International Journal of Computer Architecture and Mobility (ISSN 2319-9229) Volume 4-Issue 3, March 2016

Introduction To MapRedcuce

Mr. Shubham Sharma
Shubhamsharma1318@gmail.com

Abstract **- Map Reduce is a programming model for data-parallel programs originally intended for data centers. Map Re-duce simplifies parallel programming, hiding synchronization and task management. Big Data has come up with aureate haste and a clef enabler for the social business, Big Data gifts an opportunity to create extraordinary business advantage and better service delivery. Big Data is bringing a positive change in the decision making process of various business organizations. With the several offerings Big Data has come up with several issues and challenges which are related to the Big Data Management, Big Data processing and Big Data analysis. Big Data is having challenges related to volume, velocity and variety. Big Data has 3Vs Volume means large amount of data, Velocity means data arrives at high speed, Variety means data comes from heterogeneous resources. In Big Data definition, Big means a dataset which makes data concept to grow so much that it becomes difficult to manage it by using existing data management concepts and tools. Map Reduce is playing a very significant role in processing of Big Data. This paper includes a brief about Big Data, Map Reduce and its related issues, emphasizes on role of Map Reduce in Big Data processing. Map Reduce is elastic scalable, efficient and fault tolerant for analyzing a large set of data, highlights the features of Map Reduce in comparison of other design model which makes it popular tool for processing large scale data. Analysis of performance factors of Map Reduce shows that elimination of their inverse effect by optimization improves the performance of Map Reduce.**

Keywords**— Big Data , Map Reduce , Parallel programming ,Velocity,fault tolerant.**

I. INTRODUCTION

Mapreduce is a processing technique and a program model for distributed computing based on java. The mapreduce algorithm contains two important tasks, namely map and reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name mapreduce implies, the reduce task is always performed after the map job.The major advantage of mapreduce is that it is easy to scale data processing over multiple computing nodes. Under the mapreduce model, the data processing primitives are called mappers and reducers. Decomposing a data processing application into mappers and reducers is sometimes nontrivial. But, once we write an application in the mapreduce form, scaling the application to run over hundreds, thousands, or even tens of thousands of machines in a cluster is merely a configuration change. This simple scalability is what has attracted many programmers to use the mapreduce model.

2. ANALYSIS OF BIG DATA AND ROLE OF MAP REDUCE

In this part we will analysis the basics of big data and mapreduce

2.1 Big Data:

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. Big data "size" is a constantly moving target, as of 2012 ranging from a few dozen terabytes to many petabytes of data. Big data requires a set of techniques and technologies with new forms of integration to reveal insights from datasets that are diverse, complex, and of a massive scale. Big data is a buzzword, or catch-phrase, meaning a massive volume of both structured and unstructured data that is so large it is difficult to process using traditional database and software techniques. In most enterprise scenarios the volume of data is too big or it moves too fast or it exceeds current processing capacity.

2.1.1. History:-

The story of how data became big starts many years before the current buzz around big data. Already seventy years ago we encounter the first attempts to quantify the growth rate in the *volume of data* or what has popularly been known as the “information explosion” (a term first used in 1941, according to the *Oxford English Dictionary*). The following are the major milestones in the history of sizing data volumes plus other “firsts” in the evolution of the idea of “big data” and observations pertaining to data or information explosion.

2.1.2 Basic Model of big data:-

Big data can be described by the following

characteristics:

Volume:-The quantity of generated and stored data. The size of the data determines the value and potential insight- and whether it can actually be considered big data or not.

Variety:-The type and nature of the data. This helps people who analyze it to effectively use the resulting insight.

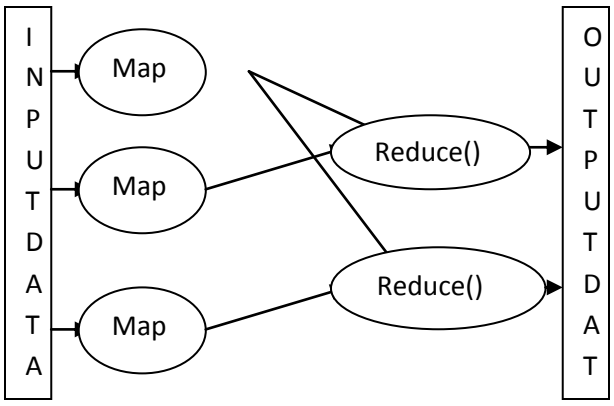
Velocity:-In this context, the speed at which the data is generated and processed to meet the demands and challenges that lie in the path of growth and development.

Variability:-Inconsistency of the data set can hamper processes to handle and manage it.

Veracity:-The quality of captured data can vary greatly, affecting accurate analysis.

