International Journal of Engineering Technology Science and Research
IJETSR
www.ijetsr.com
ISSN 2394 – 3386
Volume 5, Issue 3
March 2018

MapReduce: An Efficient and Simplified Technique for Big Data Processing

Saurabh Sindhu[1] and Divya Sindhu[2]
[1]Ph. D. scholar, Deptt. of Computer Sciences and Engineering, JJT University, Jhunjhunu , Rajasthan
[2]Ph. D. scholar, Deptt. of Computer Sciences and Engineering, JJT University, Jhunjhunu , Rajasthan

Abstract*:*
There is a rapid growth of data, data computations and analysis due to the development of web based applications and mobile computer technology in the recent years. Various organizations are facing a challenge to deal with this large scale of data which highly supports in decision making. This Big Data is used to describe a massive volume of both structured and unstructured data from multiple autonomous sources which is so large that it's difficult to process using traditional database and software techniques. The traditional relational DBMS's are also unable to handle this Big Data. Therefore, there is a need of an intelligent data analysis tool that would be helpful to satisfy the need to analyse a large amount of data. MapReduce is a popular programming model suitable for Big Data analysis in distributed and parallel computing. The high scalability of MapReduce is one of the reasons for adapting this model. Hadoop is an open source, distributed programming framework which enables the storage and processing of large data sets. In this paper, MapReduce and Hadoop technologies are discussed for the analytical processing of Big Data.

Keywords: MapReduce, Hadoop Distributed File System, Key Value pairs, Hadoop, Big Data

1. INTRODUCTION

In the present days, huge amount of data is being generated continuously and handling of this huge data is a topic of major concern for researchers. We are moving from the Petabyte to exabytes and now in zettabytes age with this enormous amount of data. With this trend, there exists a greater demand for new data storage and analysis methods. Many applications like data mining, Image processing, data analytic etc are required for processing of this massive amount of data[1]. MapReduce is one such programming model and an associated implementation for processing and generating large data sets with a parallel, distributed algorithm on a cluster. The real power of MapReduce is it's capability to divide and conquer. Take a large problem and break into smaller, more manageable chunks, operate the chunks independently, and then pull them all together at the end.

MapReduce is a processing paradigm of executing data with partitioning and aggregation of intermediate results. It works to process data in parallel in which splitting of data, distribution, synchronization and fault tolerance are handled automatically by the framework. MapReduce framework is famous for large scale data processing and analysis of voluminous datasets in clusters of machines.

2. IMPORTANCE OF BIG DATA

Big Data can be defined as large volumes of data which is either structured or unstructured and generated at high speeds globally by various new technological devices.

Big Data includes the data that is being generated every second by sensors, mobiles, and consumer-driven data from social networks. Big Data is evolving from various facets within organizations legal, sales, marketing, procurement, finance, and human resources departments etc.

Big Data Structures**:**

- Big Table – consisting of relational tables.
- Big Text – comprising of text in the form of structured, semi-structured data, natural language, and semantic data[2].

- Big Metadata – collects and stores the data about data stored in big data.
- Big Graphs – Graphs include connections between objects, their semantic discovery, and the degree of separation, linguistic analytics, and subject predicate.

3. MAPREDUCE

MapReduce is a programming model and an associated implementation for processing and generating large data sets with a parallel, distributed algorithm on a cluster. It is designed for processing extremely large volumes of data in parallel mode by splitting the job into various independent tasks.

A MapReduce program in general is composed of a Map() procedure and a Reduce() procedure. The job of Map() is to perform filtering and sorting operations such as, sorting students by first name into queues, by generating one queue for each name and the Reduce() performs a summary/aggregate operations like counting the number of students in each queue and thus yielding the name counts. The "MapReduce System" well known as MapReduce "framework" or “architecture” demonstrates the processing with the distributed servers, running the various tasks in parallel, managing all communications and data transfers between the various parts of the system, and providing appropriate mechanisms to deal with redundant data and fault tolerance.

