UNIVERSITY OF PIRAEUS – DEPARTMENT OF INFORMATICS

MASTER PROGRAM IN ADVANCED INFORMATION SYSTEMS

THESIS

The title of Thesis	**Hadoop Framework**
First and Surname Student	**Apostolis Plagakis**
Father's Name	**Panayiotis**
Registration Number	**ΜΠΣΠ/10044 .....**
SuperVisor	**Nick Pelekis, Lecturer**

Piraeus, Greece, October 2013

Three Member Examining Committee

(Signature)	(Signature)	(Signature)
Professor Yannis Theodoridis	Professor Ioannis Siskos	Lecturer Nick Pelekis

I would like to kindly thank my professor Nick Pelekis for the change he gave me to approach the newest strategies and trends that come from the highly innovative area of Big Data.

Also, I would like to think him for the exceptionally useful co-operation we had together and his understanding to myself.

Additionally, the monitoring with the Big Data area, the open-source databases and the shared-nothing architecture created knowledge in depth, experiences of new techniques, exceptionally useful to my professional career path and development.

Finally I would like to thank Mr. Theoridis and Mr. Siskos for the honor they did to be present in this committee.

Contents

Summary

With the explosion of data, new tools become available. One of the most popular tools is the open source Apache Hadoop Framework.

In this thesis, the MapReduce program model is examined, together with the Hadoop framework any open matters in the field of the databases that is called to cover.

We examined cases that the different architectures of databases systems overlay each other or each one of them is requested to fill in the open issues of the other architecture.

As well we are examining the Hadoop framework is responding in demands of structured data and more specifically in cases of geospatial values, although the Hadoop Framework is oriented to unstructured data of text type.

Furthermore, we are evaluating the advantages of Hadoop, which could be fully used by spatial system Hermes, which is created by the Information Dpt of Piraeus University.

The first scenario, of which operational features was examined is the co-operation of Hadoop with HadoopDB which is basically reflects the co-operation of a shared-nothing architecture with a system RDBMS (PostGIS). This solution was finally judged as non “usable”, because as explained below it fits to the characteristics of shared-nothing architecture, only at the minimum.

The second scenario, which is finally our proposition, is the access to spatial data, which are stored in a HDFS system by using HIVE and extended tools, that are provided by third parties and are customized in spatial querying.

Chapter 1 Introduction

1.1 Introduction

The trend that is being formed in the business world is the increased connection between decision making and information and not as a result of instinct and theory. As a result of this change, companies continually try to acquire and use systems which manage, process and analyses complex vast quantities of data. The increasing cost of the very demanding computer systems influences the production environment for the analytical data management applications which is changing at a rapid pace. Many businesses are moving from the development of databases in high cost and capacities owned servers to cheaper, common solutions which are organized into a partitioned architecture MPP (An architecture which uses multiple processors and different programming models), often in a virtual environment in the context of public or private “clouds”. At the same time, the volume of data that must be processed is spiraling and demands a large number of junction-servers working in parallel in order to analyse and process the data. The investors recognize the good prospects of this trend and fund initiatives and companies which develop relevant specialized software for data management (e.g. Netezza, Vertica, Datallegro, Greeenplum, AsterData, Infobright, KickFire, Dataupia, ParAccel) even in times of economic recession.

Two schools of thought are being formed regarding the technology which is going to be used for the data analysis in the specific working environments. The supporters of parallel databases(etc. Microsoft SQL Server PDW, Oracle Exadata) advocate placing strong emphasis – effort on the performance and efficiency of parallel databases which stems from the fact that an important part of the processing of a question is performed in the inner management mechanism of the database and creates all the conditions for successful analyzing – grouping of the primary data. On the other hand, others claim that MapReduce systems demonstrate better adaptation because of their great expansibility, error tolerance as well as their flexibility in handling unstructured data.

1.2 Big Data

In Information technology, big data is a collection of –usually unstructured- data sets so large, high-variety and complex that it becomes difficult to process using on-hand database management tools or traditional data processing applications. The challenges include capture, duration, storage, search, sharing, analysis and visualization. The trend to larger data sets is due to the additional information derivable from analysis of a single large set of related data, as compared to separate smaller sets with the same total amount of data, allowing correlations to be found to “spot business trends, determine quality of research, prevent diseases, combat crime, and determine real-time roadway traffic conditions.

As of 2012, limits on the size of data sets that are feasible to process in a reasonable amount of time were on the order of exabytes of data. Scientists regularly encounter limitations due to large data sets in many areas, including meteorology, genomics, connectomics , complex physics simulation, and biological and environment research. The limitations also affect Internet Search, finance and business informatics. Data sets grow in size in part because they are increasingly being gathered by ubiquitous information-sensing mobile devices, aerial sensory technologies, software logs, cameras, microphones, radiofrequency identification readers and wireless sensors networks.

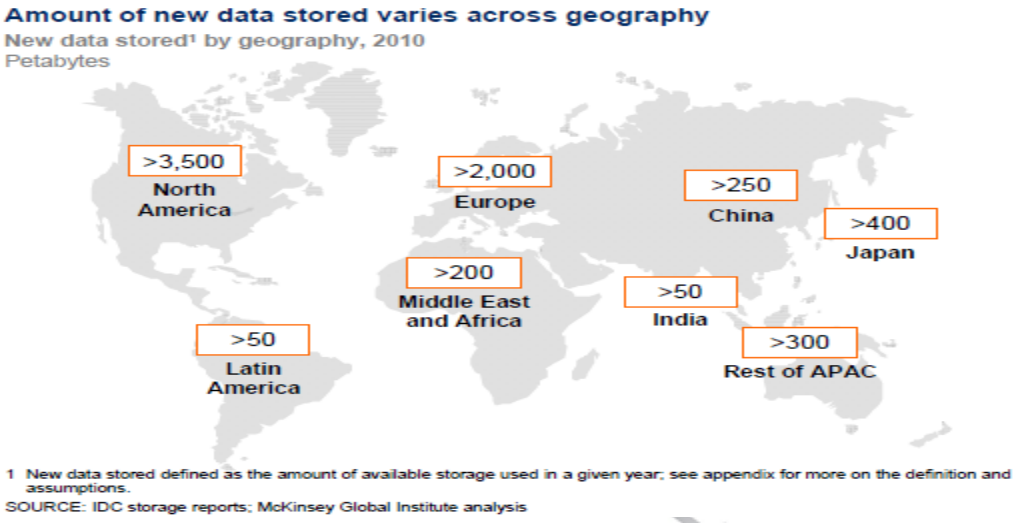
Big data is difficult to work with using relational databases and desktop statistics and visualization packages, requiring instead “massively parallel software running on tens , hundreds, or even thousands of servers.

In 2012, the Obama administration announced the Big Data Research and Development Initiative, which explored how big data could be used to address important problems facing the government. The initiative was composed of 84 different big data programs spread across six departments. Similar if the developed economies of Europe, government administrators could save more than €100 billion ($149 billion) in operational efficiency improvements alone by using big data to reduce fraud and errors and boost the collection of tax revenues.

