ISSN:0975-9646

Map Reduce a Programming Model for Cloud Computing Based On Hadoop Ecosystem

Santhosh voruganti

Asst.Prof CSE Dept ,CBIT,
Hyderabad,India

***Abstract*— Cloud Computing is emerging as a new computational paradigm shift.Hadoop MapReduce has become a powerful Computation Model for processing large data on distributed commodity hardware clusters such as Clouds. MapReduce is a programming model developed for large-scale analysis. It takes advantage of the parallel processing capabilities of a cluster in order to quickly process very large datasets in a fault-tolerant and scalable manner. The core idea behind MapReduce is mapping the data into a collection of key/value pairs, and then reducing over all pairs with the same key. Using key/value pairs as its basic data unit, the framework is able to work with the less-structured data types and to address a wide range of problems. In Hadoop, data can originate in any form, but in order to be analyzed by MapReduce software, it needs to eventually be transformed into key/value pairs.In this paper we implements MapReduce programming model using two components: aJobTracker (masternode) and many TaskTrackers (slave nodes).**

***Keywords*— Cloud Computing, Hadoop Ecosystem, apreduce, Hdfs,**

1.INTRODUCTION

'cloud' is an elastic execution environment of resources involving multiple stakeholders and providing a metered service at multiple granularities for a specified level of quality of service to be more specific, a cloud is a platform or infrastructure that enables execution of code (services, applications etc.), in a managed and elastic fashion, whereas "managed" means that reliability according to predefined quality parameters is automatically ensured and "elastic" implies that the resources are put to use according to actual current requirements observing overarching requirement definitions implicitly, elasticity includes both up and downward scalability of resources and data, but also load balancing of data throughput with Google Docs. Due this combinatorial capability, these types are also often referred to as "components"

Cloud) Infrastructure as a Service (IaaS) also referred to as Resource Clouds provide (managed and scalable) resources as services to the user in other words, they basically provide enhanced virtualisation capabilities. Accordingly, different resources may be provided via a service interface:Data & Storage Clouds deal with reliable access to data of potentially dynamic size, weighing resource usage with access requirements and / or quality definition.
Examples
Amazon S3, SQL Azure

Compute Clouds provide access to computational resources, i.e. CPUs. So far, such low-level resources cannot really be exploited on their own, so that they are typically exposed as part of a "virtualized environment" (not to be mixed with PaaS below), i.e. hypervisors. Compute Cloud Providers therefore typically offer the capability to provide computing resources (i.e. raw access to resources unlike PaaS that offer full software stacks to develop and build applications), typically virtualised, in which to execute cloudified services and applications. IaaS (Infrastructure as a Service) offers additional capabilities over a simple compute service.
Examples
Amazon EC2, Zimory, Elastichosts

(Cloud) Platform as a Service (PaaS): provide computational resources via a platform upon which applications and services can be developed and hosted. PaaS typically makes use of dedicated APIs to control the behaviour of a server hosting engine which executes and replicates the execution according to user requests (e.g. access rate). As each provider exposes his / her own API according to the respective key capabilities, applications developed for one specific cloud provider cannot be moved to another cloud host there are however attempts to extend generic programming models with cloud capabilities (such as MS Azure).
Examples
Force.com, Google App Engine, Windows Azure (Platform)

Clouds) Software as a Service (SaaS) also sometimes referred to as Service or Application Clouds are offering implementations of specific business functions and business processes that are provided with specific cloud capabilities, i.e. they provide applications / services using a cloud infrastructure or platform, rather than providing cloud features themselves. Often, kind of standard application software functionality is offered within a cloud.
Examples
Google Docs, Salesforce CRM, SAP Business by Design.
Overall, Cloud Computing is not restricted to Infrastructure / Platform / Software as a Service systems, even though it provides enhanced capabilities which act as (vertical) enablers to these systems. As such, I/P/SaaS can be considered specific "usage patterns" for cloud systems which relate to models already approached by Grid, Web Services etc. Cloud systems are a promising way to implement these models and extend them further

TECHNICAL ASPECTS:

The main technological challenges that can be identified and that are commonly associated with cloud systems are

Virtualisation is an essential technological characteristic of clouds which hides the technological complexity from the user and enables enhanced flexibility (through aggregation, routing translation).

