International Journal of Computer Applications (0975 – 8887)
Volume 121 – No.19, July 2015

Implementation and Performance Analysis of Academic_MapReduce Algorithm (AcdMR)

Ratnamala Mantri, Ashwini Jewalikar

Asst. professor
Pune institute of computer technology
Pune.

ABSTRACT

In current scenario all organizations whether it is commercial, financial or educational, face the problem of maintaining and processing huge data. If we consider the case of educational institute, here thousands of student attendance records are generated per day. This will multiply per day, week and year as a result generates vast amount of data and subsequently increases the processing time apart from cost. In order to process the data efficiently, we have proposed MapReduce based algorithm (AcdMR) for processing of an academic data in our first paper. This work describes practical implementation of AcdMR and performance analysis based on distinct cluster configuration over dataset size (1GB, 5GB).

Keywords

Big Data, MapReduce, Data analysis, Parallel programming, Distributed computing, Hadoop,

1. INTRODUCTION

Apache Hadoop ,well known platform for Big Data processing, is an open source software framework. Hadoop uses large clusters i.e. Cluster of thousands of nodes.

Two major parts of Hadoop are MapReduce and HDFS. HDFS is a hadoop distributed file system. It is designed for storing Big files which can be megabyte, gigabyte or terabyte in size. Files are stored in the form of blocks with 64 MB default block size .It also allow to replicate data with default replica factor 3. Hdfs has two types of node in cluster, operating in master-slave environment: a Namenode (the master) and number of Datanodes (slaves). File system Metadata is maintained by Namenode and actual data is stored on Datanodes in the form of blocks. Hadoop provides command line interface and many other interfaces to interact with HDFS[3,8].
MapReduce is a distributed and parallel programming framework .It is used to compute problems that can be parallelized by mapping a function over a given dataset and then combining the results using Reduce [2,4]. Map reduce framework sits on top of the HDFS.

Hadoop is an open source implementation of MapReduce which process vast amount of data on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner .Hadoop cluster configure two nodes a Jobtracker and many tasktracker to run Mapreduce Job. Job is submitted to the *Job Tracker,* it pushes job to the available *Task Trackers* nodes in the cluster The JobTracker, a program which coordinates and manages the jobs. It accepts job submissions from users, provides job monitoring and control, and manages the distribution of tasks in a job to the Task Tracker nodes. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks [3,8].

2. BACKGROUND

MapReduce was developed within Google for processing of large amounts of raw data. In order to process large data in a reasonable time data is distributed across thousands of machines. This distribution implies parallel computing since each computation performed on each CPU but with different data set [5]. It is Schema free and user has to implement only two functions Map and Reduce without knowledge of parallel and distributed systems [6].

Two Comparison approaches to large scale data analysis using Map Reduce and parallel DBMS shows that there is no question that MR does a superior job of minimizing the amount of work that is lost when a hardware failure occurs [7]. In any educational institute, if we consider case of students attendance student has on an average six attendance records per day. With 2500 students, this Results in around 15000 attendance records per day, which in turn generates vast amount of data [1].

2.1. MapReduce

MapReduce framework operates exclusively on key & value pairs, i.e. input to the job is in the form key & value pairs and generated output is a list of key & value pairs. Map input, the value is a chunk of data file and the key is generally the offset of the chunk from the beginning of the data file. The output consists of a collection of key-value pairs which are input for the reduce function. The content of the key-value pairs depends on the specific implementation [8]. Computing stages of MapReduce are shown in Figure 1 [1].

The MapReduce job is configured by setting different parameters specific to the job. The user can also specify the number of maps and reducer tasks. The user also has to specify the format of the input, and the locations of the input. The Hadoop Mapreduce framework uses this information to split of the input into several pieces. Each input piece is fed into a user-defined map function. The map tasks process the input data and emit intermediate data.

