Scientific Journal Impact Factor (SJIF) : 1.711
International Journal of Modern Trends in Engineering and Research
ISSN (ONLINE) : 2349 - 9745 ISSN (PRINT) : 2393 - 8161

Word Count Map-Reduce Job in Single Node Apache Hadoop Cluster

G Hima Bindu[1] , Y. Satish Kumar[2]
[1]PG Student, M. Tech, Department of Computer Science Engineering, Chaitanya Engineering College, Visakhapatnam, Andhra Pradesh., INDIA
[2]Assistant Professor, Department of Computer Science Engineering, Chaitanya Engineering College, Visakhapatnam, Andhra Pradesh., INDIA

Abstract - Big Data has become acronym for various clients and applications where huge amount of data should be stored and processed simultaneously. Applications like Yahoo, Facebook, and Twitter have huge data which has to be stored and retrieved as per client access. This huge data storage requires huge database leading to increase in physical storage and becomes complex for analysis required in business growth. This storage capacity can be reduced and distributed processing of huge data can be done using Apache Hadoop which uses Map-reduce algorithm and combines the repeating data so that entire data is stored in reduced format.

In this proposed project an experimental Word Count Map-Reduce Job is performed on a Single Node Apache Hadoop cluster and yields the prospective results as required by the user respectively. Hence also proves the functionality of Big Data both in storage and processing and found the results to be very encouraging.
Keywords **—** Hadoop, Map-reduce, Hadoop Distributed file system HDFS, HBase, Word Count

I. INTRODUCTION

Hadoop was created by Doug Cutting an employee at Yahoo and Michael J. Cafarella. It was originally developed to support distribution for the Nutch search engine project [12]. Hadoop was inspired by papers published by Google regarding its approach in handling an avalanche of data, and became a standard for storing, processing and analyzing hundreds of terabytes, and even petabytes of data. Hadoop‘s breakthrough advantages mean that businesses and organizations can now find value in data that was recently considered useless [11].

Hadoop enables a computing solution that is: Cost effective – Due to massive parallel computing approach by Hadoop, there is decrease in the cost per terabyte of storage. Fault tolerant – When a node is missed or a fault arises the system navigates work to another location of the data and continues processing. Flexible – Hadoop is schema-less, and can accept any type of data, structured or not, from any number of sources. Data from multiple sources can be joined and aggregated in arbitrary ways enabling deeper analyses than any one system can provide. Scalable – New nodes can be added as required and added without changing data formats [13].

This utility of Hadoop Framework can be well illustrated by Word Count, to get a flavor for how Map Reduce works, wherein Map Reduce application lies inside Hadoop Framework. Word Count is a simple

application that counts the number of occurrences of each word in an input set. Word Count reads the input text files and counts how often the typed in words occur. The input is text file and the output is text file, each line of which contains a word and the count of how often it occurred, separated by a space (or) tab (or) comma.

Internally each mapper takes a line as input and breaks it into words. It then emits a key/value pair of the word and 1. Each reducer sums the counts for each word and emits a single key/value with the word and sum. As an optimization, the reducer is also used as a combiner on the map outputs. This reduces the amount of data sent across the network by combining each word into a single record and helps in optimizing the size of the data to be transferred respectively.

This paper has been organized into following sections i.e. in section 2 Apache Hadoop Framework is described along with the huge data distribution process in Hadoop Distribution File System (HDFS). In section 3, the Installation and Configuration process of Hadoop in Ubuntu is reviewed. Section 4, describes how Word Count Job is processed in Map Reduce method optimally. A brief overview of the results which has been obtained by Map Reduce method are discussed and concluded in section 5.

II. APACHE HADOOP FRAMEWORK

A. Apache Hadoop

The framework Apache Hadoop is used for distributed processing of huge data sets known as ―Big Data‖ across clusters of computers using a simple programming model [2][1]. Hadoop permits an application to map, group, and reduce data across a distributed cloud of machines so that applications can process huge data [1]. It can scale up to large number of machines as required for the job; each machine will provide local computation and storage. Apache Hadoop software library itself detects and handles any failures at application layer [2].