Big data can generate significant financial value across sectors

SOURCE: McKinsey Global Institute analysis

There are five broad ways in which using big data can create value. First, big data can unlock significant value by making information transparent and usable at much higher frequency. Second, as organizations create and store more transactional data in digital form, they can collect more accurate and detailed performance information on everything from product inventories to sick days, and therefore expose variability and boost performance. Leading companies are using data collection and analysis to conduct controlled experiments to make better management decisions; other; others are using data for basic low-frequency forecasting to high-frequency nowcasting to adjust their business levels just in time. Third, big data allows ever-narrower segmentation of customers and therefore much more precisely tailored products on services. Fourth, sophisticated analytics can substantially improve decision-making. Finally, big data can be used to improve the development of the next generation of products and services. For instance, manufacturers are using data obtaining from sensors embedded in products to create innovative after-sales service offerings such as proactive maintenance (preventive measures that take place before a failure occurs or is even noticed).



To capture value from big data, organizations will have to deploy new technologies(e.g., storage, computing, and analytical software) and techniques(i.e, new types of analyses). The range of technology challenges and the priorities set for the tackling them will differ depending on the data maturity of the institution. Legacy systems and incompatible standards and formats too often prevent the integration of data and the more sophisticated analytics that create value from big data. New problems and growing computing power will spur the development of new analytical techniques. Sophisticated analytics can substantially improve decision making, minimize risks, and unearth valuable insights that would otherwise remain hidden. Big Data either provides the raw material needed to develop algorithms or for those algorithms to operate. There is also a need for ongoing innovation in technologies and techniques that will help individuals and organizations to integrate, analyze, visualize, and consume the growing torrent of big data.

CHAPTER 2 Map Reduce

2.1 What is Map Reduce

Map Reduce is a model of divided programming created in order to process a particularly large volume of data which are stored in different – many computer clusters. In the MapReduce model, the primary data which under processing are called mappers and reducers. The central idea of the Map Reduce model is to hide the details of parallel processing – performing and allocation of the tasks and to allow users to focus only on the data processing strategies. In 2004 the engineers of Google Jeffrey Dean and Sanjay Ghemawat published a study entitled “MapReduce: Simplified Data Processing on Large Clusters”. In this study the new programming model MapReduce appeared for the first time setting as its goal to process requests from large quantities of data and the distribution to a large number of junctions, fully utilizing the capabilities of parallel processing. It replaced the original algorithm of indexing and cataloguing in 2004, and the conclusions that were derived had to do with a proved effectiveness in the processing of substantial and unstructured data clusters. Despite the fact that MapReduce technology is much more recent than that of SQL, it is used by far more people due to its connection with Google. If we are looking for a specific term in the Google search engine, the processing of this request is based on the MapReduce technology (Fig. 2.1.).

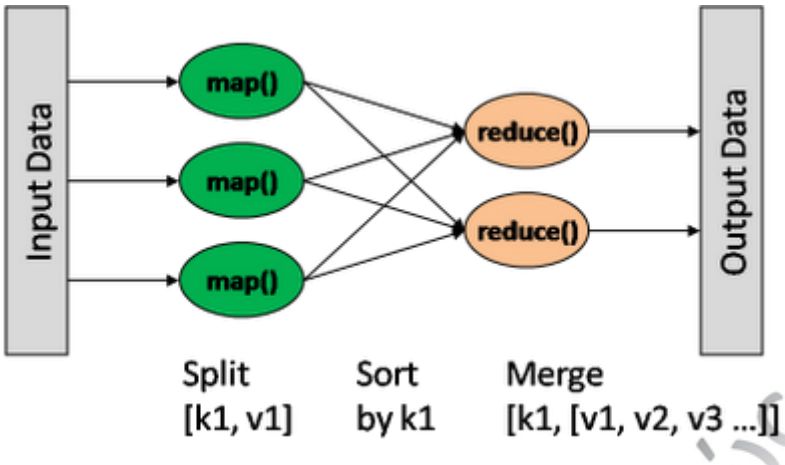


Figure 2.1.1 The Map/Reduce DAG. Source Hadoop Map/Reduce Implementation Ricky Ho

More specifically, it is used for the offline batch processing for the creation of search indices. Thus, when someone is searching in Google, the search is performed on these indices. Google and other corporations use this programming framework in order to maximize the capabilities of parallel processing, with the aim of always improving response times. Overall, the MapReduce methodology makes it possible to batch process a request, accessing the total of a huge set of data in reasonable time. Map Reduce is represented in the form of two processes (Fig. 1): the Map, which applies a procedure on all the members of a collection and produces a list which is the result of the processing, and the Reduce which contrasts and resolves the results of two or more Maps through multiple parallel tasks – threads – processors or autonomous systems. The MapReduce framework performs the tasks based on the “runtime scheduling scheme”. This means that before the performance MapReduce does not create any performance plan, which specifies which tasks will be activated and on which junctions. Whereas a traditional database system creates a tree-like performance plan for the query which is to be performed, in MapReduce this is directly specified during the performance phase. With programming during the functioning phase MapReduce achieves tolerance to errors through the process of error detection (frequent checks) and the reloading of the tasks of the junctions that presented an error to other healthy junctions of the cluster. Obviously the frequent demand for error detection (fault tolerance) influences the effectiveness of the specific methodology. The Map and Reduce functions can run in parallel in the same system at the same time. Several applications of Map Reduce are available in many programming languages like Java, C++, Python, Perl, Ruby and C et al. In some cases, as in the software development tools Lisp and Python, the Map Reduce has been incorporated in the framework itself of the programming language. In this case, the functions could be defined as follows:

```
List2 map(Functor, List1);

Object reduce(Functor2, List2);
```

The map function, when there is a case of particularly large sets of data, divides the data into two or more data blocks of smaller volume. The number of Map tasks does not depend on the number of junctions but on the number of inserted data blocks. Every collection of data contains loosely formed marked logical records or lines of a text. The complex results which derive from this process must behave like a collection of data which has been created by a simple map() function. The general formula of map() function is:

```
Map(function, list) {
        for each element in list {
        v = function(element)
        intermediateResult.add(v)
}
```

The reduce function works on one or more lists of results which are retrieved directly from the memory or the hard disk or from a network, are transferred and processed by all these procedures which run reduce functions. The general picture of the reduce function is"

```
Reduce(function, list, init) {
Result=init
Foreach value in list {
        Result-function(result,value_
}
OutputResult.add(result)
```

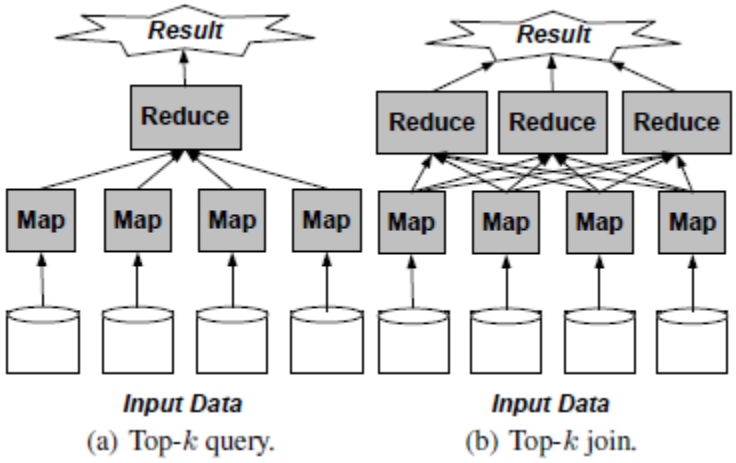


Figure 2.1.2 Map and Reduce tasks for top-k queries and joins

The MapReduce essentially follows a simple master – slave architecture. The master is a simple JobTrackers and the slave or work junctions are TaskTrackers. The JobTracker handles the planning of the MapReduce tasks and incorporates information in every new TaskTracker and in the sources available. Every task is broken down into Map sub-tasks, as the blocks of the data that require processing, and the Reduce processes.