MapReduce is a framework for processing voluminous data splitted and distributed across huge datasets using a large number of computers (nodes). The group of nodes are collectively treated as a cluster, if all nodes are with similar hardware configurations working on the same local network or else the nodes are treated as a grid, if they are geographically shared and distributed with varying hardware specifications. Processing may occur on the data that is stored either in system log files (unstructured) or in a database (structured). MapReduce takes advantage of locality of data, to minimise the data transfer distance.

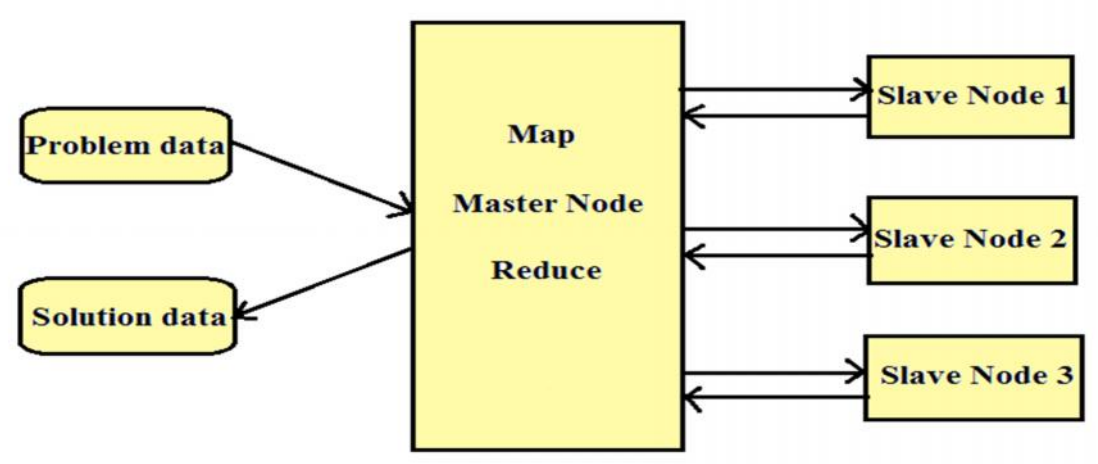

Fig. 1. MapReduce Task

The following are the two phases of MapReduce model:

Map Phase: In the map phase, the input is taken by the master node and is divided into smaller sub-tasks, and is distributed to worker nodes. A worker node repeats this again leading to a multi-level tree structure. The worker node processes the smaller task only, and passes the intermediated result back to its master node.

Reduce Phase: During the reduce phase, all the intermediated outputs of all the sub-tasks generated by various worker nodes are collected by the master node and are combined in some way to form the final output to generate the solution to the problem.

a) **Input reader**: The input reader splits the input file into appropriate sizes (in practice typically 64 MB to 512 MB as per HDFS) and one split is assigned to one Map function by MapReduce framework. The input reader takes input from stable storage (typically as in our case Hadoop distributed file system) and generates the output as key/value pairs.

b) **Map function**: Each Map function takes a series of key/value pairs generated by the input reader, processes each, and in turn produces zero or more output key/value pairs[3]. The input and output types of the map can be and often are different from each other.

c) **Partition function**: Each Map function output is assigned to a particular reducer by the application partition function for sharing purposes. The partition function is given as input the key and the number of reducers and it return the index of desired reduce.

d) **Comparison function**: The input for every Reduce is fetched from the machine where the Map is run and sorted using comparison function.

e) Reduce function: The frame work calls the applications Reduce function for each unique key in the sorted order. It also iterates through the values that are associated with that key and produce zero or more outputs[4].

f) Output writer: It writes the output of the Reduce function to stable storage, usually a Hadoop distributed file system.

