International Journal of Engineering Research and Development
e-ISSN: 2278-067X, **p**-ISSN: 2278-800X, www.ijerd.com
Volume 6, Issue 10 (April 2013), PP. 60-65

Big Data Analytics: Insights and Innovations

Ms. Manisha Saini[1], Ms. Pooja Taneja[2], Ms. Pinki Sethi[3]
Assistant Professor, Computer Science, Dronacharya College of Engineering,Gurgaon

Abstract:- Big data is a collection of data sets so large and complex that it becomes difficult to process using on-hand database management tools or traditional data processing applications .The paper is all about big data issues, and ideas and thoughts in order to advance big data challenge, which has been considered as one of the most exciting opportunities of present times.

Keyword:- Big Data, Analytics, Insights and Innovations.

I. INTRODUCTION

Today Is the World of Our Data-Driven World. When we talk about Science, There is a lot of data like Data bases from astronomy, genomics, environmental data, transportation data.in Humanities and Social Sciences data is store in th form of Scanned books, historical documents, social interactions data, new technology like GPS.

Business & Commerce data is like corporate sales, stock market transactions, census, and airline traffic. Regarding Entertainment, there is a lot of big data in form of Internet images, Hollywood movies, and MP3 files. A lot of MRI & CT scans and lot of patient records are there in world. So today business need is to analysis this big data.

Big-data computing is perhaps the biggest innovation in computing in the last decade. We have only begun to see its potential to collect, organize, and process data in all walks of life [1]

II. WHAT IS BIG DATA

Advances in digital sensors, communications, computation, and storage have created huge collections of data, capturing information of value to business, science, government, and society. For example, search engine companies such as Google, Yahoo!, and Microsoft have created an entirely new business by capturing the information freely available on the World Wide Web and providing it to people in useful ways. These companies collect trillions of bytes of data every day and continually add new services such as satellite images, driving directions and image retrieval. The societal benefits of these services are immeasurable, having transformed how people find and make use of information on a daily basis.

A. Every day, we create 2.5 quintillion bytes of data.
B. 90% of the data in the world today has been created in the last two years alone, this data comes from everywhere

2.1 Volume, Velocity and Variety of big data.
2.1.1 Volume: Enterprises are ever-growing data of all types, easily (peta bytes of information).
A. Turn 12 terabytes of Tweets created each day into improved product sentiment analysis.
B. Convert 350 billion annual meter readings to better predict power consumption.

2.1.2 Velocity: For time-sensitive processes such as catching fraud, big data must be used as it streams into your enterprise in order to maximize its value.
A. Scrutinize 5 million trade events created each day to identify potential fraud.
B. Analyze 500 million daily call detail records in real-time to predict customer churn faster.

2.1.3 Variety: Big data is any type of data - structured and unstructured data such as text, sensor data, audio, video, click streams, log files and more. New insights are found when analyzing these data types together.
A. Monitor 100’s of live video feeds from surveillance cameras to target points of interest.
B. Exploit the 80% data growth in images, video and documents to improve customer satisfaction.

2.3 What Is the Capacity of Big Data?
A. Google Processes 20 Pb A Day (2008)

B. Wayback Machine Has 3 Pb + 100 Tb/Month (3/2009)
C. Facebook Has 2.5 Pb Of User Data + 15 Tb/Day (4/2009)
D. Ebay Has 6.5 Pb Of User Data + 50 Tb/Day (5/2009)
E. Cern's Large Hydron Collider (Lhc) Generates 15 Pb A Year

III. BIG-DATA TECHNOLOGY

The rising importance of big-data computing stems from advances in many different technologies:
3.1 Sensors: Digital data are being generated by many different sources, including digital imagers (telescopes, video cameras, MRI machines), chemical and biological sensors (microarrays, environmental monitors), and even the millions of individuals and organizations generating web pages.

3.2 Computer networks: Data from the many different sources can be collected into massive data sets via localized sensor networks, as well as the Internet.