The application of the MapReduce distinguishes the business logic from a multi-processing logic. The Map and Reduce functions run through multiple systems, are synchronized in partitioned channels and communicate with one another through Remote Procedure (RPC). The business logic is applied on functions defined by the user which interchange only with logical records and are not related to multiprocessing versions. These features allow for quick

completion of applications that are processed in parallel, which take place within a particularly large number of processors as the MapReduce constitutes a framework in which programmers develop functions (functors).

2.2 Examples of MapReduce

The MapReduce essentially follows a simple master – slave architecture. The master is a simple JobTrackers and the slave or work junctions are TaskTrackers. The JobTracker handles the planning of the MapReduce tasks and incorporates information in every new TaskTracker and in the sources available. Every task is broken down into Map sub-tasks, as the blocks of the data that require processing, and the Reduce processes.

The application of the MapReduce distinguishes the business logic from a multi-processing logic. The Map and Reduce functions run through multiple systems, are synchronized in partitioned channels and communicate with one another through Remote Procedure (RPC). The business logic is applied on functions defined by the user which interchange only with logical records and are not related to multiprocessing versions. These features allow for quick completion of applications that are processed in parallel, which take place within a particularly large number of processors as the MapReduce constitutes a framework in which programmers develop functions (functors).

The Map Reduce is asked to support large clusters of autonomous systems which process large volume databases in parallel. The parallel function offers considerable possibilities of retrieval in cases of fault in certain clusters of servers or storage systems: if a mapper or reducer function fails, the process can be promoted for processing to another system as long as the input data is available. Figure 2.1 depicts a central application running on a master system which is coordinated with other instances, which performs map or reduce functions and classifies the results that derive from each reduce procedure.

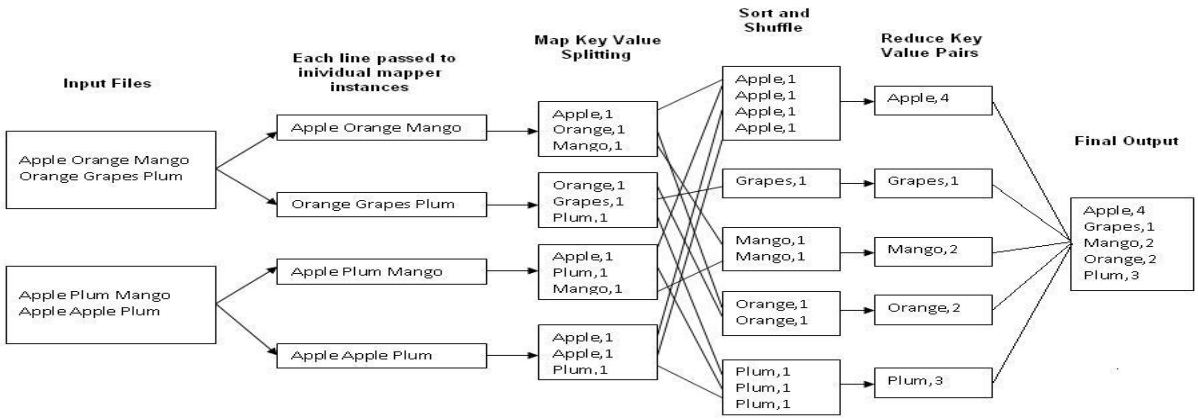


Figure 2.2.1 Java-Based MapReduce Implementation. Eugene Ciurana

Now coming to the practical side of implementation we need our input file and map reduce program jar to do the process job. In a common map reduce process two methods do the key job namely the map and reduce , the main method would trigger the map and reduce methods. For convenience and readability it is better to include the map , reduce and main methods in 3 different class files . We'd look at the 3 files we require to perform the word count job

Word Count Mapper

```
import java.io.IOException;

import java.util.StringTokenizer;

import org.apache.hadoop.io.*;

import org.apache.hadoop.mapred.*;13

public class WordCountMapper extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable>

{

    //hadoop supported data types

    private final static IntWritable one = new IntWritable(1);

    private Text word = new Text();

   //map method that performs the tokenizer job and framing the initial key value pairs

    public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException

    {

        //taking one line at a time and tokenizing the same

        String line = value.toString();

       StringTokenizer tokenizer = new StringTokenizer(line);

      //iterating through all the words available in that line and forming the key value pair

        while (tokenizer.hasMoreTokens())

        {

          word.set(tokenizer.nextToken());

       //sending to output collector which inturn passes the same to reducer

            output.collect(word, one);

        }

    }

}
```

The first and second parameter refers to the Data type of the input Key and Value to the mapper. The third parameter is the output collector which does the job of taking the output data either from the mapper or reducer, with the output collector we need to specify the Data Types of the output Key and Value from the mapper. The fourth parameter, the reporter is used to report the task status internally in Hadoop environment to avoid time outs.

Word Count Reducer

```
import java.io.IOException;
import java.util.Iterator;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
public class WordCountReducer extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable>
{
    //reduce method accepts the Key Value pairs from mappers, do the aggregation based on keys and produce the final out put
     public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException
    {
        int sum = 0;
        /*iterates through all the values available with a key and add them together and give the
        final result as the key and sum of its values*/
       while (values.hasNext())
       {
           sum += values.next().get();
       }
       output.collect(key, new IntWritable(sum));
    }
}
```

Here like for the mapper the reducer implements Reducer<Text, IntWritable, Text, IntWritable>

The first two refers to data type of Input Key and Value to the reducer and the last two refers to data type of output key and value. Our mapper emits output as <apple,1> , <grapes,1> , <apple,1> etc. This is the input for reducer so here the data types of key and value in java would be String and int, the equivalent in Hadoop would be Text and IntWritable. Also we get the output as<word, no of occurrences> so the data type of output Key Value would be <Text, IntWritable>

In Figure 2.2 the main application is responsible for the dividing of the data collection to distinguishable buckets-requests.