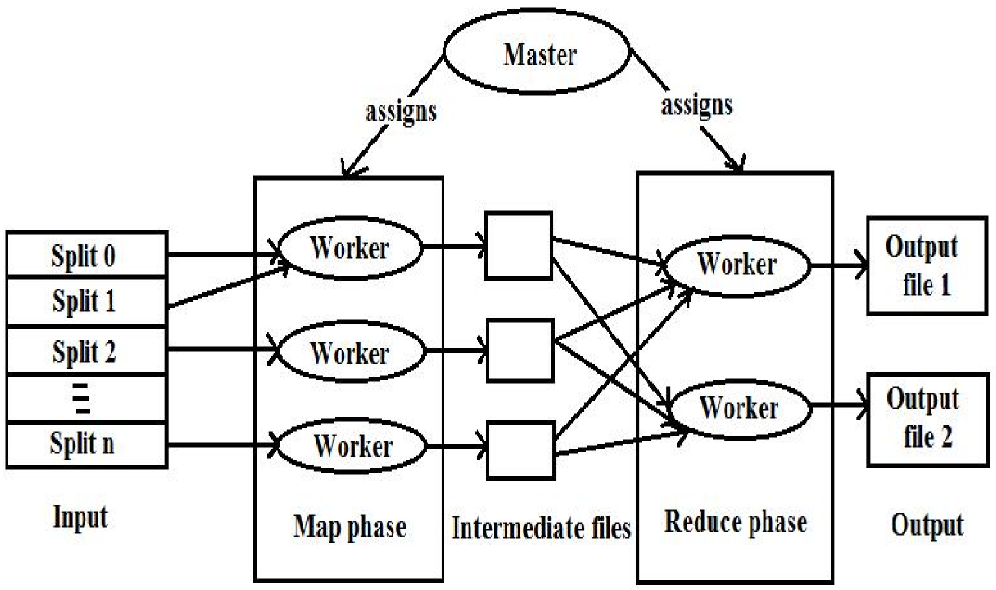

Fig. 2. MapReduce Workflow

Users can implement their own processing logic by specifying a customized map() and reduce() function. The map () function takes an input key/value pair and produces a list of intermediate key/value pairs. The Map Reduce runtime system groups together all intermediate pairs based on the intermediate keys and passes them to reduce() function for producing the final results.

Map

(in_key, in_value) --->list(out_key,intermediate_value)

Reduce

(out_key,list(intermediate_value)) -- ->list(out_value)

The signatures of map() and reduce() are as follows :

map (k1,v1) ! list(k2,v2)and reduce (k2,list(v2)) !

list(v2)

A Map Reduce cluster employs a master-slave architecture where one master node manages a number of slave nodes. In the Hadoop, the master node is called Job Tracker and the slave node is called Task Tracker as shown in the figure 4. Hadoop launches a Map Reduce job by first splitting the input dataset into even-sized data blocks. Each data block is then scheduled to one Task Tracker node and is processed by a map task. The Task Tracker node notifies the Job Tracker when it is idle. The scheduler then assigns new tasks to it. The scheduler takes data locality into account when it disseminates data blocks.

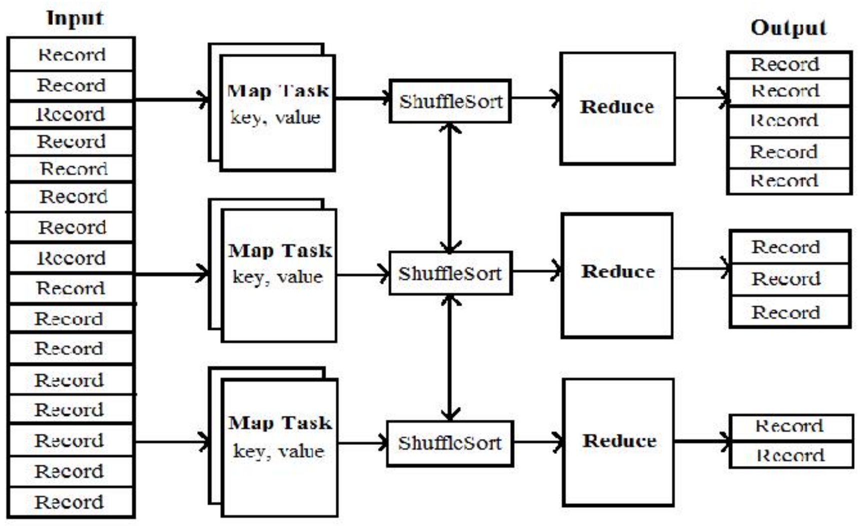

Fig. 3. MapReduce Architecture

The schedular always tries to assign a local data block to a Task Tracker. If the attempt fails, the scheduler will assign a random data block to the Task Tracker. When map() functions completes, the runtime system groups all intermediate pairs and launches a set of reduce tasks to produce the final results.

