IJISET - International Journal of Innovative Science, Engineering & Technology, Vol. 3 Issue 10, October 2016
ISSN (Online) 2348 – 7968 | Impact Factor (2015) - 4.332
www.ijiset.com

Analyzing The Hive Performance on Large Data

A. Prathima[1], Y. Rama Mohan[2]

[1] PG Student, Computer Science and Engineering Dept, GPREC,
Kurnool (District), Andhra Pradesh-518007, INDIA. alluriprathima36@gmail.com

[2] Guide: Asst.Prof. in Computer Science and Engineering Dept, GPREC,
Kurnool (District), Andhra Pradesh-518007, INDIA. yrmgprec@gmail.com

Abstract

This paper presents an implementation of a scientific data management benchmark, on Hive, a MapReduce-based data warehouse. A complete strategy of migrating HDFS to Hive is described in detail including query HQL implementation, data partition schema and adjustments of underlying storage facilities. It have tuned the performance using several system parameters provided by Hive, Hadoop and HDFS. This paper provides preliminary results and analysis.. The purpose of this project was to compare the scalability of open-source relational database management systems and distributed data management systems for small and medium data sets. To make this comparison, a business intelligence case study was investigated using three data management solutions: MySQL, Hadoop MapReduce, and Hive. This experiment involved a payment history analysis which considers customer, account, and transaction data for predictive analytics. Experiments were executed on data sets ranging from 200MB to 800MB. The results show that the single server MySQL solution performs best for trial sizes ranging from 200MB to 800MB, but does not scale well beyond that. MapReduce outperforms MySQL on data sets larger than 800MB and Hive outperforms MySQL on sets larger than 2GB. This demonstrates MapReduce and Hive as viable techniques for small and medium businesses who want to implement scalable data management techniques.

Keywords: *Big Data, Hadoop,Hive,HDFS,SQL,Map-reduce*

1. Introduction

Big data and Hadoop

From the early days of the Internet's mainstream breakout, the major search engines and ecommerce companies wrestled with ever-growing quantities of data. More recently, social networking sites experienced the same problem. Today, many organizations realize that the data they gather is a valuable resource for understanding their customers, the performance of their business in the marketplace, and the effectiveness of their infrastructure.

The Hadoop ecosystem emerged as a cost-effective way of working with such large datasets. It imposes a particular programming model, called MapReduce, for breaking up computation tasks into units that can be distributed around a cluster of commodity, server class hardware, thereby providing cost-effective, horizontal scalability. Underneath this computation model is a distributed file system called the Hadoop Distributed File System (HDFS). Although the file system is "pluggable," there are now several commercial and open source alternatives.

However, a challenge remains; how do you move an existing data infrastructure to Hadoop, when that infrastructure is based on traditional relational databases and the Structured Query Language (SQL)? What about the large base of SQL users, both expert database designers and administrators, as well as casual users who use SQL to extract information from their data warehouses? This is where Hive comes in. Hive provides an SQL dialect, called Hive Query Language (abbreviated HiveQL or just HQL) for querying data stored in a Hadoop cluster.

Hive is most suited for data warehouse applications, where relatively static data is analyzed, fast

response times are not required, and when the data is not changing rapidly. Hive is not a full database. The design constraints and limitations of Hadoop and HDFS impose limits on what Hive can do. The biggest limitation is that Hive does not provide record-level update, insert, nor delete. You can generate new tables from queries or output query results to files. Also, because Hadoop is a batch-oriented system, Hive queries have higher latency, due to the start-up overhead for MapReduce jobs. Queries that would finish in seconds for a traditional database take longer for Hive, even for relatively small data sets. Finally, Hive does not provide transactions.