Multitenancy is a highly essential issue in cloud systems, where the location of code and / or data is principally unknown and the same resource may be assigned to multiple users (potentially at the same time). This affects infrastructure resources as well as data / applications / services that are hosted on shared resources but need to be made available in multiple isolated instances. Classically, all information is maintained in separate databases or tables, yet in more complicated cases information may be concurrently altered, even though maintained for isolated tenants. Multitenancy implies a lot of potential issues, ranging from data protection to legislator issues

Security Privacy and Compliance is obviously essential in all systems dealing with potentially sensitive data and code.

Data Management is an essential aspect in particular for storage clouds, where data is flexibly distributed across multiple resources. Implicitly, data consistency needs to be maintained over a wide distribution of replicated data sources. At the same time, the system always needs to be aware of the data location (when replicating across data centres) taking latencies and particularly work-load into consideration. As size of data may change at any time, data management addresses both horizontal and vertical aspects of scalability. Another crucial aspect of data management is the provided consistency guarantees (eventual vs. strong consistency, transactional isolation vs. no isolation, atomic operations over individual data items vs. multiple data times etc.).

APIs and / or Programming Enhancements are essential to exploit the cloud features: common programming models require that the developer takes care of the scalability and autonomic capabilities him/ herself, whilst a cloud environment provides the features in a fashion that allows the user to leave such management to the system

Metering of any kind of resource and service consumption is essential in order to offer elastic pricing, charging and billing. It is therefore a precondition for the elasticity of cloud.Tools are generally necessary to support development, adaptation and usage of cloud services

2.RELATED WORK

2.1 HADOOP

Hadoop is an open-source framework for writing and running distributed applications that process very large data sets. There has been a great deal of interest in the framework, and it is very popular in industry as well as in academia. Hadoop cases include: web indexing, scientific simulation, social network analysis, fraud analysis, recommendation engine, ad targeting, threat analysis, risk modeling and other. Hadoop is core part of a cloud computing infrastructure and is being used by companies like Yahoo, Facebook, IBM, LinkedIn, and Twitter. The main benefits of Hadoop framework can be summarized as follows:

Accessible: it runs on clusters of commodity servers

Scalable: it scales linearly to handle larger data by adding nodes to the cluster

Fault-tolerant: it is designed with the assumption of frequent hardware failures

Simple: it allows user to quickly write efficiently parallel code

Global: it stores and analyzes data in its native format

Hadoop is designed for data-intensive processing tasks and for that reason it has adopted a move- code-to-data" philosophy. According to that philosophy, the programs to run, which are small in size,Are transferred to nodes that store the data. In that way, the framework achieves better performance and resource utilization. In addition,

Hadoop solves the hard scaling problems caused by large amounts of complex data. As the amount of data in a cluster grows, new servers can be incrementally and inexpensively added to store and analyze it.

Hadoop has two major subsystems: the Hadoop Distributed File System (HDFS) and a distributed data processing framework called MapReduce. Apart from these two main components, Hadoop has grown into a complex ecosystem, including a range of software systems. Core related applications that are built on top of the HDFS are presented in Figure 2.1 and a short description per project is given in Table 2.1.1.

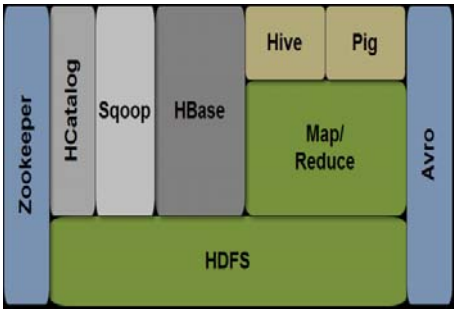


Figure 2.1: Hadoop Ecosystem

Project	Info
Hdfs	Hadoop distributed file system
Map reduce	Distributed computation framework
Zookeeper	High-performance collaboration service
Hbase	Column-oriented table service
Pig	Dataflow language and parallel execution
Hive	Data warehouse infrastructure
Hcatalog	Table and storage management service
Sqoop	Bulk data transfer
Avron	Data serialization system