3.3 Data storage: Advances in magnetic disk technology have dramatically decreased the cost of storing data. For example, a one-terabyte disk drive, holding one trillion bytes of data, costs around $100. As a reference, it is estimated that if all of the text in all of the books in the Library of Congress could be converted to digital form, it would add up to only around 20 terabytes.

3.4 Cluster computer systems: A new form of computer systems, consisting of thousands of "nodes," each having several processors and disks, connected by high-speed local-area networks, has become the chosen hardware configuration for data-intensive computing systems. These clusters provide both the storage capacity for large data sets, and the computing power to organize the data, to analyze it, and to respond to queries about the data from remote users. Compared with traditional high-performance computing (e.g., supercomputers), where the focus is on maximizing the raw computing power of a system, cluster computers are designed to maximize the reliability and efficiency with which they can manage and analyze very large data sets. The "trick" is in the software algorithms – cluster computer systems are composed of huge numbers of cheap commodity hardware parts, with scalability, reliability, and programmability achieved by new software paradigms.

3.5 Cloud computing facilities: The rise of large data centers and cluster computers has created a new business model, where businesses and individuals can rent storage and computing capacity, rather than making the large capital investments needed to construct and provision large-scale computer installations. For example, Amazon Web Services (AWS) provides both network-accessible storage priced by the gigabyte-month and computing cycles priced by the CPU-hour. Just as few organizations operate their own power plants, we can foresee an era where data storage and computing become utilities that are ubiquitously available.

3.6 Data analysis algorithms: The enormous volumes of data require automated or semi-automated analysis – techniques to detect patterns, identify anomalies, and extract knowledge. Again, the "trick" is in the software algorithms - new forms of computation, combining statistical analysis, optimization, and artificial intelligence, are able to construct statistical models from large collections of data and to infer how the system should respond to new data. For example, Netflix uses machine learning in its recommendation system, predicting the interests of a customer by comparing her movie viewing history to a statistical model generated from the collective viewing habits of millions of other customers.

IV. ANALYSIS OF BIG DATA USING HADOOP

4.1 Map reduce

Map reduce is a parallel programming framework that uses divide and conquer method to break down a complex big data problems into small task and process them in parallel

4.2 Hadoop

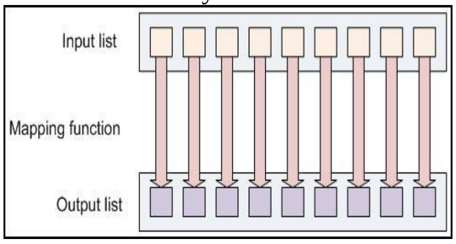
A complete technology stack that implements the concept of Map reduces. It is a standard based open source software and distributed by Apache. .In an age of big data, Hadoop is becoming a key technology that can deliver real value to its users; but it is not a panacea. There are security issues, and it requires specific skills to set up and maintain an environment using Hadoop

4.3 MAPREDUCE: HOW IT WORKS

Map and Reduce terms are taken from list processing languages such as LISP, Scheme, or ML. Map Reduce programs work on large volumes of data. They are designed to divide the load between several machines working in parallel. The Map Reduce programs process any data in two phases.

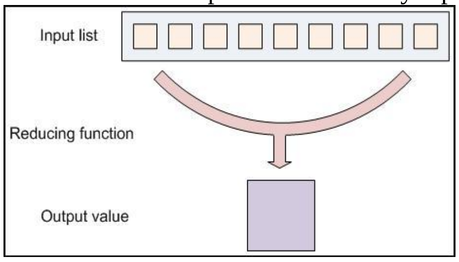
4.3.1 MAPPING

Mapping is the first phase of any MapReduce program. We supply a mapper function that gets a list of data elements from the input and transforms each element individually to output data. Several mapper instances run in parallel to process large amount of data concurrently.