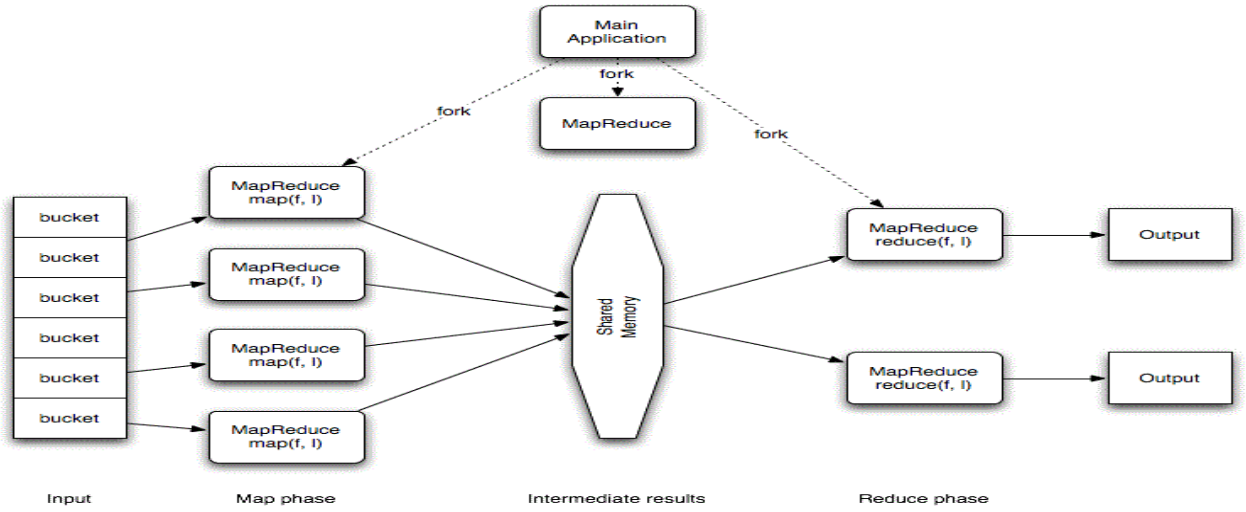


Figure 2.2.2 Map Reduce Application. Eugene Ciurana

The optimum size of the bucket depends on the application, the number of junctions and the bandwidth available. The buckets are stored on the hard disk but also in the memory if it is judged by the application that this is the best choice. The central application is also responsible for the creation or the fork or multiple copies of the Map Reduce core, which are identical except for the controller which allocates and coordinates map and reduce task junctions in inactive procedures or threads. The controller stores information for every map and reduce work junction as well as its state (waiting, performance, completion) and it might act as a conductor for the intermediate outputs between the map() and reduce tasks. The Map and Reduce functions are both defined with respect to data structured in (key, value) pairs. The Map function receives a pair of data in a certain sector and returns a list of pairs in a different sector, which continues to be performed for every call:

Map(k1, v1) → list(k2, v2)

Then the MapReduce collects all the pairs with the same key from all the lists creating distinct groups for each one of the distinct created keys. The Reduce processes each group by following the parallel model of operation, and produces a set of values in the same sector:

Reduce(k2, list (v2)) → list(v3)

The returned values for all the calls constitute the desired output list. Every processing procedure of the map work junction produces a bucket into which a list of intermediate outputs which are stored in portioned out areas (memory, hard disk, cache, etc.) is created. The task in progress informs the controller each time an intermediate output has been stored in a portioned out storage space.

The controller allocates reduce() tasks every time new intermediate outputs are available. The work junction classifies the outputs on the basis of the intermediate keys defined by the application with the introduction of comparators which group identical data with the aim of rapidly retrieving information. Then the work junction links the classified data to a single key and assigns them for processing in the user defined reduce() functions.

The processing from the map and reduce junctions is completed when all the buckets have been exhausted and all the reduce work junctions have informed the controller that the outputs to be exported have been produced. The controller informs the application in order to receive

the results of the tasks. The central application can use these data, but it can also assign them to a different MapReduce controller for further processing.

During the last years we have a true explosion of data production. IDC organization estimates the volume of data in the digital university in 2011 to 0.18 zettabytes. A zettabyte is a million terabytes. This huge amount of data comes from various sources. Some of them are:

- The New York stock market (Wall Street) produces about one terabyte of new data every day.
- The social site Facebook hosts 10 million photographs which require at least one petabyte for their storage.
- The genealogical site stores about 2.5 petabytes of data.
- The Internet Archive stores store information whose total volume reaches 2 petabytes while the monthly increase of the volume of data approaches 20 terabytes.
- Large Hadron Collider in Geneva needs 20 terabytes each month in order to store the information produced.

Therefore it is understood that there is a huge amount of information – data that are produced daily but also an increasing amount of information that is accessed by ordinary users. IDC company estimates that the amount of information that is stored in the "Digital Universe" cyberspace (a non-profit organization which collects and presents on the Internet important educational, scientific and cultural information) in 2011 is 0.18 zettabytes. One zettabyte is one million terabytes. MapReduce can handle this huge amount of data through dividing the processing into many computers, which can operate in parallel, and of course to improve efficiency times by increasing in each moment the number of processors, the cost of which is relatively low today.

2.3 MapReduce Advantages

The operation model of Google is based on the development of MapReduce applications in large clusters of widely spread systems. Each system has its own storage system which is necessary for the processing of the corresponding bucket, a reasonable memory size (2 to 4 GB RAM) and an at least two-core processor.

The MapReduce offers a simple, clever solution for the processing of data in parallel systems which has advantage such as:

- Reduced cost
- Flexibility because the MapReduce is not bound by any data model or shape. A programmer can handle non regular or unstructured data in a considerably easier way in comparison to that of the traditional DBMS.
- Overall better performance and results in comparison with traditional RDBS techniques or specialized algorithms/heuristics.
- High productivity for the programmers since the business logic is implemented regardless of the parallelization code.
- Ease of installation because it uses familiar techniques and tools which are compatible with the Java architecture.

- Error tolerance. MapReduce exhibits high responsiveness to error tolerance. For example, it has been recorded that MapReduce can continue operating despite the presence of 1.2 failures on average per analysis job at Google[44.38].
- High escalation. The main reason for choosing MapReduce as a model of parallel operation is the high escalation it exhibits. Yahoo reports that its application Hadoop can utilize more than 4,000 junctions. (2008 report)
- Independence in comparison with underlying storage layers. Thus MapReduce can work with different storage layers like BigTable etc.

2.4 MapReduce Disadvantages

The MapReduce is not suitable choice:

- Real-time processing.
- It's not always very easy to implement each and everything as a MR program.
- When an intermediate processes need to talk to each other(jobs run in isolation)
- When a processing requires lot of data to be shuffled over the network
- When it is need to handle streaming data. The MapReduce is best suited to batch process huge amounts of data which you already have with you.
- When we have OLTP needs. MR is not suitable for a large number of short on-line transactions.

Besides some essential drawbacks of MR are:

- No high level language.
- No schema and no index
- A single fixed dataflow
- Low efficiency.