B. Hadoop Distributed File System - HDFS

A distributed user-level filesystem HDFS Hadoop Distributed File System written in Java [15] stores huge files across machines in a large cluster. Hadoop DFS stores each file as a sequence of blocks, all blocks in a file except the last block are the same size typically 64 MB [14][15]. Blocks belonging to a file are replicated for fault tolerance. The block size and replication factor are configurable per file. An application can specify the number of replicas of a file. The replication factor can be specified at file creation time and can be changed later. Files in HDFS are "write once" and have strictly one writer at any time [15][16].

C. Architecture of Hadoop Distributed File System

HDFS comprises of interconnected clusters of nodes where files and directories reside. An HDFS cluster consists of a single node, known as a Name-Node that manages the file system namespace and regulates client access to files. In addition, Data-Nodes store data as blocks within files [27].

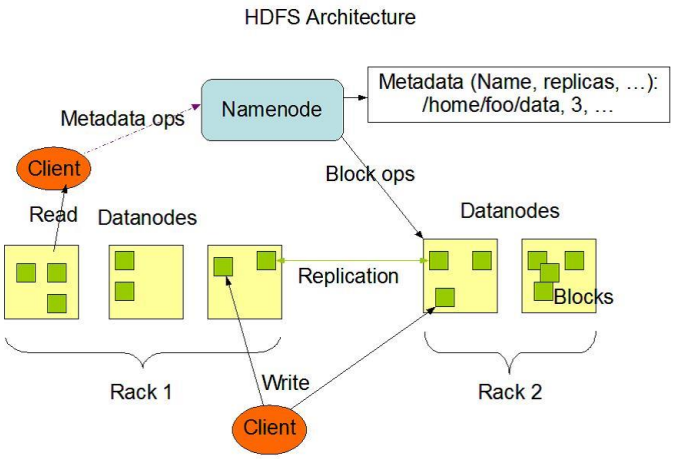

Fig.1 Block diagram of HDFS Architecture.

D. Name Node

The Name-Node executes file system namespace operations like opening, closing, and renaming files and directories. The Name-Node does not store HDFS data itself, but rather maintains a mapping between HDFS file name, a list of blocks in the file, and the Data Node on which those blocks are stored. The Name-Node makes all decisions regarding replication of blocks [15] [17].

E. Secondary Name Node

HDFS includes a Secondary Name-Node, there is a misconception that secondary Name-Node comes into action after Primary Name-Node (i.e. Name-Node) fails. The fact is Secondary Name-Node is continuously connected with Primary Name-Node and takes snapshots of Name-Node's memory structures. These snapshots can be used to recover the failed Name-Node and recent memory structure [12].

F. Data Node

A Data-Node stores data in the Hadoop File System. A functional file system has more than one Data-Node, with data replicated across them. On startup, a Data-Node connects to the Name-Node; spinning until that service comes up. It then responds to requests from the Name-Node for file system operations. Client applications can talk directly to a Data-Node, once the Name-Node has provided the location of the data [22].

G. Job-Tracker

Job-Tracker keeps track of which Map-Reduce jobs are executing, schedules individual Maps, Reduces or intermediate merging operations to specific machines, and monitors the success and failures of these individual Tasks, and works to complete the entire batch job. The Job-Tracker is a point of failure for the Hadoop Map-Reduce service. If it goes down, all running jobs are halted [20].

H. Task Tracker

A Task-Tracker is a node in the Hadoop cluster that accepts tasks such as Map, Reduce and Shuffle operations from a Job-Tracker. Task-Tracker is set up with set of slots which depicts the number of tasks it can accept. The Task-Tracker spawns a separate JVM processes to do the actual work. The Task-Tracker supervises these spawned processes, capturing the output and exit codes. When the process finishes, successfully or not, the task tracker notifies the Job-Tracker. The Task-Trackers also transmit

heartbeat messages to the Job-Tracker, usually every few minutes, to reassure the Job-Tracker that it is still alive.
These messages also inform the Job-Tracker of the number of available slots, so the Job-Tracker can stay up to date with where in the cluster work can be assigned [20].

I. Map Reduce

Map-reduce is a Parallel Programming approach used for extracting and analyzing information from unstructured Big Data storage [8]. In Map-reduce there is a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function which will immix all intermediate values associated with the same intermediate key.