Hive

As the MapReduce framework provides measurability and low-level flexibility to run complicated jobs on huge data sets, it may acquire hundreds of minutes or even more than one day to execute a single MapReduce job. Considering this, Facebook built up Hive which is based on similar concepts of tables and partitions allowing a high-level query tool for getting data from their living Hadoop warehouses. The result is, on the top of Hadoop a data warehouse layer is built which permits for querying and handling structured data using a HiveQL, Hive query language that is similar to SQL (Structured query language) and optional custom MapReduce scripts which may be added into queries. During the compilation process, Hive converts Hive Query Language transformations to a series of MapReduce jobs and Hadoop Distributed File System operations and applies several optimizations.

The Hive data model is formed into tables, buckets and partitions. The tables in Hive data model are same as that of RDBMS tables. They correspond to an HDFS directory. Each table can be divided into partitions that can be further divided into buckets. Partitions correspond to sub-directories within an HDFS table directory and buckets are stored as files within the HDFS directories. It is important to remember that Hive was designed for batch job handling, extensibility and scalability but not for real-time queries and low latency performance. Hive query response times for smallest jobs and larger jobs upto some extent can be of several minutes.

Hadoop HDFS stores records on its own way, under flat files in key value pair. Hive has an interface above Hadoop HDFS which would allow users to query Hadoop by using a language HiveQL which is familiar to SQL. Hive is a data warehouse system for Hadoop. It provides comfortable data summarization, ad-hoc queries, and the analysis of huge datasets laid in Hadoop compatible file systems. Using HiveQL we query the data which has been structured by a mechanism provided by Hive. HiveQL is the query language used by Hive, to carry out all the tasks. Hive is similar to SQL and prepares data into four major sections. Databases and tables are familiar and Partitions and Buckets are dissimilar. During partitions each table are made to distinguish with one or more partition keys, deciding how data is stored. Whereas the rows are efficiently recognized when certain standards are equalled. Based on the hash value obtained of the column of a table, partitioned data is further divided into buckets,. It is also important to remember that, there is no need of data in the tables to be partitioned or bucketed, but doing so spreads the large quantity of data across the nodes, in turn helping in faster query execution.

2.Proposed System

2.1 Existing System

The data-intensive scientific discoveries are generating huge amounts of data at an alarming rate. Most of the data are multidimensional and stored in array-based file formats. The processing of such big data becomes an urgent challenge. In this paper, we present Hive, a scalable

and easy-to-use array-based query system. Hive enables scientists to process raw array datasets in parallel with a SQL-like query language. This implement Hive as an extension of Hive which is a data warehouse system on Hadoop. Hive maps the arrays in files to a table and executes the queries via MapReduce. Files are loaded dynamically as needed. So Hive does not need any additional pre-loading or format conversion procedure. In addition, Hive includes two optimization methods to reduce the generated rows. Experiments with different queries on representative datasets show that the optimizations are very effective in most cases and Hive is scalable to handle large datasets.

2.2 Proposed System

Hive is an open-source Hadoop-based data warehouse framework. With series of tools that Hive provides, structured data files can be mapped to database tables. Hive also defines a SQL-like language called Hive Query Language, which is compatible to SQL. HQL statements can be translated into MapReduce tasks to run on multiple nodes. Data in Hive can be stored in three formats, text, sequence binary and RCFile. Text is the default data store format of Hive. It is often used in log processing and data loading. The drawback of text format is that the costs for parsing are much higher than using the binary format and data should be parsed repeatedly for each time of query execution. Sequence file is a binary format provided by Hadoop[15] and also supported in Hive. Sequence file uses standard writable interface to implement serialization and deserialization. Data are stored as <key, value> pairs in sequence file. RCFile, Record Columnar File, is designed for storing relational tables on clusters using the MapReduce framework. It divides table data first horizontally and then vertically. Table records are divided into row groups. The result will be displayed in table format. After processing the data in sql and hive compare them.

