ISSN 2348 – 9928
IJAICT Volume 5, Issue 12, December 2018 Doi:01.0401/ijaict.2015.04.02 Published on 05(12) 2018

HOW MAPREDUCE CAME INTO BEING: A SURVEY

Mr. Mir Ahmed Ali
Assistant Professor, Department of CSE,
Muffakham Jah College of Engineering and Technology,
Hyderabad, Telangana, India

Abstract— In this paper , we are giving a brief review on MapReduce which is a distributed programming paradigm proposed by Google in order to write applications which can process either small or large data sets. Moreover we will see advantages and disadvantages of MapReduce.

Keywords— Map,Reduce, Job,Task,Cluster, Hadoop, Distributed Computing, Cluster Computing

I. INTRODUCTION

Data are the values of qualitative or quantitative variables belonging to a set of items [1]. Examples of qualitative data are country of origin, treatment, gender and that of quantitative data is height, weight, blood pressure. If you want to perform any computation on the data, then the data is loaded into Main Memory and an algorithm is made to run by CPU which will access the data from the Main Memory. These algorithms are written using Machine Learning and statistics. But, if the data is too big and doesn't fit in the memory, then the concept of data mining comes into picture. A classical data mining algorithm gets portion of data into main memory from disk, process the data in the form of batches and writes results to the disk. But, if the size of the file is as big as 200TB and if the algorithm used is classical data mining algorithm then just to read 200TB of data from a disk, it roughly takes 46 days which is a very long time and is unacceptable. So, we need a better solution. The solution is splitting data into chunks and distributing these chunks into multiple disks, read the data parallely and allocate it to multiple CPU that is thousand disks are made to compute in parallel and the time taken is just an hour, which is an acceptable time. This is where cluster computing comes.

1.1 Cluster Computing Architecture

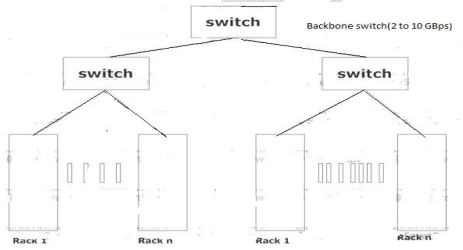

Fig 1: Cluster Computing Architecture

In cluster computing we have racks as shown in figure 1, consisting of commodity linux nodes. These nodes are connected by a switch which is a giga-bit switch whose bandwidth is 1GB/sec and each rack may contain 1 to 64 nodes Collection of racks is known as Data Center which is used for storing and mining large data sets. But our problem has not yet solved as it has its own challenges as discussed below:

1.1.1 Node failure

As discussed, since nodes are made of commodity hardware, they are prone to failure. A single node works upto 3 years that is 1000 days and once in 1000 days if a node fails then there is no problem. Likewise, if we have 1000 servers in a cluster and if anyone server fails in a day then it works out. But, if we have million servers in a cluster and 1000 of them will fail per day, then there is a problem which must be fixed. Now, let us consider about data residing in a node. If node fails, then data cannot be retrieved. In short, we are not able to achieve persistency. In another case, during computation, if node fails then again in this architecture we have to restart computation. So, we have to find a new architecture that should hide the failure and also make the computation complete though the node is failed.

1.1.2 Speed of Movement of Data

The second problem is during computation, if we have to move data between racks then it may lead to bottleneck, though the speed of backbone switch is more than 2GBps and speed between the racks is at most 1GBps. In case of complex computation, we require more data and in above scenario slows down the computation. So, we need a framework that increase the performance.

1.1.3 Programming used

The third problem is the programming used that is distributed programming is hard. Even sophisticated programmers face difficulty while writing programs as they have to keep off race condition and various kind of complication. So, we require a solution that hides most of the complexity of distributed programming and write algorithm that can mine large data sets.
So, the solution to all the above problems is MapReduce. Map Reduce solves all the above mentioned three challenges:

ISSN 2348 – 9928
Doi:01.0401/ijaict.2015.04.02 Published on 05(12) 2018

1. The problem of persistence is solved by creating multiple copies of same data and the problem of availability is solved by storing on multiple nodes that is if any one node fails, data is still available on another node.