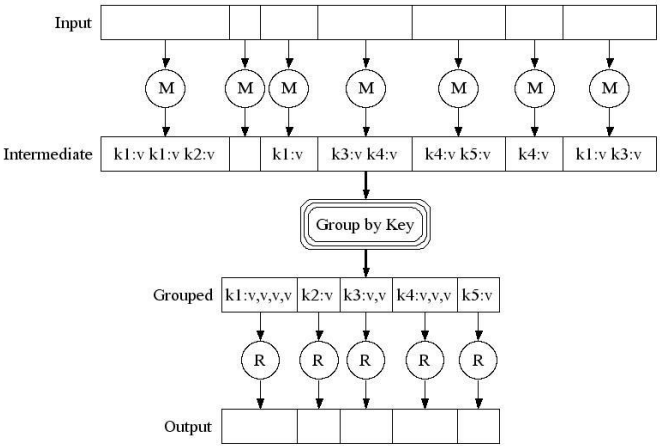

Fig.2 Block diagram of Map Reduce.

Mapping (M) is done on input data to get intermediate key/value pairs as shown in Figure 2 then this intermediate data is grouped by key. Example: All values v with key k1 in one group, all values v with key k2 etc. This grouped data is reduced to give following output i.e. {k1, 4} {k2, 1} {k3, 2} {k4, 3} and {k5, 1}

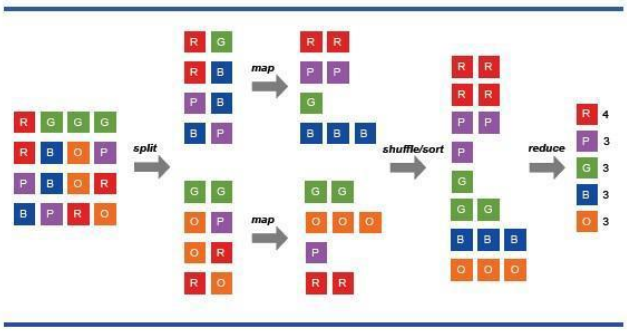

Fig.3 Example of Map Reduce.

Above is example of color data on which map-reduce is performed. The 16 blocks are split into two sets with 8 blocks in each and it is mapped to arrange blocks with respect to colors where R is red, P is pink, G is green, B is blue and O is orange. The first set contains 2 red, 2 pink, 1 green, 3 blue and second set contains 2 green, 3 orange, 1 pink, 2 red. These 2 sets are shuffled or sorted to get a single set and is reduced to give following output {R, 4} {P, 3} {G, 3} {B, 3} {O, 3}.

Following is a code for map-reduce: The mapper read-s input records and produces <word, 1> as intermediate pairs. After shuffling, intermediate counts associated with the same word are passed to a reducer, which adds the counts together to produce the sum [8].

III. INSTALLATION AND CONFIGURING OF HADOOP FRAMEWORK

A. Installation of Hadoop Framework

Download Hadoop from the Apache Download Mirrors and extract the contents of hadoop in /user/local. Change the owner of all files to hadoop user and hadoop group using chown command [6].

bindu@ubuntu:~$ su root
Password: *
root@ubuntu:~# cd /user/local/
root@ubuntu:/user/local#wget http://apache.communilink.net/hadoop/core/hadoop-0.20.2/hadoop-0.20.2.tar.gz
root@ubuntu:/user/local# tar -xvf hadoop-0.20.2.tar.gz
root@ubuntu:/user/local# chown -R hadoop: hadoop hadoop-0.20.2
root@ubuntu:/user/local# ln -s hadoop-0.20.2/ hadoop

ln command lets a file/directory on disk be accessed with more than one file/directory name, hadoop can be used instead of hadoop-0.20.2/ and then remove the tar file after extraction

root@ubuntu:/user/local# rm -rf hadoop-0.20.2.tar.gz

B. Configuration of Hadoop Framework in Ubuntu

One problem with IPv6 on Ubuntu is that using 0.0.0.0 for the various networking-related Hadoop configuration options will result in Hadoop binding to the IPv6 addresses of Ubuntu box. One can disable IPv6 only for Hadoop by adding the lines of IPv4 shown in figure to conf/hadoop-env.sh. The environment variable that has to be configured for Hadoop is JAVA_HOME. Open conf/hadoop-env.sh set the JAVA_HOME environment variable to the Sun JDK/JRE 6 directory

hadoop@ubuntu:/user/local/hadoop$ vi conf/hadoop-env.sh
export JAVA_HOME=/user/lib/jvm/java-6-sun
export HADOOP_OPTS="-Djava.net.preferIPv4Stack=true"

As of Hadoop 0.20.x and 1.x, the configuration settings previously found in hadoop-site.xml were moved to core-site.xml (hadoop.tmp.dir, fs.default.name), mapred-site.xml (mapred.job.tracker) and hdfs-site.xml (dfs.replication). Configure the directory where Hadoop will store its data files, the network ports it listens to, etc. The hadoop.tmp.dir variable can be changed to the directory of own choice.