4.3.2 Reducing

Reducing lets we combine mapped output together into results. A reducer function defined by us, gets multiple reducers' output as the input value. It produces a single combined output from the mapped values.Reducing is often used to generate analytics of the reduced data to provide a summary report of the reduced data set.



4.4 Example Application: Word Count

A simple Map Reduce program can be written to determine how many times different words appear in a set of files.
For example, if we had the files:
paragraph1.txt, paragraph2.txt etc….having sentences like all text and no practical make it boring. Is text better or practical? We would expect the output to be:

```
All 1
text  2
and 1
no 1
practical 2
make 1
it 1
boring 1
is 1
better 1
```

The Pseudo code for such a MapReduce program will be:

```
 for each word in file-contents:
output (word, 1)
reducer (word, values):
 sum = 0  for each value in values:
 sum = sum + value
 output (word, sum)
```

Each Mapper instance receives a different input file made by splitting the large data it has to map. The Mapper function's outputs are key value pairs. This output data is forwarded to the reducers based upon the number of reducers configured and grouped by the keys. The Reducer then outputs the final count to a file. The Hadoop distribution comes with a sample MapReduce program that does the same. It can be found at src/examples/org/apache/hadoop /examples/WordCount.java.

```
public static class TokenizerMapper extends
            Mapper<Object, Text, Text, IntWritable> {
      private final static IntWritable one = new IntWritable(1);
      private Text word = new Text();
      public void map(Object key, Text value, Context context)
throws IOException, InterruptedException {  StringTokenizer itr = new StringTokenizer(value.toString());
            while (itr.hasMoreTokens()) {
                  word.set(itr.nextToken());
                  context.write(word, one);
            }
      }
}
public static class IntSumReducer extends
            Reducer<Text, IntWritable, Text, IntWritable> {  private IntWritable result = new IntWritable();
public void reduce(Text key, Iterable<IntWritable> values,
                  Context context) throws IOException, InterruptedException {
            int sum = 0;
            for (IntWritable val : values) {
            sum += val.get();     }
            result.set(sum);
            context.write(key, result);
      }
}
```

