International Journal of Computer Systems (ISSN: 2394-1065), Volume 02– Issue 05, May, 2015
Available at http://www.ijcsonline.com/

Map Reducing in Big Data Using Hadoop

Tarunpreet Chawla[1], Neeraj Mangalani[2], Tarannum Sheikh[3]

[1] *Department of Computer Science Engineering Jagannath University, Chaksu, Jaipur, Rajasthan, India.*
[2] *Department of Computer Science Engineering, Jagannath university, Chaksu, Jaipur, Rajasthan, India.*
[3] *Department of Computer Science Engineering, VIT Jaipur, India.*

Abstract

Big data is used to describe a massive volume of both structured and unstructured data that is so large that it's difficult to process using traditional database and software techniques. Big data usually includes data sets with sizes beyond the ability of commonly used software tools to capture, curate, manage, and process data within a tolerable elapsed time. There are various challenges in big data. In this, we use a framework of map reducing using hadoop. MapReduce is a programming model and an associated implementation for processing and generating large data sets with a parallel, distributed algorithm on a cluster. Hadoop is an open-source software framework for distributed storage and distributed processing of big data on clusters of commodity hardware.

Keywords: *bigdata, hadoop, mapreduce, cluster*

I. INTRODUCTION (BIGDATA)

In recent years, Big data is an all-encompassing term for any collection of data sets so large and complex that it becomes difficult to process them using traditional data processing applications. The challenges include analysis, capture, search, sharing, storage, transfer, visualization, and privacy violations. Big data "size" is a constantly moving target, as of 2012 ranging from a few dozen tera-bytes to many peta-bytes of data. Big data is a set of techniques and technologies that require new forms of integration to uncover large hidden values from large data sets that are diverse, complex, and of a massive scale.[1]
We use this "3Vs" model for describing big data as Velocity, Variety, Volume. Additionally another property of variability, veracity & complexity.

Volume – The quantity of data that is generated is very important in this context. It is the size of the data which determines the value and potential of the data under consideration and whether it can actually be considered as Big Data or not. The name „Big Data" itself contains a term which is related to size and hence the characteristic [2].

Variety - The next aspect of Big Data is its variety. This means that the category to which Big Data belongs to is also a very essential fact that needs to be known by the data analysts. This helps the people, who are closely analyzing the data and are associated with it, to effectively use the data to their advantage and thus upholding the importance of the Big Data.

Velocity - The term „velocity" in the context refers to the speed of generation of data or how fast the data is generated and processed to meet the demands and the challenges which lie ahead in the path of growth and development.

Variability - This is a factor which can be a problem for those who analyze the data. This refers to the inconsistency which can be shown by the data at times, thus hampering the process of being able to handle and manage the data effectively.

Veracity - The quality of the data being captured can vary greatly. Accuracy of analysis depends on the veracity of the source data.

Complexity - Data management can become a very complex process, especially when large volumes of data come from multiple sources. These data need to be linked, connected and correlated in order to be able to grasp the information that is supposed to be conveyed by these data. This situation, is therefore, termed as the „complexity" of Big Data [3][4].

A. What is hadoop

Apache Hadoop is an open-source software framework for distributed storage and distributed processing of Big Data on clusters of commodity hardware. Its Hadoop Distributed File System (HDFS) splits files into large blocks (default 64MB or 128MB) and distributes the blocks amongst the nodes in the cluster. For processing the data, the Hadoop Map/Reduce ships code (specifically Jar files) to the nodes that have the required data, and the nodes then process the data in parallel. This approach takes advantage of data locality, in contrast to conventional HPC architecture which usually relies on a parallel file system (compute and data separated, but connected with high-speed networking) Hadoop changes the economics and the dynamics of large scale computing. Its impact can be boiled down to four salient characteristics. Hadoop solution is Scalable, Cost effective, flexible, fault tolerant. MapReduce is a programming model and an associated implementation for

