Big Data and Hadoop

Sreedhar C, Dr. D. Kavitha, K. Asha Rani

Abstract— **Big data has become a buzzword in the recent years. Big data is used to describe a massive volume of both structured and unstructured data that is so large that its difficult to process using traditional database and software techniques. Big data is a collection of massive and complex data sets that include the huge quantities of data, social media analytics, data management capabilities and real time data. Big data refers to various forms of large information sets that require special computational platforms in order to be analyzed. Big data is characterized by three Vs namely volume, variety and velocity. Hadoop framework supports the processing of large data sets in a distributed computing environment. Hadoop is the core of the Hadoop File System and MapReduce, well designed to handle huge volumes of data across a large number of nodes. This paper presents the processing functionality of Hadoop to distribute computational power in processing the massive data.**

Index Terms—**Big data, Hadoop, HDFS and MapReduce.**

I. INTRODUCTION

Although the term big data is in widespread use, there is no rigorous definition of big data [12]. Big data refers to the datasets whose size is beyond the ability of typical database software tools to capture, store, manage and analyse. This definition is intentionally subjective and incorporates a moving definition of how big a dataset needs to be in order to be considered big data—i.e., big data cannot be defined in terms of being larger than a certain number of terabytes (thousands of gigabytes). We assume that, as technology advances over time, the size of datasets that qualify as big data will also increase. Also note that the definition can vary by sector, depending on what kinds of software tools are commonly available and what sizes of datasets are common in a particular industry. With those caveats, big data in many sectors today will range from a few dozen terabytes to multiple petabytes (thousands of terabytes). According to McKinsey & Co [11], big data is the next frontier for innovation, competition and productivity. Big data is the term for describing these large and complex sets of data that can be problematic to process with traditional database management systems. An example of big data is the Large Hadron Collider, at the European Organization for Nuclear Research (CERN), which has 150 million sensors and is creating 22 petabytes of data in 2012. Overview of data scale from megabytes to yottabytes is shown in figure 1. Today, big data and Hadoop as a big data software solution are almost always talked about concurrently. Hadoop is a generally new framework developed to use a distributed file system to take advantage of parallel processing in processing big data.

The underlying technology of Hadoop was actually developed by Google. Google needed a way to index all its complex and unstructured data it was gathering from crawling through multitudes of web pages. Two Google engineers developed Google Map Reduce (GMR) and Google File System, which would come to handle all the processing of data in order to index and rank web pages for the search services we know as google.com. This framework would eventually become open source with Yahoo and Apache taking the forefront in developing enterprise applications for and managing this open-source framework.

8 bits	1 Byte
1000 Bytes	1 Kilobyte
10000 Kilobytes	1 Megabyte
1000 Megabytes	1 Gigabyte
1000 Gigabytes	1 Terabyte
1000 Terabytes	1 Petabyte
1000 Petabytes	1 Exabyte
1000 Exabytes	1 Zettabyte
1000 Zettabytes	1 yottabyte

Fig. 1. Overview of data scale

II. HADOOP AND ITS COMPONENTS

A. Overview of Hadoop

Hadoop is a free, Java-based programming framework that supports the processing of large data sets in a distributed computing environment. It is part of the apache project sponsored by the Apache Software Foundation. Hadoop's contributors work for some of the world's biggest technology companies. That diverse, motivated community has produced a genuinely innovative platform for consolidating, combining and understanding large-scale data in order to better comprehend the data deluge. Two key components of Hadoop include Map Reduce and the Hadoop Distributed File System (HDFS). The Hadoop framework uses and manages these two components to distribute data across multiples and multiples of servers and parallel process them. However, it is not only capable of processing huge amounts of data, but also able to process heterogeneous types of data. One of the main advantages of Hadoop is that it is able to process complex and unstructured data. Through these

Manuscript received May, 2014.
***Sreedhar C**, Associate Professor, CSE Department, G. Pulla Reddy Engineering College, Kurnool, India.*
***Dr. D. Kavitha**, Professor, CSE Dept., GPREC, Kurnool.*
***K. Asha Rani**,Sr.Asst. Prof., CSE Dept., GPREC, Kurnool*

ISSN: 2278 – 1323

descriptions of Hadoop, now we can see that Hadoop embraces heterogeneity through the whole procession of data ranging from heterogeneous data being processed to the heterogeneous hardware processing the data.