2.5 MapReduce vs SQL

A request that is activated in most cases is the calling of a function. Those map functions are developed by programmers and can be as complicated as their creators wish. The callings to these functions are distributed in the highest number of junctions(Figure 2.4.1). There is great similarity between the breaking up of a traditional SQL query into many tasks and the distribution of calls to the highest possible number of junctions. On the SQL programmer's part, the whole framework is realized as a set of external functions which can be called by the T-SQL.Some of these functions are incorporated and the programmers can develop corresponding functions adapted to specialized needs on a case-by-case basis. Thus software engineers do not have to learn a new programming language. What they should know is what the parameters mean and how a matrix function is to be called. Let us use an example in order to show what MapReduce means for the programmers: The query of a user who is requesting all the flights to London and at the same time all the flights departing from London on the same day and within the next hour in T-SQL is:

```
Select * From Departures as D1
Where Destination='LONDON'
AND Dep_Time+60 Minutes>=
(Select Min (Dep_Time)
```

```
From Departures As D2
Where Destination='London'
AND D2.Dep_Time>D1.Dep_Time
AND D2.Dep_Day=D1.Dep_Day)
        Order by Dep_time
```

and in the MapReduce function:

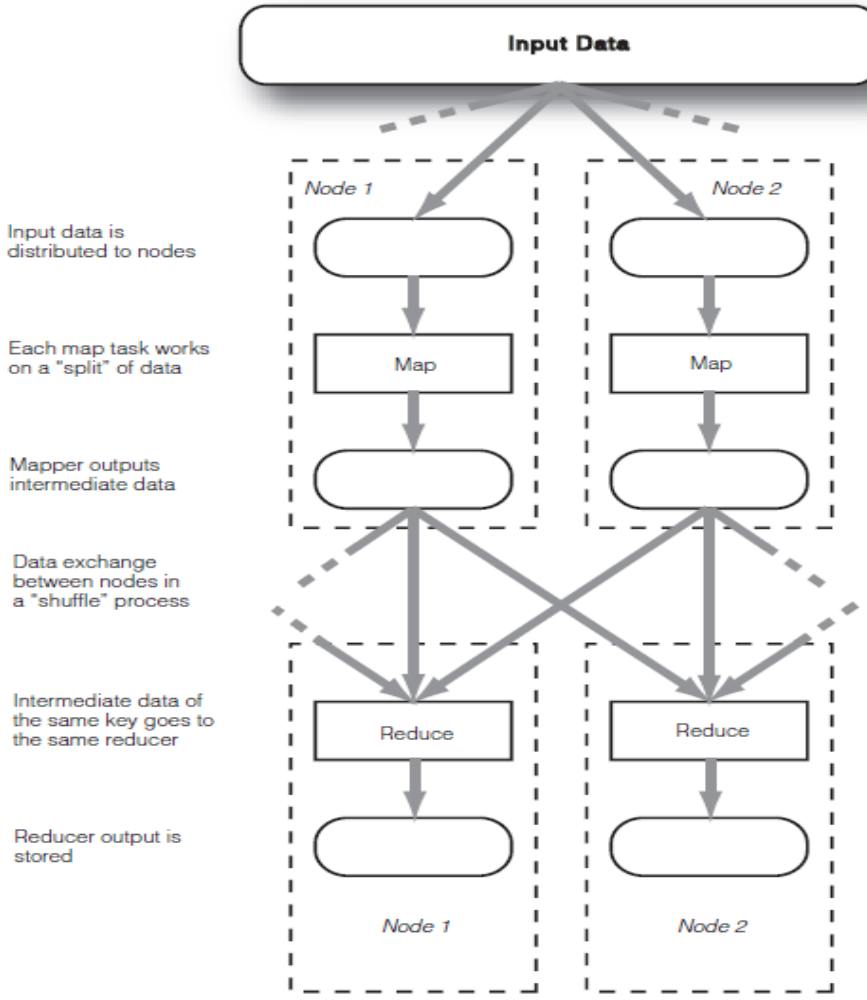
```
SELECT * FROM GET_NEXT_FLIGHT_1HR
(ON DEPARTURES PARTITION BY DESTINATION)
WHERE DESTINATION = 'London'
ORDER BY DEPARTURE_TIME
```

Apparently, the query in its second form is quite simpler in formulation compared to the original. The From call includes the call in the MapReduce function Get_Next_Flight_1Hr. This function has two parameters. The first specifies the data table which is to be asked (DEPARTURE) and the second one specifies the column in which the grouping of entries will be done (DESTINATION).

Next let us illustrate the work of the Map function with a simple example.

We assume that we want to return the results of the function GET_TOTAL_SALES_PER_STORE in a data table which contains sales movements. For each sales movement, the customer's, the store's and the product's id as well as the selling price and date are stored. So for each product that was bought by a customer, an entry is stored in the data table. The data table is distributed in all the junctions. Also let us assume that the input parameter of the function is the MIN_AMOUNT, which means that in the final result only those entries whose amounts are higher than the MIN_AMOUNT value will be included. The output of this function could be a set of entries which includes the total of the sold quantities for each store. In the Map function phase this call is distributed in the highest desired number of junctions.

During the Reduce procedure, all the calls are performed (possibly hundreds in parallel processing) and the intermediate outputs return to the controller (Master(s)). This step is called Reduce because a reduced number of entries will return to the controller. As in the case of parallel data bases, the aim is to minimize the volume of data which return to the controller. The rationale of the Reduce function depends on the programmer who has the ability to use the best possible tools of any programming language so that the function will have highest possible efficiency. Even the way in which they are going to be accessed by the data table is defined by the reduce function. So the code which composes the function is obviously non binding and the conditions and speed of storage depend on the way it is created. At the end of the procedure of the reduce function, the controller combines all the intermediate results into a final output: a set of entries.



Picture 2.6.1. Data flow based on the MapReduce model. The "Shuffle" procedure improves expansibility.

2.6 Map Reduce vs OLAP

There is a trend to treat MapReduce as a rival to other existing technologies. In reality it does not constitute a comprehensive solution to every problem. It constitutes an approach attempt to distributed tasks which are performed in many systems. Also it is not right to compare MapReduce with traditional OLAP technologies. On the contrary, it is possible to use MapReduce as part of an OLAP application. For example, one of the common usages of MapReduce is the indexing of unstructured data (like web logs which are text files containing information about every page that has been visited by millions of users) which are not easy to organize into a form. By putting data of this form into databases it is particularly difficult to handle them, because even a relatively small site with about one million page presentations per day, creates more than 100 GB of complex unstructured data per year. MapReduce and Hadoop can approach these tasks effectively using a network of computers. The superiority point has to do with the fact that it is not limited to the restricted capacity boundaries than one computer offers. Regardless of the amount of money that is going to be invested in a computer, many smaller computers will cost less and have greater capabilities. In a sense MapReduce may be regarded as a substitute of traditional databases. Instead of using a normal database,

which requires great effort of designing and maintaining, all the information are simply stored in simple text files and are retrieved when there is a relevant request. For example, for analytic applications MapReduce can handle massive data with smaller memory requirements.

2.7 MapReduce – Spatial Data

Geospatial data or geographic information it is the data or information that identifies the geographic location of features and boundaries on Earth, such as natural or constructed features, oceans and more. Spatial data is usually stored as coordinates and topology, and is data that can be mapped. Spatial data is often accessed, manipulated or analyzed through Geographic Information Systems (GIS).