processing and generating large data sets with a parallel, distributed algorithm on a cluster. A MapReduce program is composed of a Map() procedure that performs filtering and sorting (such as sorting students by first name into queues, one queue for each name) and a Reduce() procedure that performs a summary operation (such as counting the number of students in each queue, yielding name frequencies). The "MapReduce System" (also called "infrastructure" or "framework") orchestrates the processing by marshaling the distributed servers, running the various tasks in parallel, managing all communications and data transfers between the various parts of the system, and providing for redundancy and fault tolerance. The model is inspired by the map and reduce functions commonly used in functional programming, although their purpose in the MapReduce framework is not the same as in their original forms. [5][6][7]The key contributions of the MapReduce framework are not the actual map and reduce functions, but the scalability and fault-tolerance achieved for a variety of applications by optimizing the execution engine once. As such, a single-threaded implementation of MapReduce (such as MongoDB) will usually not be faster than a traditional (non-MapReduce) implementation, any gains are usually only seen with multi-threaded implementations. Only when the optimized distributed shuffle operation (which reduces network communication cost) and fault tolerance features of the MapReduce framework come into play, is the use of this model beneficial.[8]

II. DISCUSSION

A. Hadoop distributed file system (HDFS)

The Hadoop distributed file system (HDFS) is a distributed, scalable, and portable file-system written in Java for the Hadoop framework. A Hadoop cluster has nominally a single namenode plus a cluster of datanodes, although redundancy options are available for the namenode due to its criticality. Each datanode serves up blocks of data over the network using a block protocol specific to HDFS. [9][10][11]The file system uses TCP/IP sockets for communication. Clients use remote procedure call (RPC) to communicate between each other. HDFS stores large files (typically in the range of gigabytes to terabytes) across multiple machines. The HDFS file system includes a so-called *secondary namenode*, a misleading name that some might incorrectly interpreted as a backup namenode for when the primary namenode goes offline. In fact, the secondary namenode regularly connects with the primary namenode and builds snapshots of the primary namenode's directory information, which the system then saves to local or remote directories. These checkpointed images can be used to restart a failed primary namenode without having to replay the entire journal of file-system actions, then to edit the log to create an up-to-date directory structure.[12] Because the namenode is the single point for storage and management of metadata, it can become a bottleneck for supporting a huge number of files, especially a large number of small files. HDFS Federation, a new addition, aims to tackle this problem to a certain extent by allowing multiple namespaces served by separate namenodes [13].

An advantage of using HDFS is data awareness between the job tracker and task tracker. The job tracker schedules map or reduce jobs to task trackers with an awareness of the data location[14]. For example: if node A contains data (x,y,z) and node B contains data (a,b,c), the job tracker schedules node B to perform map or reduce tasks on (a,b,c) and node A would be scheduled to perform map or reduce tasks on (x,y,z). This reduces the amount of traffic that goes over the network and prevents unnecessary data transfer.[15] When Hadoop is used with other file systems, this advantage is not always available. This can have a significant impact on job-completion times, which has been demonstrated when running data-intensive jobs. HDFS was designed for mostly immutable files and may not be suitable for systems requiring concurrent write-operations. HDFS can be mounted directly with a Filesystem in Userspace (FUSE) virtual file system on Linux and some other Unix systems [16].

File access can be achieved through the native Java API, the Thrift API to generate a client in the language of the users' choosing (C++, Java, Python, PHP, Ruby, Erlang, Perl, Haskell, C#, Cocoa, Smalltalk, and OCaml), the command-line interface, browsed through the HDFS-UI webapp over HTTP, or via 3rd-party network client libraries [17].

Hadoop works directly with any distributed file system that can be mounted by the underlying operating system simply by using a file:// URL; however, this comes at a price: the loss of locality. To reduce network traffic, Hadoop needs to know which servers are closest to the data; this is information that Hadoop-specific file system bridges can provide.