III. HADOOP DISTRIBUTED FILE SYSTEM (HDFS)

A. Overview of HDFS

HDFS is a distributed, scalable, and portable filesystem written in Java for the Hadoop framework. Each node in a Hadoop instance typically has a single datanode; a cluster of datanodes form the HDFS cluster. The situation is typical because each node does not require a datanode to be present. Each datanode serves up blocks of data over the network using a block protocol specific to HDFS. HDFS is highly scalable as well as highly fault tolerant. It provides these two attributes by allowing data to scale out easily and economically with the adding of cheap servers. These servers hold the increasing amounts of data being gathered as well as multiple copies to ensure reliability. Scalability is further enhanced because HDFS is designed to work on heterogeneous hardware and software platforms. This attribute erases the limitations of having to stick with certain types of hardware or replacing legacy hardware to keep the file systems homogenous. HDFS is built with JAVA and provides a JAVA API as well as a C language API for interfacing with client code. These highly portable languages allow any machine that supports JAVA or C (almost all machines) to use HDFS.

With regards to efficiency, HDFS not only provides a way of slicing the data into multiple pieces but also provides ways to access this data so that the processing logic is sliced as well; each and every node contains a piece of the file system and logic so that applications can interface with them. By using this type of architecture, this innovation is able to allow the HDFS system to follow the principle that it is less costly to move computation to the data than it is to move data to computation. This is done by HDFS being rack-aware of data nodes and the data blocks within them. HDFS will always try to use the data block closest to the reader. Because using a data node in a different rack from the reader would incur network transportation costs, we can see how HDFS tries to eliminate this cost by using the data closest to the reader.

HDFS also provides cluster rebalancing. This feature is important in making sure that space on the data node is not under-utilized. It also provides uniform storage of data across all nodes. Another criterion for rebalancing a set of nodes would be if the data on a particular set of nodes had high demand. Replication of nodes with high demand would increase parallelism by enabling processing of this data in two multiple locations.

Delving further into fault tolerance, HDFS actually has a couple solutions that ensure that this goal is achieved. The first method is detection of failed nodes. Failed nodes can be common when dealing with hundreds and thousands of data nodes. Some of these failures occur from network connectivity issues as well as hard disk failures. In order to detect a failed node, a data node must send a heartbeat to the main node indicating it is still functional. Absence of this heart beat will have the main node mark the data node as failing and cease any input/output requests to the node. In order to detect corruption of data, HDFS implements and stores a checksum on the data when an HDFS file is created. The checksum is checked again when an HDFS client attempts to retrieve that block of data. If the checksum does not match, HDFS may attempt to send the client the data block requested from a replicate source.

B. Architecture of HDFS

In HDFS, each cluster contains of a single name node that manages multiple data nodes. The name node manages all the data nodes essentially creating master/slave architecture, shown in figure 2. The file system namespace is maintained by the name node. Users can create, open, rename, or delete files and directories just like in any other file system, and the name node will distribute the instructions to the proper data node. An HDFS installation will usually configure a web server so that the user can interface with the file system through a web browser. Internally, when a file is created, it is actually sliced into blocks of data, each going into the same or separate data nodes. Therefore as you can see, another main duty of the name node is to inherently map and store the mapping of each block of data on where it is located in relation to the data node. It is also in charge of regulating client read and write requests to the data. Currently, HDFS only supports a single writer and multiple reader architecture. HDFS strictly enforces this policy to prevent data corruption and race conditions.

As we discussed before, the HDFS architecture is a master/slave architecture. Data nodes continuously ping the name node for instructions. Their main purpose is to serve read and write requests to the client. However, the name node may also access the data for replication or file system management purposes. In this case, the data node will perform the given instructions of the name node to create, delete, or replicate a data block. As mentioned previously, data nodes send a heartbeat to indicate that it is functioning correctly. It also sends a blockreport, which serves to list all the data blocks that is contained within the data node.