Here the directory is /home/hadoop/cloud. Hadoop‘s default configurations use hadoop.tmp.dir as the base temporary directory both for the local file system and HDFS [6]. If required localhost can be replaced with ubuntu in following xml files.

hadoop@ubuntu:/user/local/hadoop$ mkdir ~/cloud

hadoop@ubuntu:/user/local/hadoop$ vi conf/core-site.xml

```
<configuration>
<property>
<name>hadoop.tmp.dir</name>
<value>/home/hadoop/cloud/hadoop-${user.name}</value>
</property>
<property>
<name>fs.default.name</name>
<value>hdfs://localhost:9000</value>
<description>The name of the default file system.
</description>
</property>
</configuration>
```

hadoop@ubuntu:/user/local/hadoop$ vi conf/mapred-site.xml

```
<configuration>
<property>
<name>mapred.job.tracker</name>
<value>localhost:9001</value>
<description>The host and port that the MapReduce job tracker runs at. If _local‘, then jobs are run in-process as a single map and reduce task.
</description>
</property>
</configuration>
```

hadoop@ubuntu:/user/local/hadoop$ vi conf/hdfs-site.xml

```
<configuration>
<property>
<name>dfs.replication</name>
<value>1</value>
<description>Default block replication.
The actual number of replications can be specified when the file is created. The default is used if replication is not specified in create time.
</description>
</property>
</configuration>
```


IV. WORD COUNT JOB OPERATION USING MAP REDUCE

As our project is dealt in single node Hadoop cluster with master and slave on same machine, ubuntu should be mentioned in master and slave files.

hadoop@ubuntu:/usr/local/hadoop$ vi conf/masters

hadoop@ubuntu:/usr/local/hadoop$ vi conf/slaves

And next to this step is starting up Hadoop installation and formatting the Hadoop file system, which is implemented on top of the local file systems of cluster. This has to be carried out when first time Hadoop installation is done. Do not format a running Hadoop file system, this will cause all data to be erased

hadoop@ubuntu:/usr/local/hadoop$ bin/hadoop namenode –format

Now run following command which will start up a Name Node, Data Node, Job Tracker and a Task Tracker on the machine.

hadoop@ubuntu:/usr/local/hadoop$ bin/start-all.sh

Now as to run the Java process status tool jps and to list all processes in Hadoop.

hadoop@ubuntu:/usr/local/hadoop$ jps

5621 JobTracker
5782 TaskTracker
5861 Jps
5545 SecondaryNameNode
5372 DataNode
5200 NameNode

The below is the Driver, Mapper and Reducer Code respectively.

```
package com.bindu.wordcount;
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.examples.WordCount.TokenizerMapper;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.reduce.IntSumReducer;
```

```
public class WordCount
{
  // DRIVER CODE
    public static void main(String args[]) throws Exception
    {
    Configuration conf = new Configuration();
    Job job = new Job(conf,"Word Count");
    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(args[0]));
    FileOutputFormat.setOutputPath(job,new Path(args[1]));
  System.exit(job.waitForCompletion(true)?0:1);
    }
  // MAPPER CODE
    public static class TockenizerMapper 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);
         }
    }

    // REDUCER CODE
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);
            }
        }
    }
}
```

V. RESULTS AND CONCLUSION

The Word Count Job operation implemented considering speed as priority, it is a simple application that counts the number of occurrences of each word in an input set. Word Count reads the input text files and counts how often the typed in words occur. The input is text file and the output is text file, each line of which contains a word and the count of how often it occurred, separated by a space (or) tab (or) comma.