B. HDFS Architecture

The HDFS is designed to run on clustered computing platform. It mirrors the already existing file nomenclature in many ways but its differences really make it stands out from existing file systems (Fig. 4). One of the salient features of HDFS is that it is fault-tolerant to a very high degree and cost effective. The system allows for greater and faster access to data of an application which is an advantage for processes that require access to large amount of data. HDFS was designed by Apache Nutch project as an infrastructure extension and is now a core component of the project [18].

Name Node and Data Nodes: HDFS is based on a typical master - slave architecture. An HDFS cluster is made up of a single Name Node and a server acting as a master managing the file access and name space regulations. To simplify the system architecture a single name node exists in a cluster. The Name Node holds & manages whole metadata of HDFS. The design of the systems is such that the data does not flow through the Name Node

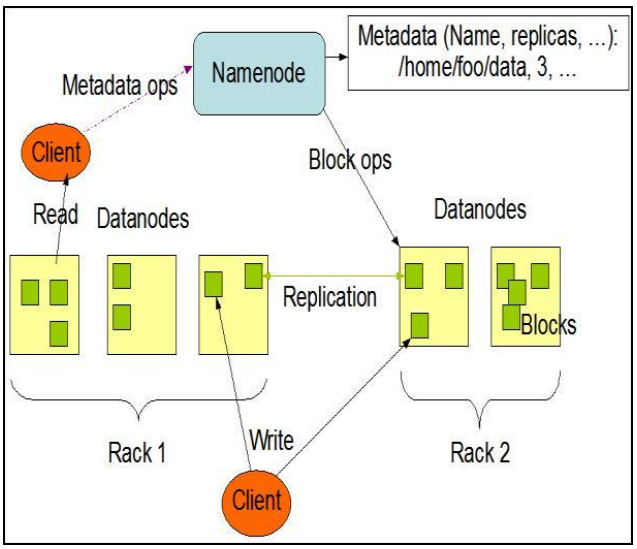

Figure 1 : Metadata

Data Replication: HDFS is programmed to manage last file stored in large cultures of data mines / structures while ensuring reliability. The way this is managed is by storing files in a sequence of blocks which are the same size, with the last block being an exception. These blocks are then replicated to test fault tolerance in which the size of the block and the replication factors are configurable. An application can then custom specify the number of copies of a file.[19]

C. Map-Reducing

The Apache Hadoop projects provide a series of tools designed to solve big data problems. The Hadoop cluster implements a parallel computing cluster using inexpensive commodity hardware. The cluster is partitioned across many servers to provide a near linear scalability. The philosophy of the cluster design is to bring the computing to the data. So each data node will hold part of the overall data and be able to process the data that it holds. The overall framework for the processing software is called MapReduce [20]. Apache Hadoop can be useful across a range of use cases spanning virtually every vertical industry. It is becoming popular anywhere that you need to store, process, and analyze large volumes of data. Examples include digital marketing automation, fraud detection and prevention, social network and relationship analysis, predictive modeling for new drugs, retail in-store behavior analysis, and mobile device location-based marketing.

MapReduce is a framework for processing parallelism problems across huge datasets using a large number of computers (nodes), collectively referred to as a cluster (if all nodes are on the same local network and use similar hardware) or a grid (if the nodes are shared across geographically and administratively distributed systems, and use more heterogeneous hardware). Processing can occur on data stored either in a filesystem (unstructured) or in a database (structured). MapReduce can take advantage of locality of data, processing it on or near the storage assets in order to reduce the distance over which it must be transmitted.

"Map" step: Each worker node applies the "map()" function to the local data, and writes the output to a temporary storage. A master node orchestrates that for redundant copies of input data, only one is processed.

- **"Shuffle" step:** Worker nodes redistribute data based on the output keys (produced by the "map()" function), such that all data belonging to one key is located on the same worker node.
- **"Reduce" step:** Worker nodes now process each group of output data, per key, in parallel.