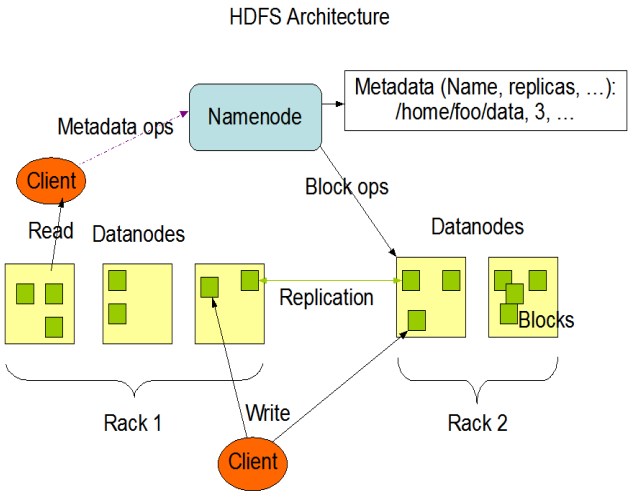

Fig. 2. HDFS Architecture

ISSN: 2278 – 1323

IV. MAPREDUCE

A. Overview of MapReduce

MapReduce is the overall software framework that allows applications to implement processing of data through a cluster of nodes through a distributed processing algorithm. The MapReduce system is divided into two parts: Map and then Reduce. The mapping portion divides and distributes a large job across the worker nodes of a cluster. When the previous portion is complete, the reduce function aggregates all results and in a sense reduces them into one complete whole. This final result is the answer to the original query. MapReduce introduces one key innovation in the area of fault tolerance. The master node will ping the worker node for a response. Failure to respond will indicate that to the master node that the worker node is unavailable. The master node will then redistribute this worker node's assigned work to another worker node.

The most important attribute to remember about MapReduce is that every map and reduce function is independent of each other. All of the processing is occurring on the separate nodes where the data is located. The input from the each map function is temporarily saved where the reducers can come in to consolidate the output

B. The Map Reduce Function

The MAP function is actually a little more complicated than previously described. Mapping takes the unstructured data and rearranges them into a format that will match the final result. The results of the mapping is actually used as input for the REDUCE function. In order to produce this format, the MAP function is actually divided into two steps. It first assigns a key (K1) to sort all the data by and then a processor to work on each set of data using those keys. The output from this creates a list based on another key (K2), which is a group key for all the pairs of the same key, shown in figure 3. In a logical sense, we can view the function like so: map(key1,value) -> list<key2,value2>.

Reduce then takes this list based on K2 and aggregates them. This is because since each mapping function is only a portion of the original job, it is a large possibility that each one may have outputted the same key. The Reduce function is needed to aggregate the outputs of all the map functions. It processes the inputs from the map functions and creates a new list to form the final output. In a logical view, it is seen like so: reduce(key2, list<value2>) -> list<value3>.

C. Components of MapReduce

MapReduce is a framework for processing highly distributable problems across huge datasets using a large number of computers (nodes), collectively referred to as a cluster (if all nodes use the same hardware) or a grid (if the nodes use different hardware). Computational processing can occur on data stored either in a filesystem (unstructured) or in a database (structured). To follow the data flow of map reduce, we need to look at some of the components map reduce and their functionality.

The first stop would be the input reader. The input reader takes all the data from the source and generates key/value pairs for the input of the map function. It splits the data into sections for each map function as well. From there, the map function once again takes these values of keys and pairs to generate another list of keys and pairs. A partition function is then utilized to distribute the output of the map function to the reducers. While this is happening, a comparison function sorts the input of the map function to make it ready for the reducers.

The reduce function as previously mentioned goes through the input, grouping them with the unique key provided to create an output. An output writer then writes the output from the reduce function back to the source.

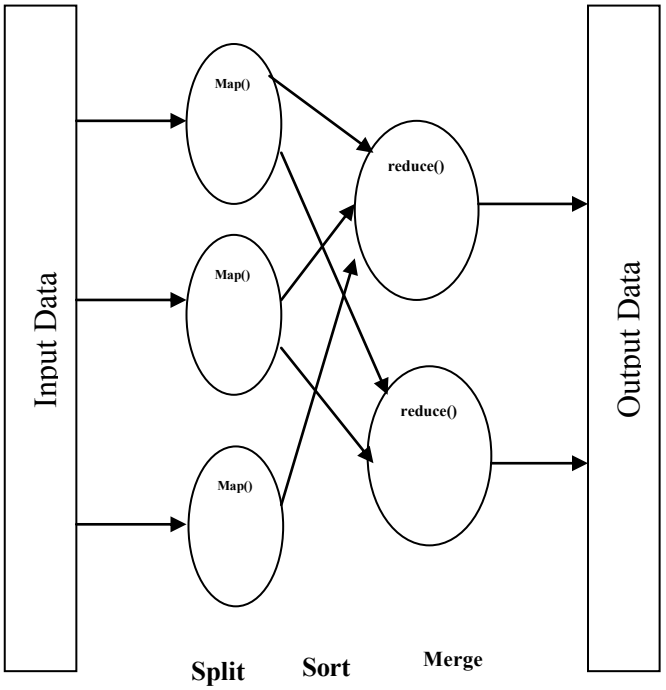

Fig. 3. Input is split into multiple map functions which is then outputted onto multiple map functions