2. Instead of moving data from disk to CPU, MapReduce framework moves computation to data and this minimize the bottle neck problem.

3. MapReduce provide simple programming model that hides complexity.

II. LITERATURE SURVEY ON MAPREDUCE

A concept of MapReduce is inspired from Map and Reduce primitives present in functional languages [3] like Lisp. The most popular implementation is the one introduced by Google in the year 2004, which utilizes large clusters of commodity computers connected with switched ethernet. Since Google MapReduce is a proprietory one and is not available to public, other implementations which are opensource and available to public are DISCO written in Erlang and the most popular Apache Hadoop and the next most popular is Elastic MapReduce by Amazon.

2.1 Redundant Storage Infrastructure

This concept is provided by Distributed File System which stores multiple copies of same data around clusters. In short DFS provide global file namespace, redundancy and availability. The most popular implementation of DFS are Google File System (GFS) and Hadoop Distributed File System(HDFS). As discussed above, when an input file comes and stores into DFS, it is divided into chunks and these chunks are distributed among multiple machines and these machines are called as Chunk Server. The replicas of each chunk is made and distributed over multiple chunk server in a way that the original copy and replica copy are not present on same machine. These chunk servers behave as compute server and the computation is send to the chunk present in the chunk server rather than moving data and by doing this we are avoiding unnecessary movement of data.

2.2 Components of DFS

2.2.1 Chunk Server

In Chunk Server, each file is split into contiguous chunks of size ranging from 16MB to 64MB. Each chunk is either replicated twice or thrice, but most oftenly thrice and these chunks are kept on different servers and one of these replicas of three chunks is kept on an entire different rack and this is done if all things are kept in same rack and if entire rack or switch fails, then it gets inaccessible.

2.2.2 Master Node

In HDFS, it is known as NameNode. It is a node that maintains metadata that keeps tracks about information of location of each file that if file1 is divided into four chunks then metadata contains information about location of each chunk and its replicas. It must be known that Master Node itself is replicated otherwise it becomes single point of failure. In HDFS, the replicted master node is known as Secondary Name Node.

2.2.3 Client Library

When a client(algorithm) wants to access a file(data), it contacts Master and find chunk server that stores chunks and once this information is found, the clients are connected to chunk Server and for further access to data, it need not contact Master Node.

In general, MapReduce Environment takes care of

i. Partitioning of the input data
ii. Scheduling the program execution across set of machines
iii. Performing group by key step
iv. Handling node failures
v. Managing required inter-machine communication

As discussed above, whenever an input file comes, it is given to Job Tracker[2] that is shown in figure 2, whose job is to divide the input file into chunks and its replicas and to distribute them among multiple different servers and to maintain this location information into a repository known as Metadata.

Now, the master schedules the input file which is in the form of chunks to Task Tracker. Now Task Tracker internally invokes map() function that contains custom logic. The number of Task Tracker present are equal to number of chunks of an input file. These Task Trackers works in parallel. The map() function takes input as key-value pair and produces a list of intermediate key-value pair. On completion of a map task, its responsibility is to send the location and size of the file that contains intermediate key value pair to Master. In short, after completion of all map task, their intermediate file is forwarded to Master. Now it is the Master that forwards this file to appropriate Reducer. It must be remembered that in a single cluster, we have only one Reducer.

ISSN 2348 – 9928
Doi:01.0401/ijaict.2015.04.02 Published on 05(12) 2018

Now it's the responsibility of Reducer to combine or group all intermediate files of Mapper(Technically known as Task Tracker) into one unit and stores into local file system and to perform computation that is now reduce() function is called that contains custom logic which produces final results.