4.4.1 THE DRIVER

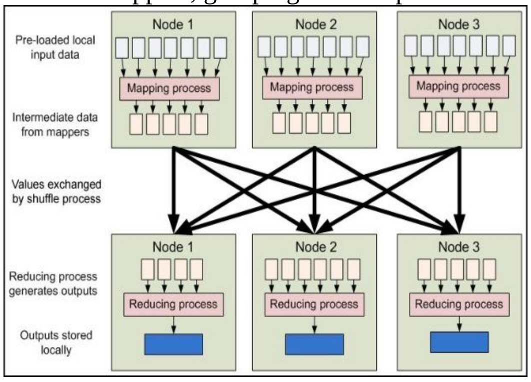
Hadoop calls the bootstrap method that manages the job a driver. It initializes the job, instructs Hadoop to execute our MapReduce program and controls where the output files are placed. A driver, from the example Java implementation, that comes with Hadoop is as below:

```
public static void main(String[] args) throws Exception
 {
      Configuration conf = new Configuration();
     String[] otherArgs = new GenericOptionsParser(conf, args)
                  .getRemainingArgs();
      if (otherArgs.length != 2) {
            System.err.println("Usage: wordcount <in> <out>");
            System.exit(2);   }
      Job job = new Job(conf, "word count");
      job.setJarByClass(WordCount.class);
      job.setMapperClass(TokenizerMapper.class);
      job.setCombinerClass(IntSumReducer.class);
      job.setReducerClass(IntSumReducer.class);
      job.setOutputKeyClass(Text.class);
      job.setOutputValueClass(IntWritable.class);
 FileInputFormat.addInputPath(job,new Path(otherArgs[0]));
   FileOutputFormat.setOutputPath(job,newPath(otherArgs[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

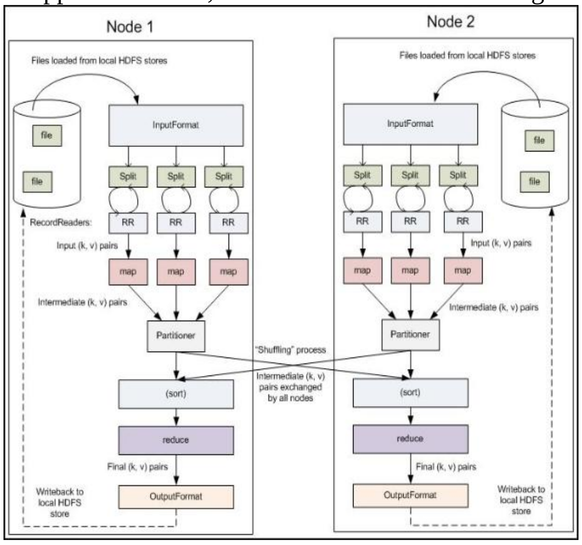
This method creates a configuration with generic options and then goes on to update the values. It uses this configuration and sets up a job to execute the word count program over the files in an input directory specified by inputPath argument. The reducers output is written into files in the directory identified by outputPath. The mapping and reducing functions are identified by the setMapperClass() and setReducerClass() methods. The data types produced by the reducer are identified by setOutputKeyClass() and setOutputValueClass(). These are the output types of the mapper as well as by default. The methods setMapOutputKeyClass(),setMapOutputValueClass() methods of the JobConf class is used to override this. The input types fed to the mapper are controlled by the InputFormat used. InputFormat API can be seen at [2] The call to waitForCompletion(boolean) will submit the job to MapReduce. This call will block until the job completes. If the job fails, it will throw an IOException.

4.4.2 MapReduce Detailed

At a higher level, Map Reduce inputs mainly come from input files loaded onto HDFS in the R processing cluster. This data is stored as chunks across the servers. A Map Reduce program runs on many or all of these nodes processing the local data. There is no data exchange between the Mapper tasks. At completion of Mapping, the mapped data is communicated across the nodes to the appropriate reducer instance. Reducer tasks are also not aware about the existence of other instances of reducers and do not communicate with each other. Hadoop manages all the data flow to the mappers, grouping their output and moving to the right reducer nodes.


Pre-loaded local input data
Intermediate data from mappers
Values exchanged by shuffle process
Reducing process generates outputs
Outputs stored locally
Node 1
Node 2
Node 3
Mapping process
Reducing process

This is the high level data flow of Hadoop MapReduce. It shows where the mapper and reducer components of the Word Count application fit in, and how the data flows through them.



This picture shows the pipeline with more of its internal mechanics.

1 **Files:** This is where the initial data for a MapReduce task is stored. It is typically HDFS. This data can be in any format and even binary. Typically the input file sizes are huge ranging into multiple gigabytes or more.

2 **InputFormat:** InputFormat is a class that defines how the input data is read and split. It defines the filter for selecting the input files, defines InputSplits that break a file into tasks and provides a RecordReader to read the data. Hadoop comes bundled with many predefined InputFormat, as shall be evident from the InputFormat API link posted earlier.

3 **InputSplits:** InputFormat class uses the InputSplit to define how the Input data will be broken up into smaller pieces. An InputSplit demarcates the chunk of data that will form a unit of work that will be fed to a single instance of MapReduce program. By default, it is the same as the block size of the HDFS. The value is controlled by setting the value of mapred.min.split.size parameter in hadoop-site.xml or at MapReduce runtime using the Configuration we create for it.