V. HADOOP THE OVERALL SYSTEM

Hadoop provides a distributed file system and a framework for the analysis and transformation of very large data sets using MapReduce paradigm. An important characteristic of Hadoop is the partitioning of data and computation across many of hosts, and the execution of application computations in paralled close to their data. A Hadoop cluster scales computation capacity, storage capacity and I/O bandwidth by simply adding commodity servers. Hadoops manages its two core components. It consists of a Job Tracker which receives the Job Configuration from the client. The Job Tracker will then split the job up into tasks, which are sent to Task Trackers. Task trackers then have the job of actually performing the map and reduce functions on the data. Finally, all of this is possible through the HDFS, which allows each data node to perform map and reduce functions.

The overall view of Hadoop using MapReduce and HDFS to perform a job is shown in figure 4. The following describes

ISSN: 2278 – 1323

the execution of a job:

- Client program uploads files to the HDFS location and notifies the JobTracker which in turn returns the JobID to the client.
- The JobTracker allocates map tasks to the TaskTrackers.
- JobTracker determines appropriate jobs based on how busy the TaskTracker is.
- TaskTracker forks MapTask which extracts input data and invokees the user provided "map" function which fills in the buffer with key/value pairs until it is full.
- The buffer is eventually flushed into two files.
- After all the MapTask completes (all splits are done), the TaskTracker will notify the JobTracker which keeps track of the overall progress of job.
- When done, the JobTracker notifies TaskTracker to jump to reduce phase. This again follows same method where reduce task is forked.
- The output of each reducer is written to a temporary output file in HDFS. When the reducer finishes processing all keys, the temp output file will be renamed atomically to its final output filename.

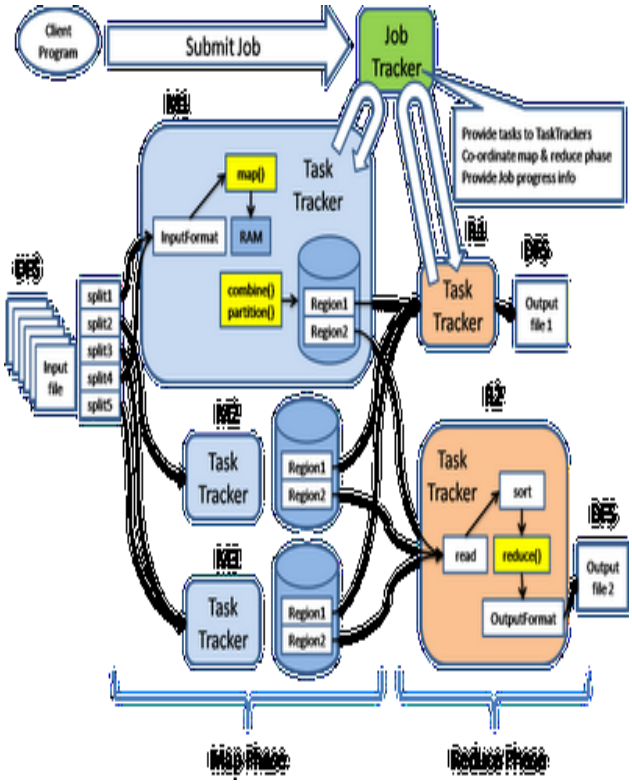

Fig. 4. Overall view of Hadoop using MapReduce and HDFS to perform a Job

VI. Benefits and Limitations

A. Hadoop Vs Traditional Databases

There are many key advantages of Hadoop over a traditional database system. First is the scalability aspect. Hadoop allows the system to scale horizontally with the use of commodity servers, while in a traditional database you would scale vertically with larger and more powerful super computers/processors.

Hadoop also manages fault tolerance and failures much better than traditional databases. The traditional database may have all its data within one or several servers. However, if one server were to go down, a huge section of data would be missing. Hadoop solves this by replicating blocks of data across multiple data nodes. Hadoop's Job Tracker also keeps track of when a Task Request has failed. If a failure in the task tracker were to occur, it reassigns that task tracker's work to another ensuring that the batch job is complete.

Even against other distributed database systems, Hadoop matches up very well. In many of these traditional distributed systems, the processing logic is located separately from the actual data storage.