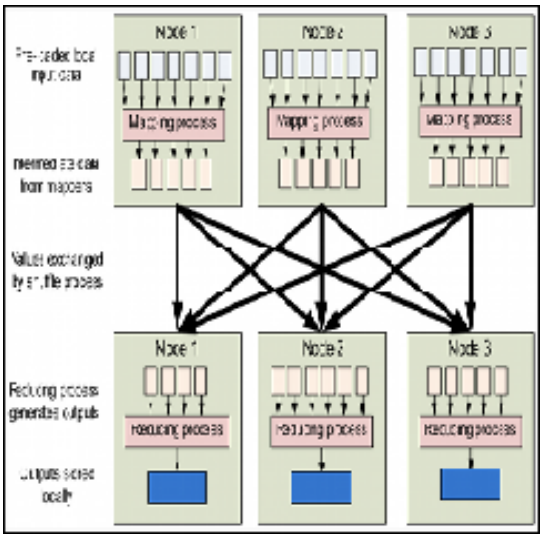

Figure 2 : Map Reducing

MapReduce allows for distributed processing of the map and reduction operations. Provided that each mapping operation is independent of the others, all maps can be performed in parallel – though in practice this is limited by the number of independent data sources and/or the number of CPUs near each source. Similarly, a set of 'reducers' can perform the reduction phase, provided that all outputs of the map operation that share the same key are presented to the same reducer at the same time, or that the reduction function is associative. While this process can often appear inefficient compared to algorithms that are more sequential, MapReduce can be applied to significantly larger data sets than "commodity" servers can handle – a large server farm can use MapReduce to sort a petabyte of data in only a few hours. The parallelism also offers some possibility of recovering from partial failure of servers or storage during the operation: if one mapper or reducer fails, the work can be rescheduled – assuming the input data is still available.

The *Map* and *Reduce* functions of *MapReduce* are both defined with respect to data structured in (key, value) pairs. Map takes one pair of data with a type in one data domain, and returns a list of pairs in a different domain:

$$Map(k1,v1) \rightarrow list(k2,v2)$$

The Map function is applied in parallel to every pair in the input dataset. This produces a list of pairs for each call. After that, the MapReduce framework collects all pairs with the same key from all lists and groups them together, creating one group for each key.

The Reduce function is then applied in parallel to each group, which in turn produces a collection of values in the same domain:

$$Reduce(k2, list\ (v2)) \rightarrow list(v3)$$

Each *Reduce* call typically produces either one value v3 or an empty return, though one call is allowed to return more than one value. The returns of all calls are collected as the desired result list.

Thus the MapReduce framework transforms a list of (key, value) pairs into a list of values. This behavior is different from the typical functional programming map and reduce combination, which accepts a list of arbitrary values and returns one single value that combines *all* the values returned by map.

It is necessary but not sufficient to have implementations of the map and reduce abstractions in order to implement MapReduce. Distributed implementations of MapReduce require a means of connecting the processes performing the Map and Reduce phases. This may be a distributed file system.[21] Other options are possible, such as direct streaming from mappers to reducers, or for the mapping processors to serve up their results to reducers that query them.

The file systems comes the MapReduce engine, which consists of one *JobTracker*, to which client applications submit MapReduce jobs. The JobTracker pushes work out to available *TaskTracker* nodes in the cluster, striving to keep the work as close to the data as possible. With a rack-aware file system, the JobTracker knows which node contains the data, and which other machines are nearby. If the work cannot be hosted on the actual node where the data resides, priority is given to nodes in the same rack. This reduces network traffic on the main backbone network. If a TaskTracker fails or times out, that part of the job is rescheduled. The TaskTracker on each node spawns off a separate Java Virtual Machine process to prevent the TaskTracker itself from failing if the running job crashes the JVM. A heartbeat is sent from the TaskTracker to the JobTracker every few minutes to check its status. The Job Tracker and TaskTracker status and information is exposed by Jetty and can be viewed from a web browser.If the JobTracker failed on Hadoop 0.20 or earlier, all ongoing work was lost. Hadoop version 0.21 added some check pointing to this process; the JobTracker records what it is up to in the file system. When a JobTracker starts up, it looks for any such data, so that it can restart work from where it left off.