Table 2.1.1: Project Descriptions

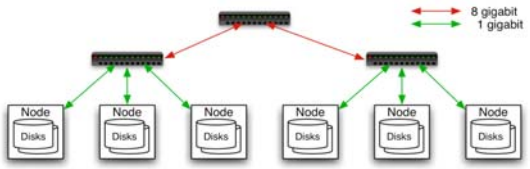


Figure 2.1.2 A Typical Hadoop cluster

3. HDFS

Hadoop comes with a distributed file system called HDFS which stands for Hadoop Distributed File System. HDFS manages the storage of files across the cluster and enables global access to them.In comparison to other distributed file systems, HDFS is on purpose implemented to be fault-tolerant, and to conveniently expose the location of data enabling Hadoop framework to take advantage of the move-code-to-data philosophy.HDFS is optimized for storing large files, for streaming the files at high bandwidth, for running in commodity hardware. While HDFS is not performing well when the user application requires low-latency data access, when there are a lot of small files, and when the application requires arbitrary file modification. The file system is architected using the pattern write-once, read-many-times (simple coherency model). A dataset is generated once, and multiple analyses are performed on it during time. It is expected that the analysis will require the whole dataset, hence fully reading the dataset is more important than the latency of a random read.

In HDFS, files are broken into block-sized chunks, which are independently distributed in a nodes. Each block is saved as a separate file in the node's local file system. The size of the block is large and a typical value would be 128MB, but it is a value chosen per client and per file. The large size of the block was picked, firstly, in order to take advantage of sequential I/0 capabilities of disks, secondly to minimize latencies because of random seeks and finally because it is logical input size to the analysis framework.

HDFS is also designed to be fault tolerant, which means that each block should remain accessible in the occur of system failures. The fault-tolerance feature is implemented through a replication mechanism. Every block is stored in more than one node making highly unlikely that it can be lost.By default, two copies of the block are saved on two different nodes in the same rank and a third copy in a node located in a different rank. The HDFS software continually monitors the data stored on the cluster and in case of a failure (node becomes unavailable, a disk drive fails, etc.) automatically restores the data from one of the known good replicas stored elsewhere on the cluster.

3.1 NAMENODE

The namespace of the filesystem is maintained persistently by a master node called namenode.Along with the namespace tree, this master node holds and creates the mapping between file blocks and datanodes. In other words, namenode knows in which node the blocks of a file are saved (physical location). The mapping is not saved persistently because it is reconstructed from the datanodes during start phase, and because it dynamically changes over time. The internal structure of a namenode is given in the Figure 3.1

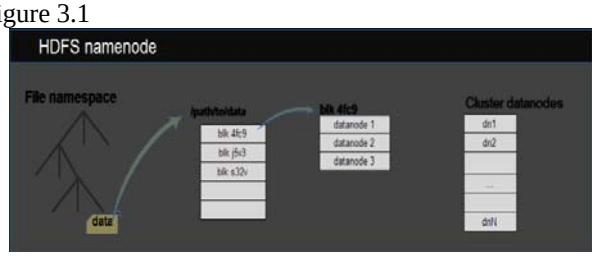


Figure 3.1 Namenode

Holding the critical information (namespace, file metadata, mapping) in a unique master node rather node than in a distributed way makes the system simpler. However, at the same time it makes this node a single point of failure (SPOF) and creates scalability issues. From performance point of view, it is required that the namenode holds the namespace entirely in RAM and that it will respond fast to a client request. For reliability, it is necessary for the namenode to never fail, because the cluster will not be functional. The information that is stored in the namenode should also never be lost, because it will be impossible to reconstruct the files from the blocks. For the above reasons,Great effort has been made in order to maintain the master node, but at the same time to overcome the drawbacks. In past versions, it was a common tactic to backup the information either in a remote NFS mount or using a secondary namenode in which the namespace image was periodically merged and could replace the original in case of failure.

3.2 DATANODE

As has been implied before, the blocks of a file are independently stored in nodes, which are called datanodes. Every datanode in the cluster, during startup, makes itself available to the name node through a registration process. A part from that, each datanode informs namenode which blocks has in its possession by sending a block report. A block reports are sent periodically or when a change takes place. Furthermore, every datanode sends heartbeat messages to the namenode to confirm that it remains operational and that the data is safe and available. If a datanode stops operating, there are error mechanisms in order to overcome the failure and maintain the availability of the block.Heartbeat messages also hold extra information, which helps the namenode run the cluster efficiently (e.g. storage capacity which enables namenode to make load balancing). One important architectural note is that namenode never directly calls datanodes; it uses replies to heartbeats to order a command to be executed in the datanode (e.g. to transfer a block to another node).

