadoop. Except results from the first 7 steps are stored in the CJBT.

Step 9: JobTracker sends the result to Client "A". In this step, NameNode keeps the names of the blocks that produced the results in the local lookup table (CJBT) by the Common Job Name (Job1) that has common feature as explained above.

Step 10: Client "B" sends a new MapReduce job "Job2" to the JobTracker with the same common job name and same common feature or super-sequence of "Job1".

Step 11: JobTracker sends "job2" to Task Trackers who hold the blocks, which have the first result of the MapReduce "Job1" (DN2, DN4, DN5). In this step,the JobTracker starts with checking the CJBT first to find if it is a new job which has the same common name and common features of any previous ones or not – In this case yes. Then the JobTracker sends "Job2" only to TT2, TT4 and TT5. We may assume here that the lookup table will be updated with more details OR just remain as is because every time we have a new job that may carry the same name of "Job1".

Step 12: TaskTrackers execute the tasks and send the results back to the JobTracker. Step 13: JobTracker sends the final result to Client "B".

This algorithms flow is same as first algorithm but there are few changes as like, first, we test the data is it in HDFS format or no. After launching a job, we tested either this job uses or new architecture by having a common job name from a list that we have in the user API or not. If not, it goes to be processed via a traditional way. Otherwise, we tested either it has a common feature or not. If it doesn't have a common feature, which means it has new features that are not related to any previous job, it goes to be processed via a traditional way. Otherwise it means it has a common job name and a common feature, NameNode checks the CJBT and retrieves block that

contain the common feature and assigns the job to read data from these blocks.

A. Common Job Block Table(CJBT)

CJBT stores information about the jobs and the blocks associated with specific data and features. This enables the related jobs to get the results from specific blocks without checking the entire cluster. Each CJBT is related to only one HDFS data file, which means that there is only one table for each data source file(s) in HDFS.

Common feature that is a sequence or subsequence is identified and updated in CJBT. Common features in CJBT can be compared and updated each time clients submit a new job to Hadoop. Consequently, the size of this table should be controlled and limited to a specific size to keep the architecture reliable and efficient.

VI. CONCLUSION

This paper describes the proposed approach and a new method which will be able to process the datasets efficiently with the help of cloud computing environment. At the same time the quality of service of an application in terms of computation speed, memory utilization and performance ratio, with comparison to the existing approach will show significant difference to achieve the appropriate result.

REFERENCES

[1] Javier Conejero , Omer Rana, Peter Burnap, Jeffrey Morgan, Blanca Caminero, Carmen Carrión , “Analyzing Hadoop power consumption and impact on application QoS”, Elsevier, 2016.

[2] Samira Daneshyar and Majid Razmjoo, “LARGE-SCALE DATA PROCESSING USING MAPREDUCE IN CLOUD COMPUTING ENVIRONMENT”, International Journal on Web Service Computing (IJWSC), Vol.3, No.4, December 2012.

[3]M. Chen, S. Mao, and Y. Liu, "Big Data: A Survey," Mobile Networks and Applications, vol. 19, pp. 171-209, 2014.

[4] R. Gu, X. Yang, J. Yan, Y. Sun, B. Wang, C. Yuan, et al., "SHadoop: Improving MapReduce performance by optimizing job execution mechanism in Hadoop clusters," Journal of Parallel and Distributed Computing, vol. 74, pp. 2166-2179, 2014.

[5] N. Tiwari, S. Sarkar, U. Bellur, and M. Indrawan, "Classification Framework of MapReduce Scheduling Algorithms," ACM Comput. Surev.vol47,pp1-38.

[6] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Job scheduling for multi-user mapreduce clusters," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS2009-55, pp. 1-16, 2009.

[7] C. He, Y. Lu, and D. Swanson, "Matchmaking: A new mapreduce scheduling technique," in IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom), pp. 40-47, 2011.

[8] C. Qi, L. Cheng, and X. Zhen, "Improving MapReduce Performance Using Smart Speculative Execution Strategy," IEEE Transactions on Computers, vol. 63, pp. 954967, 2014.

[9] L. Longbin, Z. Jingyu, Z. Long, L. Huakang, L. Yanchao, T. Feilong, et al.,"ShmStreaming: A Shared Memory Approach for Improving Hadoop Streaming Performance," in IEEE 27th International Conference on Advanced Information Networking and Applications (AINA), pp. 137-144, 2013.

[10] J. B. Buck, N. Watkins, J. LeFevre, K. Ioannidou, C. Maltzahn, N. Polyzotis, et al., "SciHadoop: Array-based query processing in Hadoop," in International Conference for High Performance Computing, Networking, Storage and Analysis (SC), pp. 1-11, 2011.

[11] Y. Xiao and H. Bo, "Bi-Hadoop: Extending Hadoop to Improve Support for Binary-Input Applications," in 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp. 245-252, 2013.

[12] S. Leo, F. Santoni, and G. Zanetti, "Biodoop: Bioinformatics on Hadoop, Parallel Processing Workshops," International Conference on Parallel Processing Workshops, pp. 415-422, 2009.

[13] C. Vorapongkitipun and N. Nupairoj, "Improving performance of small-file accessing in Hadoop," in 11th International Joint Conference on Computer Science and Software Engineering (JCSSE), pp. 200-205, 2014.

[14] L. Xuhui, H. Jizhong, Z. Yunqin, H. Chengde, and H. Xubin, "Implementing WebGIS on Hadoop: A case study of improving small file I/O performance on HDFS," in IEEE International Conference on Cluster Computing and Workshops, CLUSTER '09, pp. 1-8, 2009.

[15] M. Hammoud and M. F. Sakr, "Locality-Aware Reduce Task Scheduling for MapReduce," in IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom), pp. 570-576, 2011.

[16] B. Palanisamy, A. Singh, L. Liu, and B. Jain, "Purlieus: locality-aware resource allocation for MapReduce in a cloud," in Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 58-71, 2011.

[17] I. Elghandour and A. Aboulnaga, "ReStore: reusing results of MapReduce jobs," Proc. VLDB Endow, vol. 5, pp. 586-597, 2012.

[18] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, "Improving MapReduce Performance

in Heterogeneous Environments," in OSDI, pp. 1-7, 2008.

[19] J. Xie, S. Yin, X. Ruan, Z. Ding, Y. Tian, J. Majors, et al., "Improving mapreduce performance through data placement in heterogeneous hadoop clusters," in IEEEInternational Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), pp. 1-9, 2010.

[20] M. Y. Eltabakh, Y. Tian, Fatma, #214, zcan, R. Gemulla, et al., "CoHadoop:flexible data placement and its exploitation in Hadoop," Proc. VLDB Endow., vol.4, pp. 575-585, 2011.

[21] S. Nishanth, B. Radhikaa, T. J. Ragavendar, C. Babu, and B. Prabavathy, "CoHadoop++: A load balanced data colocation in Hadoop Distributed File System," in Fifth International Conference on Advanced Computing (ICoAC), pp. 100-105, 2013.