$$Map\ (k1,\ v1) \rightarrow list\ (K2,\ v2)$$

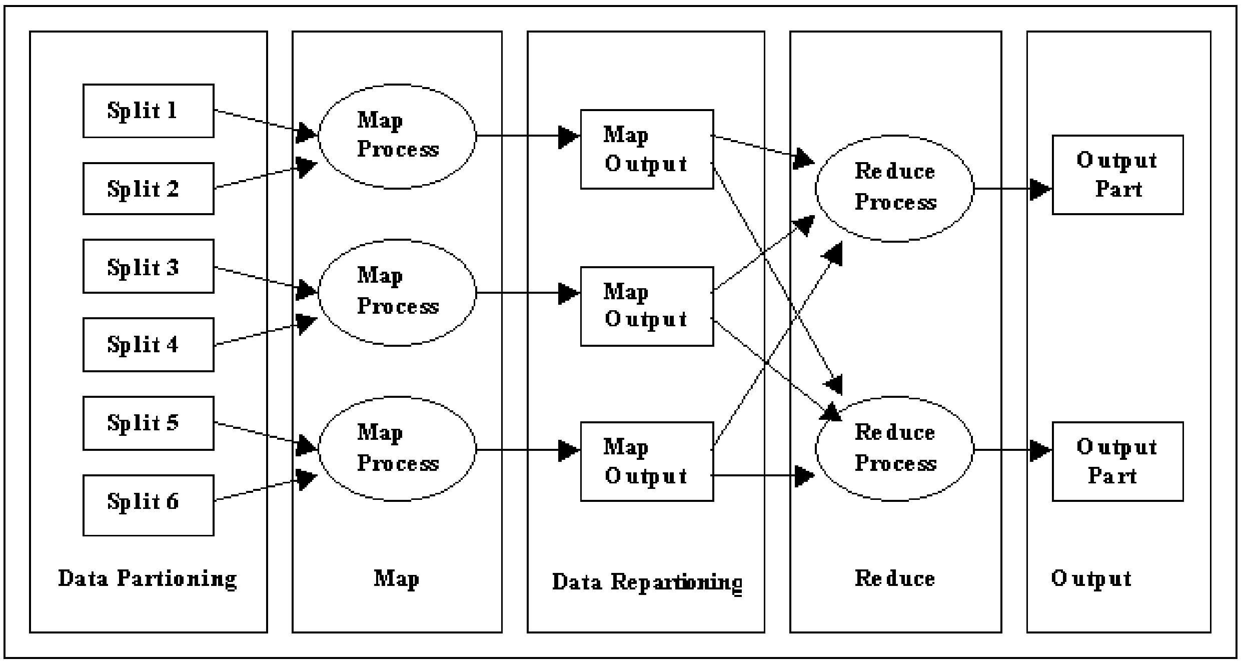


Figure 1: Computing stages of MapReduce

The map and reduce phase does the sorting of the data and storing in partitions as per reducer operations. The default practitioner present in Hadoop is HashPartitioner. Users can optionally specify a *combiner,* for aggregation of intermediate outputs for reducing the data transfer overhead [3,8]. After sorting, MapReduce framework collects all pairs with the same key from all lists and groups them together, which helps to generate different keys and Reduce function can be applied parallel on each group, which in turn produces a collection of values.

Reduce (k2, list (v2)) → list (v3).

Output of the Reduce method is then stored in text file, which is stored in Hadoop distributed file system(HDFS).

2.2. MapReduce Programming

The mapper and Reducer are two main component of MapReduce. Mapper and Reducer interfaces, typically they are implemented to provide the map and reduce methods.

Since Hadoop APIs are available in many languages, we have used Java .MapReduce job is initialized by creating an object of *Configuration* class. Methods like ***setJarByClass, setOutputKeyClass, setOutputValueClass, setMapperClass, setReducerClass*** *etc..* are invoked to set different parameters.

The setOutputKeyClass() and setOutputValueClass() identifies the data type emitted by the reducer. The assumption is made for these output types will be mapped default. If the assumption is not true, these methods are override in the JobConf class. *InputFormat* controls the use of input types fed to the mapper.

3. ACADEMIC MapReduce ALGORITHM (AcdMR)

In this section we demonstrated how MapReduce based algorithm analyze the Students attendance data .Every attendance record contains following information. *Academic_Year(AY), Branch(Br), Student_id(S_{id}), Class (Cl), Subject_id(Sb_{id}), Date(Dt), Attendance (1-Present, 0-Absent)*

Here we are processing the data with different keys. Equation 1 models the input.txt file, which is input for the MapReduce. This file is stored in Hadoop distributed file system (HDFS)
Equation 1: input.txt *Input= Academic_Year(AY) X Branch(Br) X Student_id(S_{id}) X Class (Cl) X Subject(Sb_{id}) X Date(Dt) X Attendance*
Input may be a partial Cartesian product .
Some assumptions are made, S_{id} (Student_id) is taken instead of his (name+ roll no.), each key is formed along with *Academic_Year(AY),* Branch. In attendance, 1-Present, 0-Absent [1].