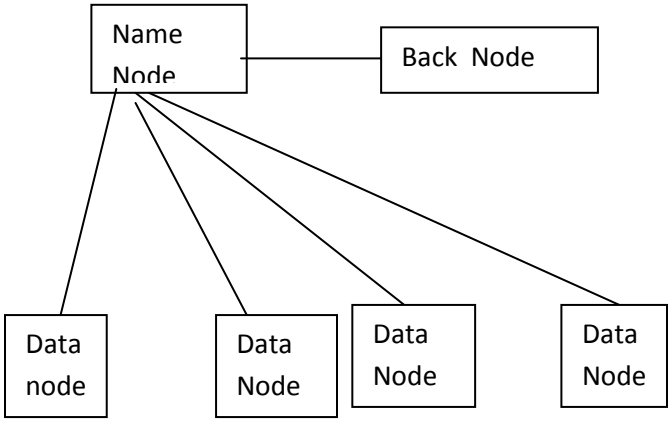
2.1.2 Basic Model of Map Reduce:-

MapReduce is a programming model and software framework first developed by Google (Google's MapReduce paper submitted in 2004).It is organized as a "map" function which transform a piece of data into some number of key/value pairs. Each of these elements will then be sorted by their key and reach to the same node, where a "reduce"
function is use to merge the values (of the same key) into a single result.



2.1.3. Basic Model of HDFC:-

The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware. It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets. HDFS relaxes a few POSIX requirements to enable streaming access to file system data.



3. PROGRAMMING MODEL

The computation takes a set of *input* key/value pairs, and produces a set of *output* key/value pairs. The user of the MapReduce library expresses the computation as two functions: map and reduce.Map, written by the user, takes an input pair and produces a set of *intermediate* key/value pairs. The MapReduce library groups together all intermediate values associated with the same intermediate key *I* and passes them to the reduce function.The reduce function, also written by the user, accepts an intermediate key *I* and a set of values for that key. It merges these values together to form a possibly smaller set of values. Typically just zero or one output value is produced per reduce invocation. The intermediate values are supplied to the user's reduce function via an iterator. This allows us to handle lists of values that are too large to fit in memory.

3.1 Example

Consider the problem of counting the number of occurrences of each word in a large collection of documents. The user would write code similar to the following pseudocode.

```
map(String key, String value):
// key: document name
// value: document contents
or each word w in value:
EmitIntermediate(w, "1");
reduce(String key, Iterator values):
// key: a word
// values: a list of counts
int result = 0;
for each v in values:
```

```
result += ParseInt(v);
Emit(AsString(result));
```

The map function emits each word plus an associated count of occurrences (just 1 in this simple example). The reduce function sums together all counts emitted for a particular word.In addition, the user writes code to fill in a *mapreduce specification* object with the names of the input and output files and optional tuning parameters. The user then invokes the M*apReduce* function, passing it to the specification object. The user's code is linked together with the MapReduce library (implemented in C++). Our original MapReduce paper contains the full program text for this example [8].More than ten thousand distinct programs have been implemented using MapReduce at Google, including algorithms for large-scale graph processing, text processing, data mining, machine learning, statistical machine translation, and many other areas. More discussion of specific applications of MapReduce can be found elsewhere.

3.2 Types

Even though the previous pseudocode is written in terms of string inputs and outputs, conceptually the map and reduce functions supplied by the user have associated types.

Map	(k1,v1) →	list(k2,v2)
Reduce	(k2,list(v2)) →	list(v2)

That is, the input keys and values are drawn from a different domain than the output keys and values. Furthermore, the intermediate keys and values are from the same domain as the output keys and values.

4. IMPLEMENTATION

Many different implementations of the Map Reduce interface are possible. The right choice depends on the environment. For example, one implementation may be suitable for a small shared-memory machine, another for a large NUMA multi-processor, and yet another for an even larger collection of networked machines.

Below is the implementation :

```
Map(input_record) {

Emit(k1,v1)
Emit(k2,v2)
…
…
}
These Map(input_record) function takes
All of your record as input and then convert
Them into key,value pair.
Reduce(key,values){
Aggregate=initialize()
While(values.has_next){
Aggregate=merge(values.next)
}
Collect(key,aggregate)
}
```

These reduce function then split the whole part and sort the records and merge them into a single result.

5. PERFORMANCE

In this section we measure the performance of MapReduce on two computations running on a large cluster of machines. One computation searches through approximately one terabyte of data looking for a particular pattern.The other computation sorts approximately one terabyte of data.These two programs are representative of a large subset of the real programs written by users of MapReduce .one class of programs shuf_es data from one representation to another, and another class extracts a small amount of interesting data from a large data set.

ACKNOWLEDGMENT

This research paper is made possible through the help and support from everyone, including: parentsteachers,family, friends, and in essence, all sentient beings.Especially, please allow me to dedicate my acknowledgment of gratitude toward the following significant advisors and contributors:

First and foremost, I would like to thank Ms. Shaifali shrivastava for his most support and encouragement. She kindly read my paper and offered invaluable detailed advices on grammar, organization, and the theme of the paper . She had delivered content for my research paper .

Finally, I sincerely thank to my parents, family, and friends, who provide the advice and financial support. The product of this research paper would not be possible without all of them.

REFERENCES:

[1]. Hadoop: Open source implementation of MapReduce. http://lucene. apache.org/hadoop/.

[2].The Phoenix system for MapReduce programming. http:// csl.stanford. edu/~christos/sw/phoenix/.

[3].BigData: International Journal of Big Data Intelligence,http://www.inderscience.com/jhome.php?jcode=ijbdi

[4].MapReduce: Google Research Publication: http://research.google.com/archive/mapreduce .htm.