In order for computation to occur, the data set is transported from the storage to the processing computer. While in smaller data sets this is not a problem, this is a huge problem for big data due to the bandwidth of having to transport such data. There is also the problem of having to store temporary computational data sets. A data set alone can be in the terabytes or more. The intermediate data sets generated during computation can easily be several sizes above the original data set.

Hadoop resolves this issue by dividing data and processing into each node. This not only solves the network/bandwidth issue of having to transport huge chunks of data, but also solves the memory issue of having to reserve huge chunks of memory to perform a large scale computation on a huge set of data.

Another key attribute of Hadoop is that it is designed to work with unstructured and complex data. Traditional databases currently use structured data in the form of tables to create relational databases.

B. Limitations

There are however, some limitations on what Hadoop can do. It is definitely not the end all solution for every enterprise. Hadoop was designed with batch job architecture and as such it does not provide real-time access of data.

Moreover, it can also be very inefficient for processing small amounts of data. The Map Reduce framework is also not particularly faster than an indexed database. Because Hadoop is meant for processing huge chunks of unindexed data, it could also be slower than indexed database when repeating the same jobs or requests as an indexed database would certainly increase its speed significantly.

The Hadoop system can also underperform for clusters with ten or less nodes. There is quite a bit of overhead needed to merge and sort results. As a result of this, traditional databases would have the advantage in not having to perform these extra operations. This is more apparent when a data set is smaller in size.

VII. Conclusion

Research on Big data emerged in the 1970s but has seen an explosion of publications since 2008. In conclusion, enterprises should be sure they need Hadoop before they

ISSN: 2278 – 1323

attempt to migrate their database system. Hadoop's main advantage is its ability to take on large data sets all while providing easy scalability. As such, it is when large clusters of data nodes are created, that Hadoops starts providing the greatest return. With that being said, Hadoop and big data are in today's industry almost synonymous. After looking at all the key aspects that make Hadoop what it is, we can easily see why.

REFERENCES

[1] Mike Miller, Cloudant, "Why the days are numbered for Hadoop as weknow",http://gigaom.com/2012/07/07/hy-the-days-are-numbered-for-hadoop- as-we-know-it/, July 2012.
[2] Dhruba Borthakur, "The Hadoop Distributed File System: Architecture and Design", 2007.
[3] J. Jeffrey Hanson, "An Introduction to the Hadoop Distributed File Syste: Explore HDFS framework and subsystems", *developer Works*, Feb. 2011.
[4] Ricky Ho, "How Hadoop Map/Reduce Works", http://architects. dzone.com /articles/how-hadoop-mapreduce-works.
[5] O'Reilly,"strata.oreilly.com/2011/01/what-is-hadoop.htmlhttp:/architects. dzone.com/articles/how-hadoop-mapreduce-works", 2014.
[6] http://en.wikipedia.org/wiki/MapReduce
[7] IBM, "What is Map Reduce" , http://www-01.ibm.com/ software/data/infosphere/hadoop/mapreduce/
[8] Hadoop Wiki, "MapReduce", http://wiki.apache.org/ hadoop/MapReduce.
[9] Map Reduce Tutorial, https://hadoop.apache.org/docs/r1.2.1/ mapred_tutorial.html#Job+Submission+and+Monitoring.
[10] Apache Hadoop, "https://developer.yahoo.com/hadoop/ tutorial/module1.html#scope.
[11] McKinsey Global Institute, "Big Data: The next frontier for innovation, competition and productivity", June 2011.
[12] Mayer-Schonberger and Cukier, "Big Data: A revolution that will transform how we live, work and think", 2013.

Sreedhar C received B.E degree in Computer Science & Engineering from Madras University and M.E in Computer Science & Engineering degree. Presently, he is working as Associate Professor in Computer Science & Engineering Department at G Pulla Reddy Engineering College, Kurnool. His research interest areas include Big Data, Wireless Networks, Security and Robotics. He has published 12 papers in various National/International Journals/Conferences.

Dr. D. Kavitha received her Ph.D degree from S K University, Anantapur. At present she is working as Professor in Computer science & Engineering Department at G Pulla Reddy Engineering College, Kurnool. Her research areas include Wireless Networks, Mobile IP, and Security.

K. Asha Rani is presently working as Senior Assistant Professor in Computer Science & Engineering Department at G Pulla Reddy Engineering College, Kurnool. Her research areas include Wireless Networks, Data Mining.

ISSN: 2278 – 1323