Following MapReduce based algorithm is designed to find students whose subject wise attendance is less than say 50%.

Firstly we describe working of an algorithm, each map function processes different data chunk of input file. For each record or line of input data, *Ti,* $1 \le i \le N$, key will be Prepared K_i , $1 \le i \le N$, *(N is no of lectures conducted*). The case, calculating subject wise attendance of the student then *Student_id* X *Subject_id* is a composite key. For each key *Ki* associated value will be taken from corresponding attendance field (present or absent). The output of Map function is ***list*** of <key_i', $value_i$'> pair. All pairs with the same key from all lists are collected and groups them together in MapReduce framework, thus creating one group for each one of the different generated keys < *key$_i$', list(value$_i$')>, 1≤i≤N.*

Each reducer is responsible for processing the list of values associated with a different key and generate the output in key value pair. In this input key *key'* is not processed only their lists of values are processed.

```
//Acd_MapReduce algorithm for subjectwisw attendence
Job_Subjectwise_Map(key,value)
//(key,value)
        student_atted(S)=getstudent_atted(value);
        for each stud_atted(S) do
                key'=$S.S_{id}$ +$S.Sb_{id}$
                value'=S.Attendance
                emit(key', value')
        end     //for loop
End     //Map method
Job_Subjectwise_Reduce(key',list(value'))
//(key', value')
        int count=0,sum=0;
        for each value' in the list do
                count++;
                sum+=value'.get();
                value"=sum/count;
        end     //for loop

        if(value''<50)
        emit(key' ,value")
End  //Reduce method
```

4. EXPERIMENTAL SETUP

Experiment has been carried out for each dataset size (1GB and 5GB). Each dataset has been processed on five different clusters. The numbers of reducers is set to number of nodes in the cluster (Default is 1) to speed up the execution. For each execution processing time is recoded in seconds.

Table 1.Node configuration

Processor	Intel core 2 Duo 2.22GHz
L1 cache	2048 KB
RAM	2GB
Operating system	Ubuntu 10.04
Mapreduce environment	Hadoop-1.2.1, Sun Java 6

Hadoop cluster has been set up using 5 nodes as shown in figure 2 .It uses single rack topology. Master is designated as NameNode and Jobtracker. Slaves are designated as DataNodes and Tasktracker. Table 1 illustrates system configuration for nodes.

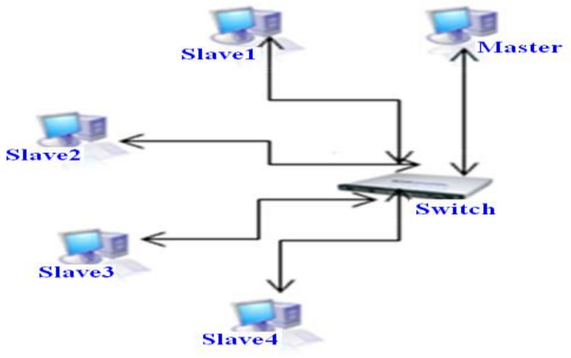


Figure2: Experimental setup

Performance analysis has been carried out over distinct cluster configuration corresponding to *1, 1+1, 1+2, 1+3, 1+4* as shown in Table 2. Here 1 indicates single node cluster means master and slaves are on same machine. *1+1* is one master and two slaves. *1+2* one master and three slaves. So in general *1+n* is 1 master and n+ 1 slave.

Table 2:Cluster Configuration

Cluster Id	Cluster size
C1	1
C11	1+1
C12	1+2
C13	1+3
C14	1+4

5. RESULTS & DISCUSSION

Table 3 and 4 shows processing time of AcdMR job with 1 GB and 5GB size of dataset respectively.

Table3. Processing time for Default Vs modified configuration parameter (1GB)

Cluster ID	Default processing time(sec)	Improved processing time(sec)	Computing speed
C1	135	135	1
C11	116	116	1.16
C12	82	72	1.88
C13	75	62	2.18
C14	67	54	2.50

Computing speed for given cluster is calculated by taking the ratio improved response time of given cluster to time required on single node for same data size.