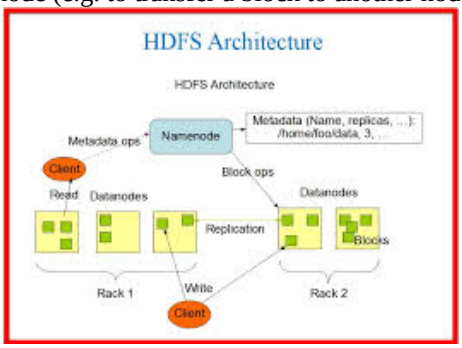


Figure 3.2.1 HDFS Architecture

4. MAP REDUCE

MapReduce programming model using two components: a Job Tracker (masternode) and many Task Trackers (slave nodes). The Job Tracker is responsible for accepting job requests,for splitting the data input, for defining the tasks required for the job, for assigning those tasks to be executed in parallel across the slaves, for monitoring the progress and finally for handling occurring failures. The

Task Tracker executes tasks as ordered by the master node. The task can be either a map (takes a key/value and generates another key/value) or a reduce (takes a key and all associated values and generates a key/value pair). The map function can run independently on each key/value pair, enabling enormous amounts of parallelism. Likewise, each reducer can also run separately on each key enabling further parallelism.When a job is submitted to the Job Tracker, the Job Tracker selects a number of Task Trackers (not randomly but according to data locality) to execute a map task (Mappers) and a number of Task Trackers to execute a reduce task (Reducers). The job input data is divided into splits and is organized as a stream of keys/values records. In each split there is a matching mapper which converts the original records into intermediate results which are again in the form of key/value. The intermediate results are divided into partitions (each partition has a range of keys), which after the end of the map phase are distributed to the reducers .Finally reducers apply a reduce function on each key.

4.1 MAP REDUCE COMPUTATION

A MapReduce paradigm is given in Figure 4.1 MapReduce is designed to continue to work in the face of system failures. When a job is running, MapReduce monitors progress of each of the servers participating in the job. If one of them is slow in returning an answer or fails before completing its work, MapReduce automatically starts another instance of that task on another server that has a copy of the data. The complexity of the error handling mechanism is completely hidden from the programmer.

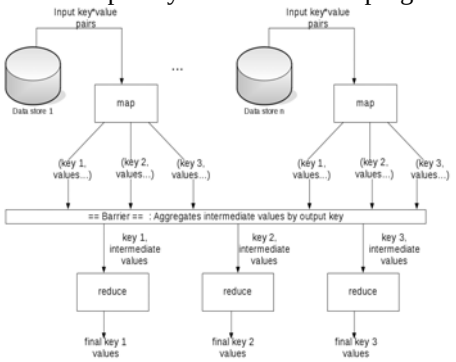


Figure 4.1 A Map Reduce Computation

MapReduce is triggered by the map and reduce operations in functional languages,such as Lisp. This model abstracts computation problems through two functions: map and reduce. All problems formulated in this way can be parallelized automatically.Essentially, the MapReduce model allows users to write map/reduce components with functional-style code. These components are then composed as a dataflow graph to explicitly specify their parallelism. Finally, the MapReduce runtime system schedules these components to distributed resources for execution while handling many tough problems: parallelization, network communication, and fault tolerance.

A map function takes a key/value pair as input and produces a list of key/value pairs as output. The type of output key and value can be different from input:

map :: (key1; value1) → list(key2; value2)… (1)

A reduce function takes a key and associated value list as input and generates a list of new values as output:

reduce :: (key2; list(value2)) -> list(value3)… (2)

A MapReduce application is executed in a parallel manner through two phases. In the first phase, all map operations can be executed independently from each other. In the second phase, each reduce operation may depend on the outputs generated by any number of map operations. All reduce operations can also be executed independently similar to map operations.

4.2 USES OF MAP REDUCE

At Google:
- Index building for Google Search
- Article clustering for Google News
- Statistical machine translation

At Yahoo!:
- Index building for Yahoo! Search
- Spam detection for Yahoo! Mail

At Facebook:
- Ad optimization
- Spam detection

5.IMPLEMENTATION

5.1 A MAPREDUCE WORKFLOW

When we write a MapReduce workflow, we'll have to create 2 scripts: the mapscript, and the reduce script. The rest will be handled by the Amazon ElasticMapReduce (EMR) framework.