4 **RecordReader:** The RecordReader class defines how to load the data from its source and convert it into key, value pairs suitable for reading by the Mapper. The RecordReader instance is also defined by the InputFormat. The RecordReader is invoked repeatedly on the input until the entire InputSplit gets consumed. Each invocation of the RecordReader leads to another call to the map() method of the Mapper.

5 **Mapper:** The Mapper performs the user-defined algorithm on the input data and produces the summary as key value output. The individual mappers execute independently of each other and can't communicate among them. The map () method receives an OutputCollector and a Reporter instance to produce its output.

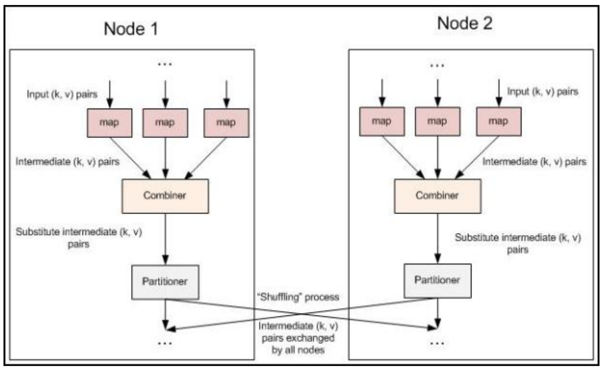
6 **Partition & Shuffle:** Exchanging the data is needed to get the mapped data to the right reducer node. The nodes begin it even before all map instances have finished execution. This is called shuffling. There is a space allocated to each reduce node to store this data, called partitions, which is used by reducers to store this data. A Practitioner class dictates the grouping and redirection of the mapped data to the appropriate reducer.

7 **Sort:** The set of partitioned keys on a single node are sorted by Hadoop before they are made available to the Reducer.

8 **Reduce:** Reducers combine multiple data inputs to typically deduce a summary from the input data. Just like a Mapper, it receives OutputCollector and Reporter instance to produce its output.

9 **OutputFormat:** The way output is written is governed by the OutputFormat. The instances of OutputFormat provided by Hadoop write to files on the local disk or in HDFS; they all inherit from a common FileOutputFormat.

10 **RecordWriter:** Like the RecordReader, the RecordWriter governs the writing of individual records to the files as directed by the OutputFormat. These output files in HDFS are the end result of the MapReduce process. They can be used as the input of another MapReduce process, processed by another program or just directly read by humans.


Node 1
Node 2
Input (k, v) pairs
map
map
map
Input (k, v) pairs
map
map
map
Intermediate (k, v) pairs
Intermediate (k, v) pairs
Combiner
Combiner
Substitute intermediate (k, v) pairs
Substitute intermediate (k, v) pairs
Partitioner
Partitioner
"Shuffling" process
Intermediate (k, v) pairs exchanged by all nodes

11 **Combiner:** Combiners can be optionally used to optimize bandwidth by running a pre-reduce algorithm on the mapped data on a node. This can be used to further reduce data before it goes into shuffling. Applying a Combiner to WordCount can on a node, pre group the Words parsed from the mappers and reduce the number of inputs going into the reducers. Combiners are actually reducers. We add a Combiner to a MapReduce task using the command:**conf.setCombinerClass(ReducerClassName.class);**

A.Chaining Jobs
B.MapReduce cannot solve every problem and more often a single MapReduce instance cannot fulfill the requirements. But we can execute a chain of MapReduce tasks to arrive at the desired output incrementally.
C.We can write multiple driver methods, each taking care of a Job. These jobs can be producing output at location used as input for the next job and so on. The problems that seem too complex for MapReduce can be solved by splitting the steps into small MapReduce tasks that can be run in chains. Hadoop support chained execution by providing the API of job1.addDependingJob (job2). This specifies that job1 cannot start until job2 has been successfully completed. Jobs can have multiple dependent jobs too. JobControl provides methods to track the execution of chained jobs. The job submission process does not begin until the run() method of the JobControl object is called.

V. CONCLUSION

This blog details the internal working of Hadoop's Map Reduce process. MapReduce's power lies in the way it uses multiple nodes to parallel process large amounts of data. It avoids communicating over the network using the node local processing as much as possible to optimize the speed further.

REFERENCES

[1]. http://www.cra.org/ccc/docs/init/Big_Data.pdf
[2]. http://hadoop.apache.org/docs/r1.1.1/api/org/apache/hadoop/mapred/InputFormat.html