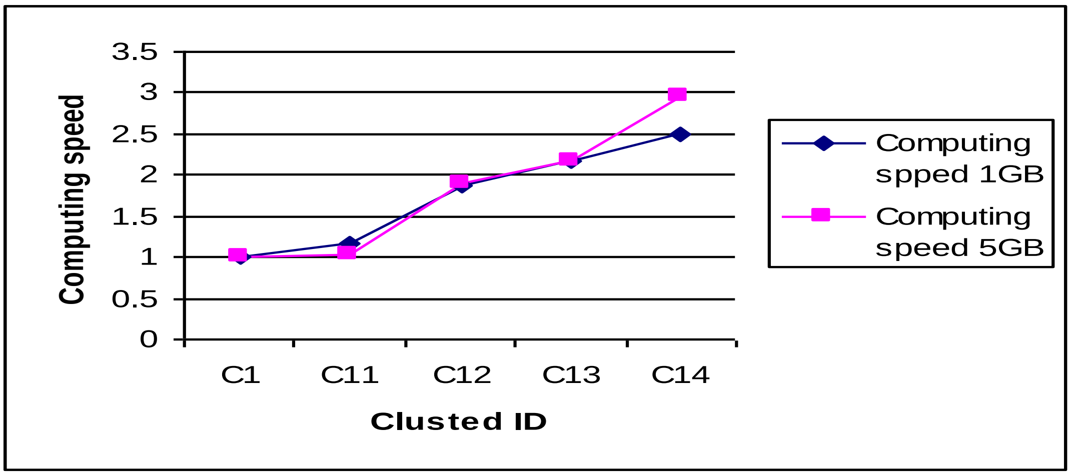
Ideal performance of hadoop scales linearly i.e. computing speed should increase linearly with the increase of cluster size used for computation.

Table4. Processing time for Default Vs modified

Cluster ID	Default processing time(sec)	Improved processing time(sec)	Computing speed
C1	565	565	1
C11	547	547	1.03
C12	380	300	1.9
C13	350	260	2.17
C14	300	190	2.97

configuration parameter (5GB)

Graph 1 shows to get Hadoop Performance, dataset size must be larger. One observation is made that slow network communication affects the performance.



Graph1:Speedup curve for dataset size (1GB and 5GB)

6. CONCLUSION

This work implemented efficient processing of vast amount of academic data using Hadoop. We have focused on Processing of student's attendance with different keys. As a proof of performance analysis, benchmarking is done based on distinct cluster configuration over dataset size (1GB, 5GB). Result shows better performance is obtained by selecting number of reducer tasks value over default value. As expected, results in table shows the processing time to complete each experimental decreased (Processing speed increased) for a given data size when number of nodes increased. It has been observed that Hadoop produces effective results as data size increases.

7. REFERENCES

[1] Ratnamala Mantri, Rajesh Ingle and Prachi Patil, "SCDP: Scalable, Cost –Effective, Distributed and Parallel Computing Model for Academics," ICNCS,Vol 5, 77-80, 2011. ISBN 978-1-4244-8677-9 published by IEEE.

[2] Maryam Kontagora, Horacio Gonz´lez–V´lez, "Benchmarking a MapReduce Environment on a Full Virtualization Platform," 2010 IEEE International Conference on Complex, Intelligent and Software Intensive Systems,page-433-438,2010

[3] Tom White, "Hadoop: The definitive guide," O'Reilly Media / Yahoo Press, October 2010.

[4] J. Dean and S. Ghemawat, "MapReduce: Simplified data Processing on large clusters," in OSDI'04. San Francisco: USENIX, Dec. 2004, pp. 137–150.

[5] Google Code University, "Introduction to Parallel Programming and MapReduce", http://code.google.com/edu/parallel/mapreducetutorial.html, Nov 2010.

[6] Shimin Chen, Steven W. Schlosser, "Map-Reduce Meets Wider Varieties of Applications," research at Intel, pages 1- 8, 2008.

[7] Andrew Pavlo, Erik Paulson, Alexander Rasin, "A Comparison of Approaches to Large-Scale Data Analysis," ACM SIGMOD'09, June 29–July 2, 2009.

[8] The Apache Software Foundation, "Hadoop Map-Reduce Tutorial," Hadoop Project, https://hadoop.apache.org/docs/r1.0.4, Feb. 2013 .