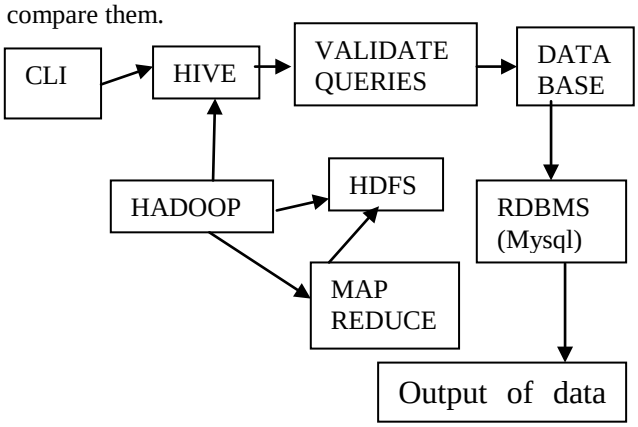

Fig:2.2 Data flow diagram of proposed system

3. Methodology

3.1 Hive Architecture and functionality

Hive is a data warehousing infrastructure placed on Hadoop. On commodity hardware a massive scale out and fault tolerance capabilities for data storage and processing is provided by hadoop by using the map-reduce programming paradigm.

Data summarization, ad-hoc querying and analysis of large volumes of data can be easily enabled by designing Hive. A simple query language called Hive QL, which is based on SQL is provided by Hive. Users familiar with SQL can easily adapt to this Hive query language. To do more sophisticated analysis which are not supported by built-in-capabilities of the language HiveQL allows traditional map/reduce programmers to plug in their custom mappers and reducers.

Hive architecture explains the major components of hive and its interactions with Hadoop which is shown in Figure 3.1 below:

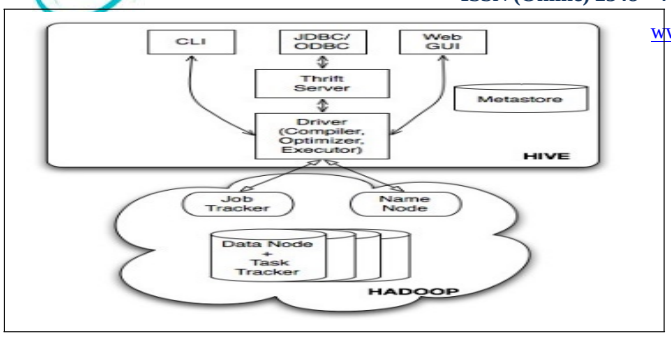

Fig:3.1:HiveArchitecture

3.1.1. Meta store

The metastore is the system catalog which contains meta-data about the tables stored in Hive. This metadata is specified during table creation and reused every time the table is referenced in HiveQL. The metastore distinguishes Hive as a traditional warehousing solution (ala Oracle or DB2) when compared with similar data processing systems built on top of map-reduce like architectures like Pig and Scope.

The metastore contains the following objects:

Database

It is a name space for tables. The database 'default' is used for tables with no user supplied database name.

Table

Metadata for table contains list of columns and their types, owner, storage and SerDe information. It can also contain any user supplied key and value data; this facility can be used to store table statistics in the future. Storage information includes location of the table's data in the underlying filesystem, data formats and bucketing information. SerDe metadata includes the implementation class of serializer and deserializer methods and any supporting information required by that implementation. All this information can be provide during the creation of table.

Partition

Each partition can have its own columns and SerDe and storage information. This can be used in the future to support schema evolution in a Hive warehouse. The storage system for the metastore should be optimized for online transactions with random accesses and updates. A file system like HDFS is not suited since it is optimized for sequential scans and not for random access. So, the metastore uses either a traditional relational database (like MySQL, Oracle) or file system (like local, NFS, AFS) and not HDFS. As a result, HiveQL statements which only access metadata objects are executed with very low latency. How-ever, Hive has to explicitly maintain consistency between metadata and data.