root@ubuntu:/home/bindu# hadoop fs –mkdir /WordCountInput

root@ubuntu:/home/bindu# hadoop fs -put Input-Big.txt /WordCountInput

root@ubuntu:/home/bindu# hadoop jar BATCH-WORDCOUNT.jar WordCountNew /root/user/bindu/WordCountInput/Input-Big.txt /root/user/bindu/WordCountOutput

```
15/03/24 02:50:00 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments. Applications should implement Tool for the same.
15/03/24 02:50:00 INFO input.FileInputFormat: Total input paths to process: 1
15/03/24 02:50:00 WARN snappy.LoadSnappy: Snappy native library is available
15/03/24 02:50:00 INFO util.NativeCodeLoader: Loaded the native-hadoop library
15/03/24 02:50:00 INFO snappy.LoadSnappy: Snappy native library loaded
15/03/24 02:50:00 INFO mapred.JobClient: Running job: job_201408310158_0002
15/03/24 02:50:01 INFO mapred.JobClient:  map 0% reduce 0%
15/03/24 02:50:07 INFO mapred.JobClient:  map 100% reduce 0%
15/03/24 02:50:14 INFO mapred.JobClient:  map 100% reduce 33%
15/03/24 02:50:16 INFO mapred.JobClient:  map 100% reduce 100%
15/03/24 02:50:16 INFO mapred.JobClient: Job complete: job_201408310158_0002
15/03/24 02:50:16 INFO mapred.JobClient: Counters: 26
15/03/24 02:50:16 INFO mapred.JobClient:   Job Counters
15/03/24 02:50:16 INFO mapred.JobClient:     Launched reduce tasks=1
15/03/24 02:50:16 INFO mapred.JobClient:     SLOTS_MILLIS_MAPS=5140
15/03/24 02:50:16 INFO mapred.JobClient:     Total time spent by all reduces waiting after reserving slots (ms)=0
15/03/24 02:50:16 INFO mapred.JobClient:     Total time spent by all maps waiting after reserving slots (ms)=0
15/03/24 02:50:16 INFO mapred.JobClient:     Launched map tasks=1
15/03/24 02:50:16 INFO mapred.JobClient:     Data-local map tasks=1
15/03/24 02:50:16 INFO mapred.JobClient:     SLOTS_MILLIS_REDUCES=8545
15/03/24 02:50:16 INFO mapred.JobClient:   FileSystemCounters
15/03/24 02:50:16 INFO mapred.JobClient:     FILE_BYTES_READ=139
15/03/24 02:50:16 INFO mapred.JobClient:     HDFS_BYTES_READ=153395
```

```
15/03/24 02:50:16 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=119182
15/03/24 02:50:16 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=122
15/03/24 02:50:16 INFO mapred.JobClient:   Map-Reduce Framework
15/03/24 02:50:16 INFO mapred.JobClient:     Map input records=5487
15/03/24 02:50:16 INFO mapred.JobClient:     Reduce shuffle bytes=139
15/03/24 02:50:16 INFO mapred.JobClient:     Spilled Records=22
15/03/24 02:50:16 INFO mapred.JobClient:     Map output bytes=251174
15/03/24 02:50:16 INFO mapred.JobClient:     CPU time spent (ms)=2000
15/03/24 02:50:16 INFO mapred.JobClient:     Total committed heap usage (bytes)=177016832
15/03/24 02:50:16 INFO mapred.JobClient:     Combine input records=25872
15/03/24 02:50:16 INFO mapred.JobClient:     SPLIT_RAW_BYTES=125
15/03/24 02:50:16 INFO mapred.JobClient:     Reduce input records=11
15/03/24 02:50:16 INFO mapred.JobClient:     Reduce input groups=11
15/03/24 02:50:16 INFO mapred.JobClient:     Combine output records=11
15/03/24 02:50:16 INFO mapred.JobClient:     Physical memory (bytes) snapshot=186167296
15/03/24 02:50:16 INFO mapred.JobClient:     Reduce output records=11
15/03/24 02:50:16 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=749965312
15/03/24 02:50:16 INFO mapred.JobClient:     Map output records=25872
```

root@ubuntu:/home/bindu# hadoop fs -ls /root/user/bindu/WordCountInput

```
Found 1 items
-rw-r--r--   1 root supergroup     153270 2014-08-31 02:33 /root/user/bindu/WordCountInput/Input-
Big.txt
```

root@ubuntu:/home/bindu# hadoop fs -ls /root/user/bindu/WordCountOutput

```
Found 3 items
-rw-r--r--   1 root supergroup          0 2014-08-31 02:50 /root/user/bindu/WordCountOutput/_SUCCESS
drwxrwxrwx   - root supergroup          0 2014-08-31 02:50 /root/user/bindu/WordCountOutput/_logs
-rw-r--r--   1 root supergroup        122 2014-08-31 02:50 /root/user/bindu/WordCountOutput/part-r-
00000
```

root@ubuntu:/home/bindu# hadoop fs -cat /root/user/bindu/WordCountOutput/part-r-00000

```
good  4312
hadoop  4312
having  2156
is 4312
knowledge  1078
leader1078
learn  1078
market  3234
now  2156
people  1078
the  1078
```