- The allocation of work to TaskTrackers is very simple. Every TaskTracker has a number of available *slots* (such as "4 slots"). Every active map or reduce task takes up one slot. The Job Tracker allocates work to the tracker nearest to the data with an available slot. There is no consideration of the current system load of the allocated machine, and hence its actual availability.
- If one TaskTracker is very slow, it can delay the entire MapReduce job – especially towards the end of a job, where everything can end up waiting for the slowest task. With speculative execution enabled, however, a single task can be executed on multiple slave nodes.

The following diagram shows the logical flow of a MapReduce programming model:

- **Input:** This is the input data / file to be processed.
- **Split:** Hadoop splits the incoming data into smaller pieces called "splits".
- **Map:** In this step, MapReduce processes each split according to the logic defined in map() function. Each mapper works on each split at a time. Each mapper is treated as a task and multiple tasks are executed across different Task Trackers and coordinated by the JobTracker.
- **Combine:** This is an optional step and is used to improve the performance by reducing the amount of data transferred across the network. Combiner is the same as the reduce step and is used for aggregating the output of the map() function before it is passed to the subsequent steps.
- **Shuffle & Sort:** In this step, outputs from all the mappers is shuffled, sorted to put them in order, and grouped before sending them to the next step.
- **Reduce:** This step is used to aggregate the outputs of mappers using the reduce() function. Output of reducer is sent to the next and final step. Each reducer is treated as a task and multiple tasks are executed across different TaskTrackers and coordinated by the JobTracker.
- **Output:** Finally the output of reduce step is written to a file in HDFS.

Here are few highlights of MapReduce programming model in Hadoop:

- MapReduce works in a master-slave / master-worker fashion. JobTracker acts as the master and TaskTrackers act as the slaves.

- MapReduce has two major phases - A Map phase and a Reduce phase. Map phase processes parts of input data using mappers based on the logic defined in the map() function. The Reduce phase aggregates the data using a reducer based on the logic defined in the reduce() function.
- Depending upon the problem at hand, we can have One Reduce Task, Multiple Reduce Tasks or No Reduce Tasks.
- MapReduce has built-in fault tolerance and hence can run on commodity hardware.
- MapReduce takes care of distributing the data across various nodes, assigning the tasks to each of the nodes, getting the results back from each node, re-running the task in case of any node failures, consolidation of results, etc.
- MapReduce processes the data in the form of (Key, Value) pairs. Hence, we need to fit out business problem in this Key-Value arrangement.

III. CONCLUSION

Hadoop with its distributed file system & programming framework based on concept of mapped reduction, is a powerful tool to manage large data sets. In this paper, we discussed map reduce over a hadoop cluster by using streaming libraries of hadoop. We can effectively use Linux shell command to extract or to compute the desired result. With its map-reduce programming paradigms, overall architecture, ecosystem, fault- tolerance techniques and distributed processing, Hadoop offers a complete infrastructure to handle Big Data. Users must leverage the benefits of Big-Data by adopting Hadoop infrastructure for data processing. However, the issues such as lack of flexible resource management, application deployment support, and multiple data source support pose a challenge to Hadoop‟s adoption.

REFERENCES

[1] Jens Dittrich, Stefan Richter and Stefan Schuh, " Efficient OR Hadoop: Why Not Both?," Datenbank-Spektrum, Volume 13, Issue 1 , pp 17-22

[2] Humbetov, S, "Data-Intensive Computing with Map-reduce and Hadoop," in Proc. 2012 Application of Information and Communication Technologies (AICT), IEEE ,6th International Conference pp. 5

[3] Hadoop Tutorial, Apache Software Foundation, 2014, Available: http://hadoop.apache.org/

[4] Sherif Sakr, Anna Liu and Ayman G. Fayoumi, " The family of mapreduce and large-scale data processing systems," ACM Computing Surveys, Volume 46 Issue 1, October 2013, Article No. 11

[5] Aditya B. Patel, Manashvi Birla and Ushma Nair, " Addressing Big Data Problem Using Hadoop and Map Reduce," in Proc. 2012 Nirma University International Conference on Engineering, pp. 1-5.