3.2. Compiler

The driver invokes the compiler with the HiveQL string which can be one of DDL, DML or query statements. The compiler converts the string to a plan. The plan consists only of metadata operations incase of DDL statements, and HDFS operations incase of LOAD statements. For insert statements and queries, the plan consists of a directed-acyclic graph(DAG)ofmap-reducejobs.

3.2.1. Query Processor:

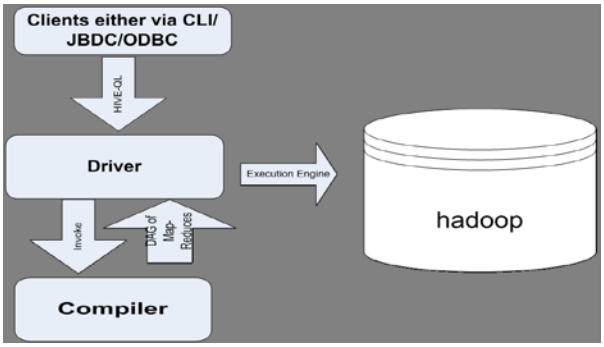

Fig:3.2.1.Execution flow of Hive Architecture

IJISET - International Journal of Innovative Science, Engineering & Technology, Vol. 3 Issue 10, October 2016
ISSN (Online) 2348 – 7968 | Impact Factor (2015) - 4.332
www.ijiset.com

Figure 3.2.1. shows how a typical query flows through the system. The UI calls the execute interface to the Driver The Driver creates a session handle for the query and sends the query to the compiler to generate an execution plan(step 2). The compiler gets the necessary metadata from the metastore (steps 3 and 4). This metadata is used to typecheck the expressions in the query tree as well as to prune partitions based on query predicates. The plan generated by the compiler is a DAG of stages with each stage being either a map/reduce job, a metadata operation or an operations on hdfs. For map/reduce stages, the plan contains map operator trees(operator trees that are executed on the mappers) and a reduce operator tree(for operations that need reducers). The execution engines submits these stages to appropriate components. In each task(mapper/reducer) the deserializer associated with the table or intermediate outputs is used to read the rows from hdfs files and these are passed through the associated operator tree. Once the output is generated, it is written to a temporary hdfs file though the serializer(this happens in the mapper in case the operation does not need a reduce). The temporary files are used to provide data to subsequent map/reduce stages of the plan. For DML operations the final temporary file is moved to the table's location. This scheme is used to ensure that dirty data is not read(file rename being an atomic operation in hdfs). For queries, the contents of the temporary file are read by the execution engine directly from hdfs as part of the fetch call from the Driver.

4. Implements and Results

4.1. Implementation of MySql Connector

In MySql only small amount of data is processed without any delay in fraction of seconds. But as the data is being increased in day by day life, it became difficult to process large amount of data with high performance by MySql.

For starting the MySql connector in hadoop command can be given as

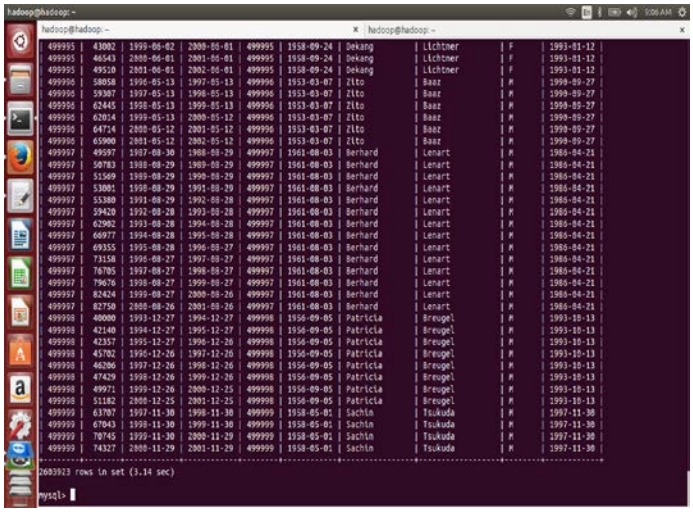

Fig: 4.1. Data running on mysql

4.2. Creation of Databases:

The Hive concept of a database is essentially just a catalog or namespaceof tables. However, they are very useful for larger clusters with multiple teams and users, as a way of avoiding table name collisions. It's also common to use databases to organize production tables into logical groups. If you don't specify a database, the default database is used.