A spatial database is a database that is optimized to store and query that represents objects defined in a geometric space. Most spatial databases allow representing simple geometric objects such as points, lines and polygons. While typical databases are designed to manage various numeric and character types of data, additional functionality needs to be added for databases to process spatial data types efficiently.

Today, many computer applications, directly or indirectly, are based on carrying out spatial analysis at the back-end. MapReduce technology has attracted researchers as an alternative solution to parallel DBMS to study the parallelization of spatial operations and their performance evaluation in distributed framework. MapReduce programming model is biased towards distributed processing of spatial data. MapReduce has the power to efficiently map most spatial query logic to Map and Reduce functions. For straight-forward spatial queries limited to WHERE clause and without joins, MapReduce paradigm require to have only map function. Input tuples read are tested if they qualify the Where criteria in the map-phase itself. For spatial queries involving spatial joins between two data sets, in addition to Map phase, Reduce phase is also required. General, spatial joins are performed on spatial objects that are spatial proximal. Map phase read the input tuples of join operand data sets, and create set of groups, each group contains only the set of spatial objects that are within pre-defined spatial boundaries. Each group is then processed for spatial join in parallel by reduce processes on cluster machines. Thus Vector spatial data , by its nature, is well suited to be processed on clusters following shared-nothing architecture. Figures 2.8.1 shows the MapReduce formulation of spatial join between two heterogeneous data sets: Rivers (Linestrings) and Settlements (Polygons). The Map-phase partition the spatial objects of two data sets and create groups. Each group contains spatial objects that lies within spatial pre-defined boundaries.

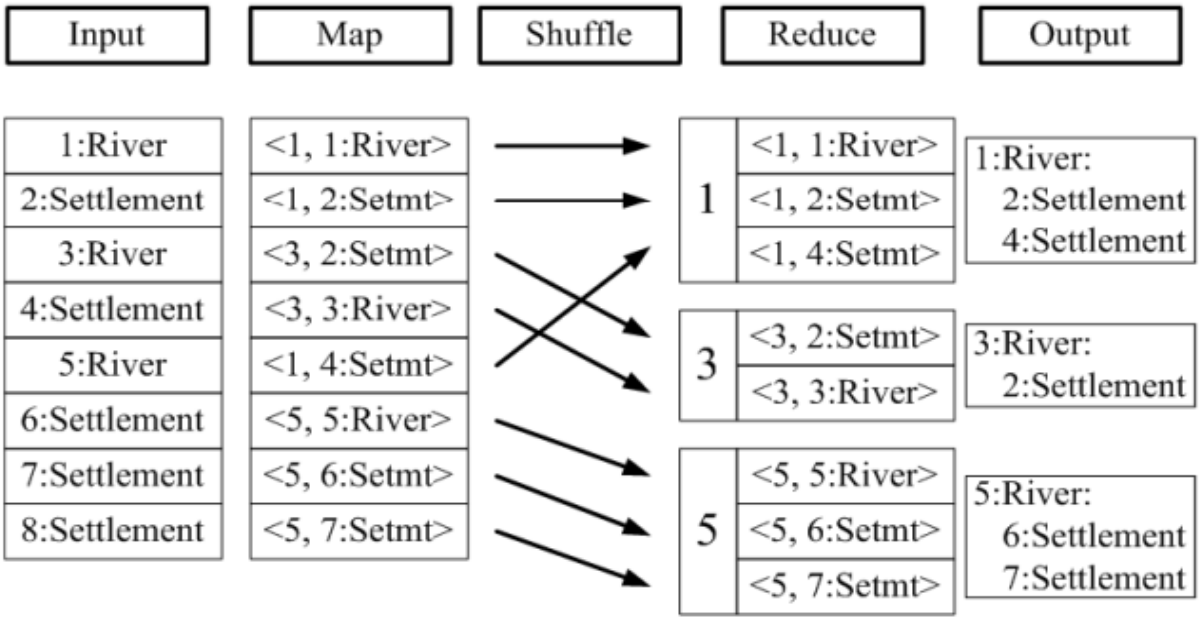


Figure 2.8.1 Spatial Join via MapReduce (Source: Large Spatial Data Computation on Spatial DBMS cluster via MapReduce)

2.8 Join Algorithms in MapReduce

When processing large data sets the need for joining data by a common key can be very useful, if not essential. The needs for joining data are many and varied. It is essential to mention that join algorithms developed for RDBMS are not appropriate to execute over MapReduce . The join algorithms suitable to run for MapReduce fall into the three categories.

- Reduce-side joins
- Map-side joins
- Broadcast joins (In-memory join)

If the join is performed in the map phase, it is a map-side join, and if done in the reduce phase, it is called the reduce-side join (Figure 2.9.1). Broad-cast join is a more efficient map-side join.

The reduce-side joins seems like the natural way to join tables using Map/Reduce. Hadoop offers another way of joining tables without using reducers. This allows a faster join, however requires, that (i) all the input tables be sorted in the same order on the join key. This is simply for performing the least number of comparisons on the join key. (ii) All the input tables use the same partitioner module with the same algorithm and parameters. (iii) The number of partitions of the input tables must all be the same. A given key must be in the same partition in both tables so that all partitions having the same key are joined together.

The Map-side join (Figure 2.9.2) eliminates the sorting and shuffling (distributing map outputs to reducers) and performs better than the reduce-side join in terms of response time . Map-side join is quite popular in DFS. The reason is that , unlike in a parallel RDBMS where data is located near to the computation module, there is no guarantee that the tables that join

are located on the same node. In other words, the NameNode makes independent decisions over where to put data blocks.

The Broadcast join(Figure 2.9.3) is a map-only algorithm. If one table is small enough to fit in the memory, it is loaded into the memory. The map function is then called for each tuple in the bigger table, one at a time. The map function probes the in-memory table and finds the matching tuples. Loading the small table into a hash table can further speed up this process.

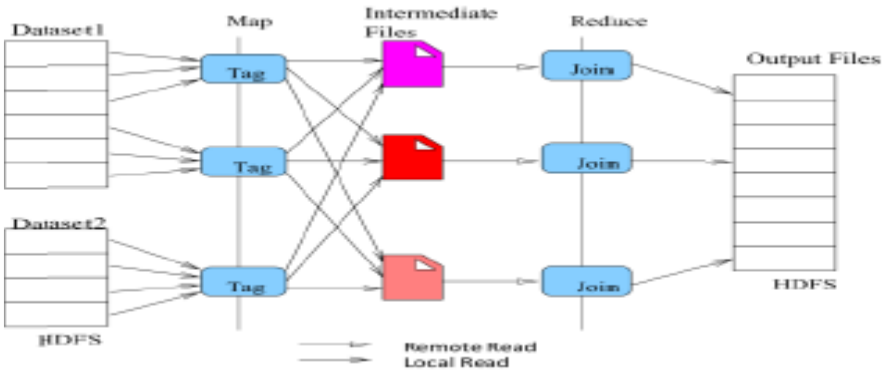


Figure 2.9.1 Data Flow Reduce Join example (Source: Join Algorithms Using MapReduce: A Survey)