[6] Abouzeid, A., Bajda-Pawlikowski, K., Abadi, D., Rasin, A., and Silberschatz, A. 2009. HadoopDB: An architectural hybrid of mapreduce and dbms technologies for analytical workloads. Proc. VLDB Endow. 2, 1,922–933.

[7] Abouzeid, A., Bajda-Pawlikowski, K., Abadi, D., Rasin, A., and Silberschatz, A. 2010. HadoopDB in action: Building real world applications. In Proceedings of the 36th ACM SIGMOD International Conference on Management of Data (SIGMOD‟10).

[8] Parallel Data Processing with MapReduce: A Survey: www.cs.arizona.edu/~bkmoon/papers/sigmodrec 11.pdf

[9] Jyoti Nandimath, Ankur Patil, Ekata Banerjee,Pratima Kakade and Saumitra Vaidya, " Big Data Analysis Using Apache Hadoop," IEEE IRI 2013, August 14-16, 2013, San Francisco, California, USA

[10] Jens Dittrich and JorgeArnulfo Quian´eRuiz, " Efficient Big Data processing in Hadoop Mapreduce," Proceedings of the VLDB Endowment, Volume 5 Issue 12, August 2012, Pages 2014-2015

[11] MapReduce: Simplified Data Processing on Large Clusters. Available at http://labs.google.com/papers/mapreduceosdi04.pdf

[12] Stephen Kaisler, Frank Armour, J. Alberto Espinosa, William Money,“Big Data: Issues and Challenges Moving Forward”, IEEE, 46th Hawaii International Conference on System Sciences, 2013. Sam Madden, “ From Databases to Big Data”, IEEE, Internet Computing, May-June 2012.

[13] Yuri Demchenko, Zhiming Zhao, Paola Grosso, Adianto Wibisono, Cees de Laat, “Addressing Big Data Challenges for Scientific Data Infrastructure”, IEEE , 4th International Conference on Cloud Computing Technology and Science, 2012.

[14] HDFS Architecture Guide [Online] Available: http://hadoop.apache.org/docs/hdfs/current/hdfs_design

[15] Levy E. and Silberschatz A., "Distributed FileSystems:Concepts and Examples"

[16] Apache Hadoop-Petabytes and Terawatts [Online].Available:http://www.youtube.com/watch?v=SS27FhYWfU& feature=related

[17] Jeffrey Dean and Sanjay Ghemawat. "Mapreduce:simplified data processing on large clusters", Commun. ACM, 51(1):107–113, 2008.

[18] Sanchita Kadambari, Praveen Kumar and Seema Rawat " A comprehensice study on Big Data and its future opportunities," in Proc. 2014 Fourth International Conference on Advanced Computing & Communication Technologies, pp. 277-281

[19] Jeffrey Dean and Sanjay Ghemawat, " MapReduce: a flexible data processing tool," Communications of the ACM, Volume 53 Issue 1, January 2010, Pages 72-77.

[20] T. Bharathi, Sri K.Sreenivasulu, “A Bigdata Sets Approach for Data Anonymization Using Map-Reduce on Cloud”, In International Journal of Computer Systems, pages: 141-145, Volume 02– Issue 04, April, 2015.

[21] “Big Data: The next frontier for innovation, competition, and productivity”, McKinsey Global Institute, May 2011, p. 11: http://wwwmckinsey.com/Insights/MGI/Research/Technology_and _Innovation/Big_data_The_next_frontier_for_innovation.