The simplest syntax for creating a database is shown in the following example; here we are taking a payment history analysis of a company in that we are using the database is payment:

If u want to see in which database u are in then we can use these commands:

hive>set hive.cli.print.current.db = true;

5. Joins:

MapReduce can perform joins between large datasets, but writing the code to do join

From scratch is fairly involved. Rather than writing MapReduce programs, you might consider using a higher-level framework such as Pig, Hive, or Cascading, in which join operations are a core part of the implementation.

Let's briefly consider the problem we are trying to solve. We have two datasets; for

Example, the weather stations database and the weather records—and we want to reconcile the two. For example, we want to see each station's history, with the station's metadata inlined in each output row.

How we implement the join depends on how large the datasets are and how they are partitioned. If one dataset is large (the weather records) but the other one is small enough to be distributed to each node in the cluster (as the station metadata is), then the join can be effected by a MapReduce job that brings the records for each station together (a partial sort on station ID, for example). The mapper or reducer uses the smaller dataset to look up the station metadata for a station ID, so it can be written out with each record.

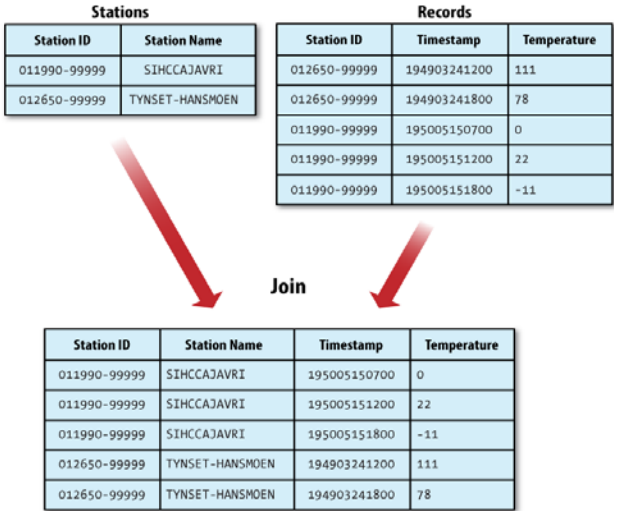

Fig: 5.0: An Example for joining station records

5.1. Partitioning Hive Tables:

Hive is a powerful tool to perform queries on large data sets and it is particularly good at queries that require full table scans. Yet many queries run on Hive have filtering where clauses limiting the data to be retrieved and processed, e.g. SELECT * WHERE state='CA'. Hive users tend to have or develop a domain knowledge, understand the data they work with and the queries commonly executed or scheduled. With this knowledge we can identify common data structures that surface in queries. This enables us to identify columns with a (relatively) low cardinality like geographies or dates and high relevance to key queries. For example, common approaches to slice the airline data may be by origin state for reporting purposes. We can utilize this knowledge to organise our data by this information and tell Hive about it. Hive can utilize this knowledge to exclude data from queries before even reading it. Hive tables are linked to directories on HDFS or S3 with files in them interpreted by the meta data stored with Hive. Without partitioning Hive reads all the data in the directory and applies the query filters on it. This is slow and expensive since all data has to be read. In our example a common reports and queries might be generated on an origin state basis. This enables us to define at creation time of the table the state column to be a partition. Consequently, when we write data to the table the data will be written in sub-directories named by state (abbreviations). Subsequently, queries filtering by origin state, e.g. SELECT * FROM Airline_Bookings_All WHERE origin_state = 'CA', allow Hive to skip all but the relevant sub-directories and data files. This can lead to tremendous reduction in data required to read and filter in the initial map stage. This reduces the number of mappers, IO operations. [11]

5.2. Comparison of Hive and SQL:

Apache Hadoop MapReduce[20] is a framework for processing large data sets in parallel across a Hadoop cluster. Data analysis uses a two-step map and reduce process. The job configuration supplies map and reduce

analysis functions and the Hadoop framework provides the scheduling, distribution, and parallelization services.