When we start a map/reduce workflow, the framework will *split* the input into segments, passing each segment to a different machine. Each machine then runs the *map script* on the portion of data attributed to it.

The *map script* (which you write) takes some input data, and maps it to <key, value>pairs according to your specifications. For example, if we wanted to count word frequencies in a text, we'd have <word, count> be our <key, value> pairs.

MapReduce then would emit a <word, 1> pair for each word in the input stream. Note that the map script does no *aggregation* (i.e. actual counting) this is what the reduce script it for. The purpose of the map script is to model the data into <key,value> pairs for the reducer to aggregate.

Emitted <key, value> pairs are then "shuffled" (to use the terminology in the diagram below), which basically means that pairs with the same key are grouped and passed to a single machine, which will then run the *reduce script* over them.The *reduce script* (which you also write) takes a collection of <key, value> pairs and"reduces" them according to the user-specified reduce script.

In our word count example, we want to count the number of word occurrences so that we can get frequencies. Thus, we'd want our reduce script to simply sum the *values* of the collection of <key, value> pairs which have the same key.

The Figure word count example below illustrates the described scenario nicely

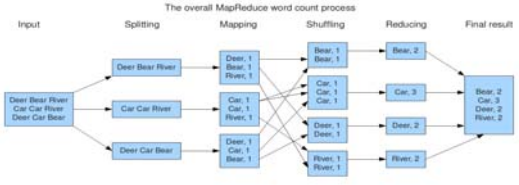


Figure 5.1 Word Count Example

5.2 WORDCOUNT MAPPER

```
public static class MapClass extends MapReduceBase
implements Mapper<LongWritable, Text, Text,
IntWritable>
{
private final static IntWritable one = new IntWritable(1);
private Text word = new Text();
public void map(LongWritable key, Text value,
OutputCollector<Text, IntWritable> output,
Reporter reporter) throws IOException {
String line = value.toString();
StringTokenizer itr = new StringTokenizer(line);
while (itr.hasMoreTokens()) {
word.set(itr.nextToken());
output.collect(word, one);
}
}
}
```

5.3 WORDCOUNT REDUCER

```
public static class Reduce extends MapReduceBase
implements Reducer<Text, IntWritable, Text,
IntWritable>
{
public void reduce(Text key, Iterator<IntWritable> values,
OutputCollector<Text, IntWritable> output,
Reporter reporter) throws IOException {
int sum = 0;
while (values.hasNext()) {
sum += values.next().get();
}
output.collect(key, new IntWritable(sum));
}
}
```

5.4 HADOOP JOB EXECUTION

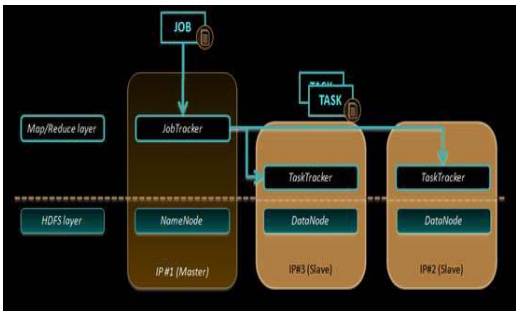
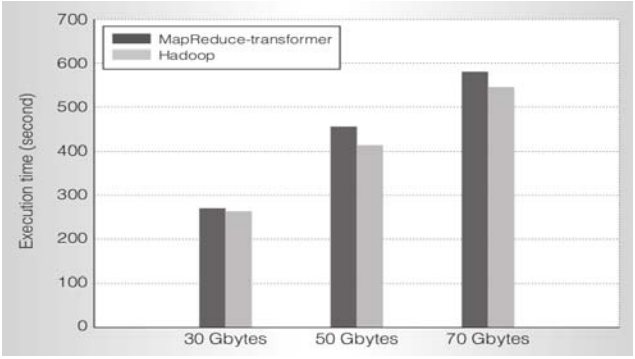


Figure 5.4 Hadoop Job Execution

Hadoop helps us to process huge data sets by distributing the replicated forms of same data into several datanodes whose information is stored in a namenode server. There is a job tracker that splits the job into several tasks each of which is handled by a task tracker. The split files are fed into mappers where the mapping function works and keys and values are generated as (k,v) sets. These are shuffled and put to reducers who cumulate or combine the count or value of similar data sets there by reducing redundancy of data. Also several parallel processing can be obtained by such a framework. The bottom line is that we divide the job, load it in HDFS, employ MapReduce on them, solve them in parallel, and write the cumulative results back to the HDFS. It ensures a powerful, robust and fault tolerant system that can be used to deploy huge data set processing as image processing, weather forecasting and genome grafting.