Internally each mapper takes a line as input and breaks it into words. It then emits a key/value pair of the word and 1. Each reducer sums the counts for each word and emits a single key/value with the word and sum.

As an optimization, the reducer is also used as a combiner on the map outputs. This reduces the amount of data sent across the network by combining each word into a single record and helps in optimizing the size of the data to be transferred and optimal results are obtained respectively. In this way the above project illustrates the importance of Hadoop in current Big-Data world, power of Map-reduce algorithm and necessity of processing the data.

REFERENCES

[1] http://adcalves.wordpress.com/2010/12/12/a-hadoop-primer/
[2] http://hadoop.apache.org/
[3] http://programminggems.wordpress.com/2012/02/20/cloud-foundry-install-error-no-candidate-version-available-for-sun-java6-bin/
[4] https://launchpad.net/ubuntu/+ppas
[5] http://packages.debian.org/squeeze/sun-java6-jdk
[6] http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-single-node-cluster/
[7] Map-Reduce: Simplified Data Processing on Large Clusters, Jeffrey Dean and Sanjay Ghemawat jeff@google.com, sanjay@google.com Google, Inc.
[8] Oracle In-Database Hadoop: When MapReduce Meets RDBMS, Xueyuan Su Computer Science Yale University New Haven, CT 06520 xueyuan.su@yale.edu Garret Swart Oracle Corporation Redwood Shores, CA 94065 , garret.swart@oracle.com SIGMOD‘12, May 20–24, 2012,
[9] "Our abstraction is inspired by the map and reduce primitives present in Lisp and many other functional languages." - "MapReduce: Simplified Data Processing on Large Clusters", by Jeffrey Dean and Sanjay Ghemawat; from Google Research.
[10] Lämmel, R. (2008). "Google's Map Reduce programming model — revisited". Science of Computer Programming 70: 1. doi:10.1016/j.scico.2007.07.001.
[11] Cheng-Tao Chu; Sang Kyun Kim, Yi-An Lin, Yuan Yuan Yu, Gary Bradski, Andrew Ng, and Kunle Olukotun. "Map-Reduce for Machine Learning on Multicore". NIPS 2006.
[12] He, B.; Fang, W.; Luo, Q.; Govinda Raju, N. K.; Wang, T. (2008). "Mars: a MapReduce framework on graphics processors". Proceedings of the 17th international conference on Parallel architectures and compilation techniques - PACT '08. p. 260. doi:10.1145/1454115.1454152. ISBN 9781605582825.
[13] Chen, R.; Chen, H.; Zang, B. (2010). "Tiled-MapReduce: optimizing resource usages of data-parallel applications on multicore with tiling". Proceedings of the 19th international conference on Parallel architectures and compilation techniques - PACT '10. p. 523. doi:10.1145/1854273.1854337. ISBN 9781450301787.
[14] US Patent 7,650,331: "System and method for efficient large-scale data processing "

[1]G HIMA BINDU received her MCA from R.V.R and J. C College Of Engineering, Guntur. She worked as Assistant Professor in Department of MCA, Chebrolu Engineering College, Chebrolu, Guntur and is presently pursuing her M.Tech in Chaitanya Engineering College, Visakhapatnam, Andhra Pradesh, India.

[2]Y SATISH KUMAR received his M.Tech in Power and Industrial Drives from GITAM Institute of Technology, GITAM University, Visakhapatnam and MCA from V R Institute of PG Studies, Nellore. He has Industrial Experience of two years from Vista InfoTech, Bangalore and worked as Teaching Assistant (2007-10) in V R Institute of PG Studies, Nellore, Assistant Professor (2010-13) in Sanketika Institute of Technology & Management, Visakhapatnam and is presently working as an Assistant Professor in Chaitanya Engineering College, Visakhapatnam, Andhra Pradesh, India.