The top level unit of work in MapReduce is a job. A job usually has a map and a reduce phase, though the reduce phase can be omitted. For example, consider a MapReduce job that counts the number of times each word is used across a set of documents. The map phase counts the words in each document, then the reduce phase aggregates the per-document data into word counts spanning the entire collection.

During the map phase, the input data is divided into input splits for analysis by map tasks running in parallel across the Hadoop cluster. By default, the MapReduce framework gets input data from the Hadoop Distributed File System (HDFS). Using the MarkLogic Connector for Hadoop enables the framework to get input data from a MarkLogic Server instance. For details, see Map Task.

The reduce phase uses results from map tasks as input to a set of parallel reduce tasks. The reduce tasks consolidate the data into final results. By default, the MapReduce framework stores results in HDFS. Although the reduce phase depends on output from the map phase, map and reduce processing is not necessarily sequential. That is, reduce tasks can begin as soon as any map task completes. It is not necessary for all map tasks to complete before any reduce task can begin.

MapReduce operates on key-value pairs. Conceptually, a MapReduce job takes a set of input key-value pairs and produces a set of output key-value pairs by passing the data through map and reduce functions. The map tasks produce an intermediate set of key-value pairs that the reduce tasks uses as input. The diagram below illustrates the progression from input key-value pairs to output key-value pairs at a high level:

Fig: 5.2. Joining the Data in Hive Query Lang.

Example Hive table partitioning It is important to consider the cardinality of a potential partition column and avoid fragmenting the data too much. Itinerary ID would be a very poor choice for partitioning. Queries for single itineraries by ID would be very fast but any other query would require to parse a huge amount of directories and files incurring serious overheads. Additionally, HDFS uses a very large block size of usually 64 MB or more which means that each file, even with only a few bytes of data, will have to allocate that block size on HDFS[3]. This can potentially fill the file system up with large number of files carrying barely any actual data.

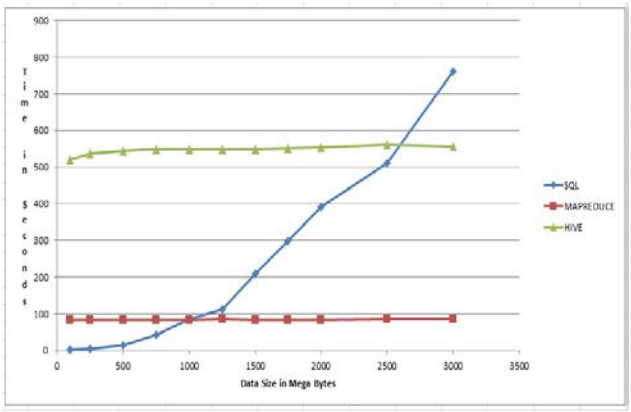

Fig:5.2.1. comparison of sql and hive query lang.

6. CONCLUSION

A Simple SQL query reads the entire dataset and performs the operations on the datasets which results in higher latency rate. This becomes a bottleneck for running operation on vast datasets. So as an alternative to this Hive is used, the efficiency of hive collating to SQL. HiveQL Analyze and Hive join operation the results of required Hive query are displayed in tabular form i.e structured format. It can be observed from the map-reduce program performance comparison between Hive and SQL that, Hive performance remained constant and better for all sizes of data, matching and surpassing MapReduce performance specially in case of larger data sets. Hence it is concluded that Hive is aviable technique for implementing in small and medium businesses who want to implement scalable data management techniques.