A brief description of MapReduce components and it's working is provided below to understand it's functioning with more clarity :

A. Map Reduce Components

1. Name Node –manages HDFS metadata, (since with files directly)
2. Data Node – stores blocks of HDFS
3. Job Tracker –schedules, allocates and monitors job execution on slaves –Task Trackers
4. Task Tracker –runs Map Reduce operations

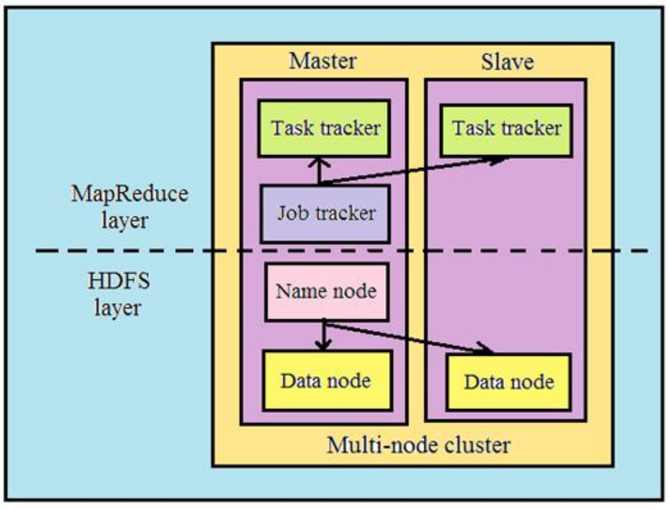

Fig. 4. MapReduce Components

B. MapReduce Working

We implement the Mapper and Reducer interfaces to provide the map and reduce methods and these form the core of the job.

1) Mapper

Mapper maps input key/value pairs to a set of intermediate key/value pairs. Maps are the individual tasks that transform input records into intermediate records. The transformed intermediate records do not need to be of the same type as the input records. A given input pair may map to zero or many output pairs.

2) Reducer

Reducer reduces a set of intermediate values which share a key to a smaller set of values. Reducer has 3 primary phases: shuffle, sort and reduce.

2.1) Shuffle

Input to the Reducer is the sorted output of the mappers. In this phase the framework fetches the relevant partition of the output of all the mappers.

2.2) Sort

The frame work doesn't group Reducer deal with inputs by keys (since different mappers may have given the same key as output) in this stage. The shuffle and sort phases occur simultaneously; while map outputs are being fetched as they are merged.

2.3) Secondary Sort

If equivalence rules for grouping the intermediate keys are required to be different from those for grouping keys before reduction, then one may specify a Comparator(Secondary Sort).

2.4) Reduce

In this phase the reduce method is called for each <key, (list of values)> pair in the grouped inputs. The output of the reduce task is typically written to the File System via Output Collector . Applications can use the Reporter to report progress, set application-level status messages and update Counters, or just indicate that they are alive. The output of the Reducer is not sorted.