5.5 RESULTS



Statistics of Word Count Application

6.CONCLUSION AND FUTURE ENHANCEMENTS

With the emergence of Clouds and a general increase in the importance of data-intensive applications, programming models for data-intensive applications have gained significant attention: a prominent example being Map-Reduce. The usability and effectiveness of a programming model is dependent upon the desired degree of control in the application developement,deployment and execution.

Hadoop is in general best known for the MapReduce and the HDFS components. Hadoop is basically designed to efficiently process very large data volumes by linking many commodity systems together to work as a parallel entity. In other words, the Hadoop philosophy is to bind many smaller (and hence more reasonably priced) nodes together to form a single, cost-effective compute cluster environment.

Future programming frameworks must allow client systems to develop robust, scalable programming models that, while relying on parallel computation, abstracts the details of parallelism. The end programmer should be exposed only with a set of APIs rather than the details of the distributed hardware.

REFERENCES:

[1] Chris Miceli et al., Programming Abstractions for Data Intensive Computing on Clouds and Grids, 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, 2009.

[2] DEAN, J., AND GHEMAWAT, S. Mapreduce: Simplified data processing on large clusters. In *OSDI'04: Sixth Symposium on Operating System Design and Implementation* (December 2004).

[3] Shantenu Jha, Daniel S. Katz, Andre Luckow, Andre Merzky,Katerina Stamou, Understanding Scientific Applications For Cloud Environments,2009.

[4] Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J. and Brandic, I.(2009). Cloud computing and emerging IT platforms: Vision,hype, and reality for delivering computing as the 5thutility, *Future Generation Computer Systems—The InternationalJournal of Grid Computing: Theory Methods andApplications* **25**(6): 599–616.

[5] Chao Jin and Rajkumar Buyya, MapReduce Programming Model for.NET-based Cloud Computing, The University of Melbourne, Australia.

[6] H. Liu and D. Orban, "Cloud mapreduce: a mapreduce implementation on top of a cloud operating system," AccentureTechnology Labs, Tech. Rep., 2009.

[7]. B.Thirmala Rao, N.V.Sridevei, V. Krishna Reddy, LSS.Reddy. Performance Issues of Heterogeneous Hadoop Clusters in Cloud Computing. Global Journal Computer Science & Technology Vol. 11, no. 8, May 2011,pp.81-87.

[8] Apache Hadoop. http://hadoop.apache.org.

[9] C. Jin, C. Vecchiola and R. Buyya. MRPGA: An Extension of MapReduce for Parallelizing Genetic Algorithms. In Proc. of 4th International Conference on e-Science, 2008

[10] S. Bardhan and D.A.Menasce, ´Queuing Network Models to Predict the Completion Time of the Map Phase of MapReduce Jobs, CMG Intl. Conf., LasVegas, NV, Dec. 3-7, 2012

[11] Herodotos Herodotou, Fei Dong, and Shivnath Babu, No one (cluster) size fits all: automatic cluster sizing for data-intensive analytics, Proc.2nd ACM Symposium on Cloud Computing, ACM,2011.

[12] Xuan Wang,Clustering in the cloud:Clustering Algorithms to Hadoop Map/Reduce Framework" (2010),Published by Technical Reports-Computer Science by Texas State University.

[13] Huan Liu and Dan Orban, Cloud MapReduce: a MapReduce Implementation on top of a Cloud Operating System. Published in Cluster, Cloud and Grid Computing(CCGrid) 2011,11th IEEE/ACM International Symposium.

[14] S. Vijayakumar, A. Bhargavi, U. Praseeda, and S. Ahamed, “Optimizing sequence alignment in cloud using hadoop and mpp database,” in CloudComputing (CLOUD), 2012 IEEE 5th International Conference on, june 2012, pp. 819–827.

[15] Mastering in cloud computing text book by Rajkumar Buyya publisher :Morgan Kaufmann may 2013