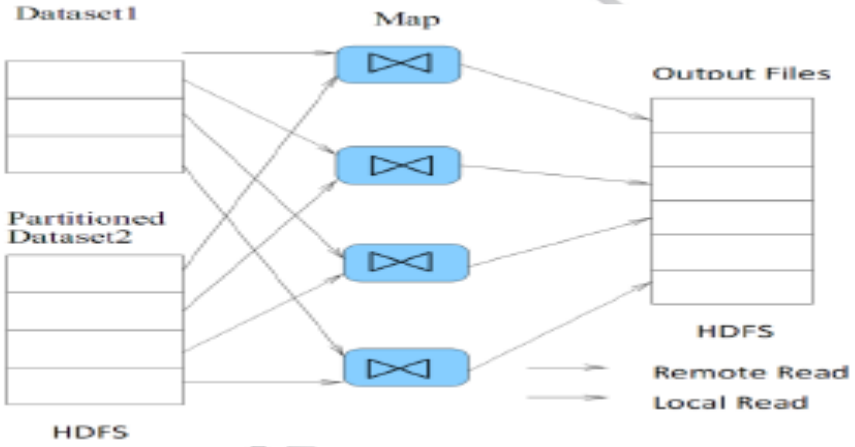


Figure 2.9.2 Data Flow for Map-Side Join(Source: Join Algorithms Using MapReduce: A Survey)

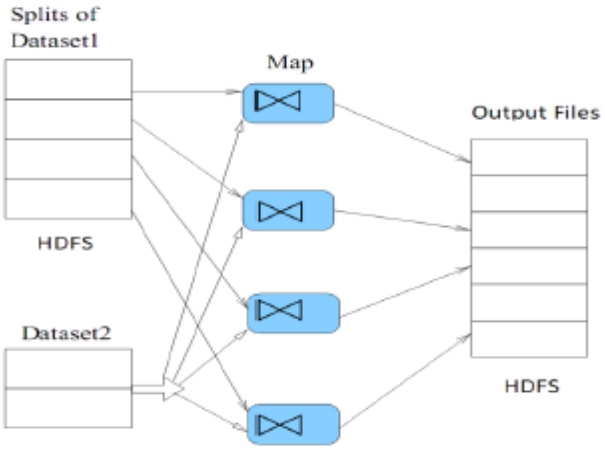


Figure 2.9.3 Data Flow for Memory Backed Join (Source: Join Algorithms Using MapReduce: A Survey)

The indexing process is split into m map tasks. Each map task operates on its own subset of the data, and is similar to the single-pass indexing corpus scanning phase. However, when memory runs low or all documents for that map have been processed, the partial index is flushed from the map task, by emitting a set of <term, posting list pairs. The partial indices (flushes) are the sorted by term, map and flush numbers before being passed to a reduce task. As the flushes are collected at an appropriate reduce task, the posting lists for each term are merged by map number and flush number, to ensure that the posting lists for each term are in globally correct ordering. The reduce function takes each term in turn and merges the posting lists for that term into the full posting list as a standard index.

2.9 Why MapReduce

The question that is raised of course is why not use traditional data bases which will access a very large number of disks? The answer comes from another trend which prevails in connection with hard disks: the accessing time is improved at a very slow rate in comparison with data transfer times. Disk accessing is the procedure during which there is shifting by its head to a more specific point in order to read or record data. This characterizes the latent state of disk operation, whereas the data transfer rate corresponds to its zone range. Consequently more time will be needed in order to read or record a data subset on the disk in comparison with the continuous flow of data at a certain transfer speed. On the other hand, for the updating of a subset of recordings in a database, a traditional B-tree (the data structure used in relational databases and is characterized by the limiting of the processing rate of the search) works satisfactorily. In the majority of databases as regards the process of recording updating, a B-Tree is less effective in comparison with the MapReduce methodology, which uses Sort/Merge procedures in order to construct the database.

In many cases, the MapReduce can be considered to complement a traditional RDBMS. (The differences between the two systems are presented in Table 1-1). MapReduce is a very good application for problems which require the analysis of the total of data in batch form, especially in cases of ad hoc analysis. A traditional RDBMS is an ideal solution for queries or updates regarding an indexed data set, with short waiting and data retrieval time and relatively small number of records. MapReduce is suitable in cases of applications where the data are

recorder once but are retrieved many times, whereas a relational database is appropriate for data sets which are constantly updated.

	Traditional RDBMS	MapReduce
Data size	Gigabytes	Petabytes
Access	Interactive and batch	Batch
Updates	Read and write many times	Write once, read many times
Structure	Static schema	Dynamic schema
Integrity	High	Low
Scaling	Nonlinear	Linear

Table 1-1. Comparison between a traditional RDBMS and MapReduce**.**

Another difference between MapReduce and RDBMSs is the amount of structure in the datasets that they operate on. Structured data are the data which are organized in entities based on a unified form, like XML documents or data tables which are created based on a predefined form. This is the realm of traditional databases. On the other hand, semi-structured data are distinguished by a looser form, and if a shape can exist this is ignored, so it can be used simply as a guide for the data structure. For example, in an Excel spreadsheet in which the structure is the total of the cells each one of which can have a specific type-form (although the cells themselves may hold any form data). Unstructured data sets have no internal structure: for example a simple text or a photograph. MapReduce responds very well to scenarios of unstructured or semi-structured data, given that its main mission is to interpret the information during the processing. In other words, the input keys and values for MapReduce are not an inherent quality of the data, but are selected by the programmer who is going to analyses the data. MapReduce enjoys such great success and acceptance despite its short existence because it offers the possibility to write a simple program and perform these effectively in thousands of computers within a very short period of time. It also allows programmers who have no prior experience in allocated or parallel systems to utilize an indefinite number of resources.

CHAPTER 3 NoSQL Databases

3.1 What is NoSQL Databases

NoSql is a broad class of database management systems that differ from the traditional relational databases models at a significant level and do not use SQL as their primary query language. The storage of this type of data might not necessarily follow fixed table schemas, it does not usually support join operations (joins), while it is also possible that it does not fully conform to atomicity, consistency, isolation and durability attributes (ACID). NoSQL databases are often distinguished based on the type of storage and fall under categories such as key-value, BigTable application, document oriented and graph databases. NoSQL application systems arose alongside the gigantic development of Internet companies, like Google, Amazon, Twitter and Facebook, which face greatly different challenges considering the data that conventional databases would not be able to cope with. With the rapid increase of real-time

web, the need for the provision of information of large volume, consistency and quality, which follow relatively the similar horizontal structures also emerged. These companies realize that the return and flow of information in real time is a much more important element than the consistence privileged by conventional relational databases and spend considerable effort and time towards that goal. Therefore, NoSQL databases show significant optimization with regard to recovery and adding operations, while on the other hand they offer limited functionality other than document storage processes. Reduced flexibility compared to full SQL systems is compensated by large gains in performance and scalability of the specified data model.