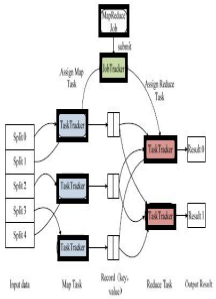

Fig 2: Map Reduce Architecture

To increase the efficiency of Reducer, an intermediate step is introduced that is the output of the Mapper, now is given to Combiner instead of Reducer. The combiner then invokes a combine () function that contains the custom logic. The output of the combiner is then forwarded to Reducer. Now let us understand how the MapReduce framework which handles the problem of node failures. It is the responsibility of Master to periodically ping the Mappers to find whether they are alive or not. And this happens by making Mappers to send a heartbeat signal every three seconds. But, in some case if Mapper does not give heartbeat signal till 30 seconds, then the Job Tracker comes to conclusion that Mapper is dead or working slowly and takes a decision to hand this task to nearest Mapper. But, if Mapper fails, the work done by Mapper, whether it is either started or half completed or full completed, is not taken into consideration and a nearest Mapper is selected and the same task is given from starting. But, if this happens in case of Reducer, the completed tasks are not reseted, but the uncompleted tasks are reseted and are given to the another Reducer. But, if Master fails then it is the secondary master that takes the responsibility. But, if both fails, then entire map reduce task is aborted and the client is notified.

Now let us understand the concept of MapReduce with the help of an example as shown in figure 3 where we have a 1TB file that consists of million words and we have to find a specific word and its count.

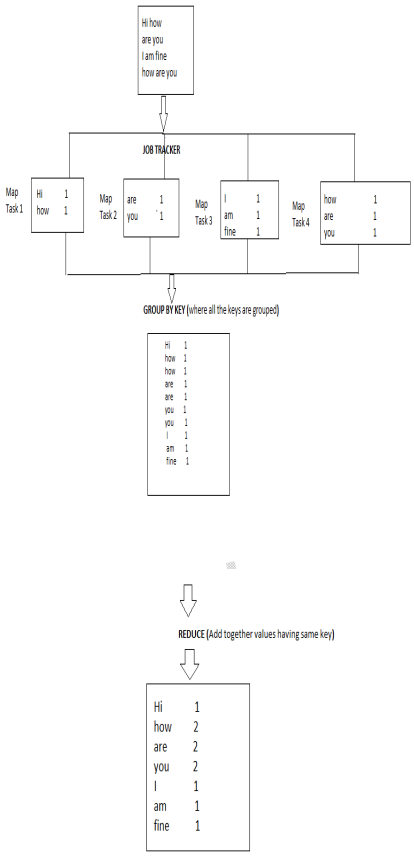

Fig 3: Example of Map reduce

Advantages:

The following are the advantages of MapReduce over the predecessors:

1. Load Balancing
2. Efficient and Reliable distributed data storage
3. Easy to use as most of the work is done by daemons and the responsibility of the programmer is just to write code for map function and reduce function
4. Its flexible as MapReduce can work with any type of data and any type of storage layers like Big Table and so on.
5. Its highly fault tolerant as it works inspite of average 1.2 failures per job analysis.

Limitations:

1. Doesn't support languages like SQL and any query optimization techniques.

ISSN 2348 – 9928
Doi:01.0401/ijaict.2015.04.02 Published on 05(12) 2018

2. Since MapReduce is scheme free and index free, it has to parse each input and convert to objects for data processing that degrades performance.

III. CONCLUSION

The future versions of Map Reduce should support SQL as it is basic language to work with DBMS. Moreover the concept of parsing input to be given to some daemons so that performance can be increased.

References

[1] MapReduce Online By Tyson Condie, Neil Conway, Peter Alvaro, Joseph M. Hellerstein from UC Berkeley and Khaled Elmeleegy, Russell Sears from Yahoo! Research

[2] Use of MapReduce for Data Mining and Data Optimization on a Web Portal by Christopher A. Moturi and Silas K. Maiyo ,School of Computing and Informatics , University of Nairobi ,Kenya.