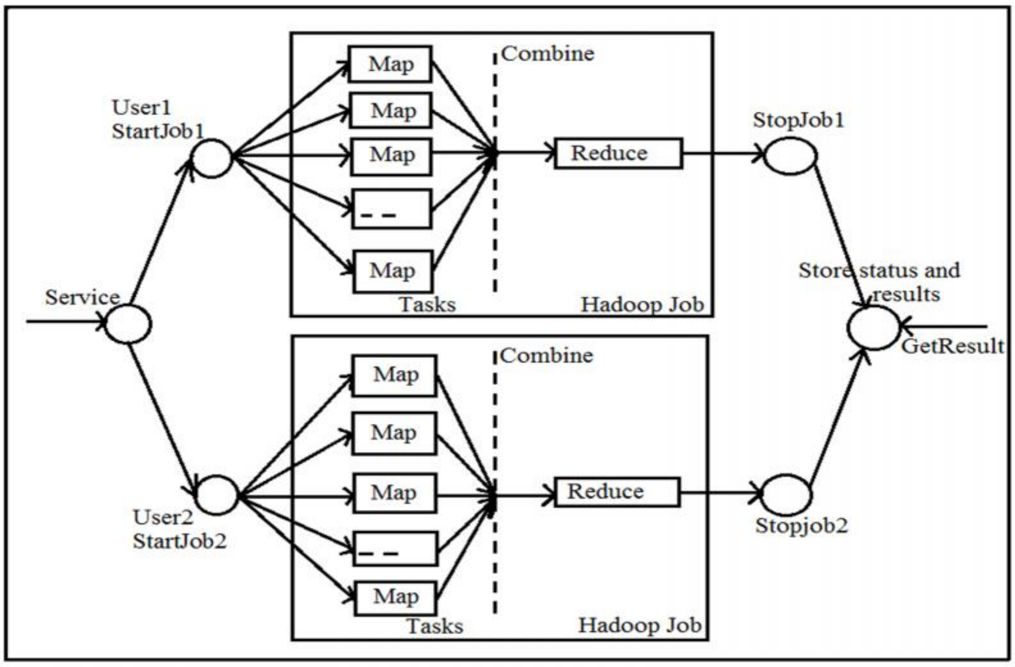

Fig. 5. MapReduce Working

a) Partitioner

Partitioner partitions the key space. Partitioner controls the partitioning of the keys of the intermediate map -outputs. The key (or a subset of the key) is used to derive the partition, typically by a hash function. The total number of partitions is the same as the number of reduce tasks for the job. Hence this controls which of the m reduce tasks, the intermediate key (and hence the record) is sent to for reduction. Hash Partitioner is the default Partitioner.

b) Reporter

Reporter is a facility for MapReduce applications to report progress, set application-level status messages and update Counters. Mapper and Reducer implementations can use the Reporter to report progress or just indicate that they are alive. In scenarios where the application takes a significant amount of time to process individual key/value pairs, this is crucial since the framework might assume that the task has timed-out and kill that task. Applications can also update Counters using the Reporter.

c) Output Collector

Output Collector is a generalization of the facility provided by the MapReduce framework to collect data output by the Mapper or the Reducer (either the intermediate outputs or the output of the job). Hadoop MapReduce has a bundle of library of generally useful mappers, reducers, and partitioners.

The above diagram shows the logical flow of a MapReduce programming model

1. **Input:** This is the input data / file to be processed.

2. Split: Hadoop splits the incoming data into smaller pieces called "splits"

3. **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.

4. **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.

5. **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.

6. **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.

7. **Output:** Finally the output of reduce step is written to a file in HDFS.

4. CONCLUSION

Large scale data processing is a difficult task, managing hundreds or thousands of processors and managing parallelization and distributed environments makes is more difficult. Map Reduce provides solution to the above mentioned issues, as is supports distributed and parallel I/O scheduling, it is fault tolerant and supports scalability and it has inbuilt processes for status and monitoring of heterogeneous and large datasets as in Big Data [5]. Hadoop with its distributed file system & programming framework based on concept of mapped reduction, is a powerful tool to manage large data sets. With its map-reduce programming paradigms, overall architecture fault- tolerance techniques and distributed processing, Hadoop offers a complete infrastructure to handle Big Data. 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. The primary focus of this paper is to highlight some Map/Reduce implementation which worked well to accomplish a specific purpose. The aim of this paper was giving an overview of the Map Reduce concepts in big data analytics. Map Reduce was developed by Google to handle big data analysis which is unstructured data such as web related data. But, it also can be used for structured data. We have discussed a number of Map Reduce models, still researchers can develop a more efficient Map Reduce with improved functionalities.

References

[1] V. Patil, V.B. Nikam, "Study of Mining Algorithm in cloud computing using MapReduce Framework", Journal of Engineering, Computers & Applied Sciences (JEC&AS) Vol.2, No.7, July 2013.

[2] https://en.wikipedia.org/wiki/MapReduce

[3] D. Usha, A.P.S. AslinJenil, " A Survey of Big Data Processing in Perspective of Hadoop and Mapreduce**",** International Journal of Current Engineering and Technology, Vol.4, No.2,April 2014.

[4] S. Ghemawat et al ."The Google File System." ACM SIGOPS Operating Systems Review, 37(5):29–43, 2003.

[5] http://hadoop.apache.org, 2010