Future Direction:

Beyond the scope of this project, we can further compare Hive's performance with PIG or Spark technologies.

References

[1]Sachchidanand Singh,Nirmala Singh,"Big Data Analytics", International Conference on Communication, Information & Computing Technology (ICCICT),Oct. 19-20, Mumbai, India.

[2]A.Bialecki,M.Cafarella,D.Cutting,andO.OSMalley,"Hadoop: A Framework for Running Applications on Large Clusters Built of Commodity Hardware," Wiki at http://lcene. apache.Orglhop

[3] T. White, "The Hadoop Distributed Filesystem," Hadoop: The Definitive Guide, pp. 41-73, GravensteinHighwaNorth, Sebastopol: O'Reilly Media, Inc., 2010.

[4] https://hadoop.apache.org/

[5]J.shi,j.zou,j.lu,z.cao,s.li,and c.wang,"MRTuner:A toolkit to enable Optimization for MapReduce Jobs,"PVLDB,vol. 7,no. 13,pp.1319-1330,2014.

[6]https://cwiki.apache.org/confluence/display/Hive/LanguageManual.

[7]http://blog.safaribooksonline.com/2012/12/05/tip-using-joins-inhive/).

[8] Anja Gruenheid,Edward Omiecinski, Leo Mark, "Query Optimization Using Column Statistics in Hive", Procedings of the 15th Symposium on International Database Engineering & Applications,2013.

[9] http://hive.apache.org/

[10] https://www.mapr.com/products/apache-hadoop

[11] http://hive.apache.org/

[12]Herodotos Herodotou and Harold Lim and Gang Luo and Nedyalko Borisov and Liang Dong and Fatma Bil-gen Cetin and Shivnath Babu, "Starfish: A Self-tuning System for Big Data Analytics," in In CIDR, 2011, pp. 261–272.

[13]L. C. Abhishek Verma and R. H. Campbell, "ARIA: Automatic Resource Inference and Allocation for Mapreduce Environments",in Proceedings of the 8th ACM International Conference on autonomic.

[14].Ms.VibhavariChavan,Prof.Rajesh.N.Phursle,"survey paper on bigdata",(IJCSIT),VOL.5 ,2014.

[15].*"Welcome to Apache Hadoop". hadoop.apache.org*. Retrieved 2015-12-16.

[16].Serge Blazhievsky, "Introduction to Hadoop,Map Reduce and HDFS for Big Data Application". SNIA Education, 2013.

[17].Vishal S Patil, Pravin D. Soni, "hadoop skeleton & fault tolerance in hadoop clusters", International Journal of Application or Innovation in Engineering & Management (IJAIEM)Volume 2, Issue 2, February 2013 ISSN 2319 – 4847 [8] Sanjay Rathe.

[18].Harshavardhan S.Bhosale, prof.Devendra P.Gadekar,"A review paper on big data and Hadoop",(IJSRP)10 ,October 2014.

[19].Mahesh Maurya ,Sunita Mahajan,"performane analysis of MapReduce programs on Hadoop cluster, IEEE, 2012.

[20].Trong -Tuan Vu, fabric Huet," A light weight continuous jobs mechanism for MapReduce frameworks",IEEE,2013.

First Author: Prathima.A was born in Andhra Pradesh, India. She received the B.Tech Degree in Computer Science and Engineering from Jawaharlal Nehru Technological University Anantapur branch, India in 2014 and M.Tech Degree in also same branch and University. Her research interests are in the area of Semantic Web and Big Data Analytics.

Second Author : Y. Rama Mohan, M.Tech, was born in Andhra Pradesh, India. He is working as Asst.Prof. in Computer Science & Engineering Dept, GPREC Kurnool(district),A.P.518007,INDIA.His research interest s are in the area of big data analyst **.**