The promise of the NoSQL database has generated a lot of enthusiasm, but there are many obstacles to overcome like:

a. *Maturity*

RDBMS systems have been around for a long time. NoSQL advocates will argue that their advancing age is sign of their obsolescence, but for most CIOs, the maturity of the RDBMS is reassuring. For the most part, RDBMS systems are stable and richly functional. In comparison, most NoSQL alternatives are in pre-production versions with many key features yet to be implemented

b. *Support*

The most NoSQL systems are open source projects, and although there are usually one or more firms offering support for each NoSQL database, these companies often are small start-ups without the global reach, support resources, or credibility of an Oracle, Microsoft, or IBM

c. *Analytics and business intelligence*

NoSQL databases offer few facilities for ad-hoc query and analysis. Even a simple query requires significant programming expertise, and commonly used BI tools do not provide connectivity to NoSQL.

d. *Administration*

The design goals for NoSQL may be to provide a zero-admin solution, but the current reality falls well short of that goal. NoSQL today requires a lot of skill to install and a lot of effort to maintain.

e. *Expertise*

Almost every NoSQL developer is in a learning mode. This situation will address naturally over time, but for now, it's far easier to find experienced RDBMS programmers or administrators than a NoSQL expert.

NoSQL databases are becoming an increasingly important part of the database landscape, and when used appropriately, can offer real benefits. However, enterprises should proceed with caution with full awareness of the legitimate limitations and issues that are associated with these databases.

Chapter 4 Hadoop

4.1 Hadoop Framework

Hadoop, Hive, Hbase and Cassandra architectures distribute data in such a way that the whole processing power stemming from the Hadoop nodes cloud is exploited.

They guarantee a non-relational approach to the definition of the database schema. As to queries and updates, Cassandra employs syntax similar to T-SQL, known as CQL. Hive and Hbase use HiveQL**.**

Hadoop is the most common application –written in open source Java-based MapReduce technology. It is the commands and operations network used for the creation and performance of distributed applications that process large volume data. It has been designed for offline processing and large scale data analysis. It does not operate in random read/write contexts for a small number of documents, which is the objective of OLTP systems. It was created by Doug Cutting, who is the creator the widespread text search library Apache Lucene. It specializes in continuous data flow applications which should be stored once and managed and analyzed multiple times. In January 2008, Hadoop was the top-level Apache project, thus confirming its success and impact. Hadoop was launched in February 2008 when Yahoo announced that the search engine has used 10,000 Hadoop clusters. It was later used by other companies too (Facebook, Last.fm , New York Times, IBM, Google, Yahoo, Amazon/ec2 etc.). In April 2008, Hadoop was registered as the fastest system in classification of large volume data (terabytes) (29 seconds). One year later, it announced that only 62 seconds were needed for the classification of one terabyte of data. Queries are processed in parallel exploiting all nodes (ability to exploit more than 1.000 nodes) using MapReduce programming model. Hadoop's main goal is to create the necessary conditions for storing data and performing commands in multiple commodity computers. There machines normally operate Linux operating system.

Hardware cluster running Hadoop in Yahoo.

A fundamental element of relational databases is that their data is stored in tables on relational structure defined by a schema. Although the relational model has a complete set of attributes, several new applications specialize in types of data that do not fit very well in this model. Text files, images and XML files are a few popular examples. Hadoop was not initially designed for structured data analysis, therefore the performance of parallel database systems for the analysis of this type of data is significantly better. Hadoop uses key/value pairs as its basic data unit, which is flexible enough to collaborate with less structured types of data. In Hadoop, data can initiate from any other format but ultimately they are converted into key/value pairs so as to be processed later.

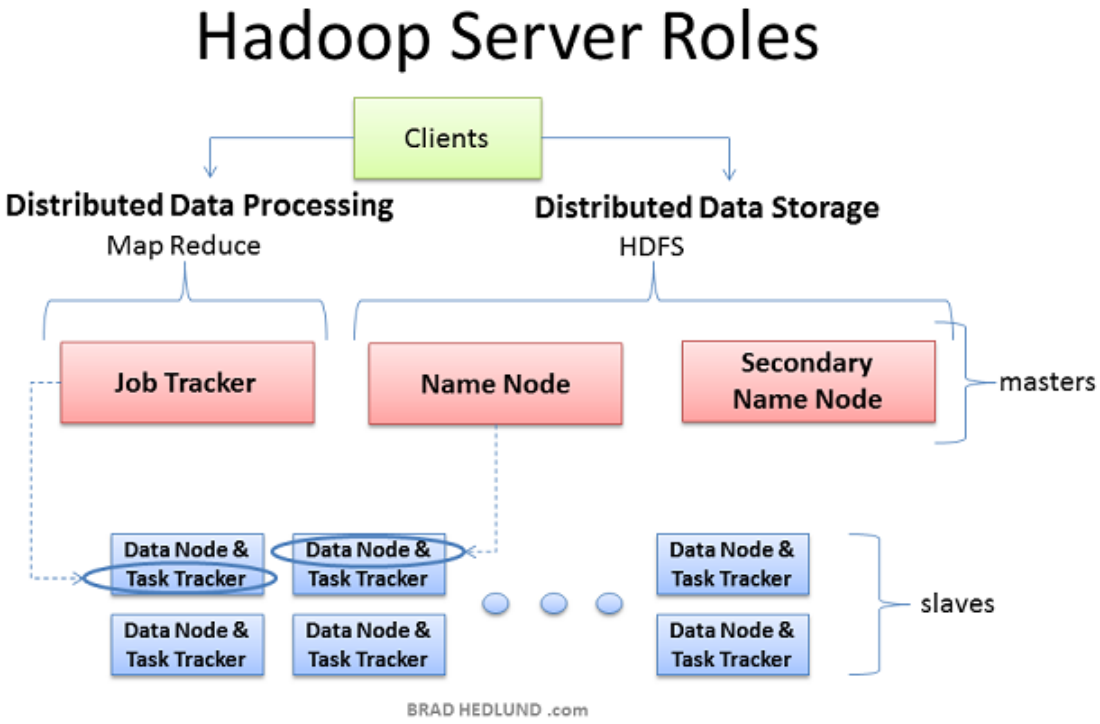


Figure 4.1.1: Hadoop Architecture

Some of the advantages of Hadoop include:

- Accessible: Hadoop runs in large commodity computers clusters or as a cloud computing service as is the case with Amazon's Elastic Compute Cloud (EC2).
- Robust: because Hadoop is configured to run on hardware with relatively limited capacity, its architecture is tailored to address common hardware malfunctions in a short period of time through automated recovery procedures. There is no comparable mechanism used for troubleshooting determinant software failures due to bad records. Instead, it offers techniques for the deletion of records which are expected to cause termination of a current job. When the process of deletion is active, JobTracker detects and clears out the documents area that causes the failure. Then TaskTracker restarts the job but ignores the specified documents area.
- Scalable: Hadoop extends linearly in order to manage more data and serve more clients, adding more nodes on cluster.
- Simple: Hadoop allows programmers to write effective code simultaneously (thus faster).
- HDFS stores a large amount of information.
- HDFS is a simple, reliable, powerful and coherent data storage model.
- HDFS should integrate well with Hadoop MapReduce, allowing data to be computed upon and read locally when possible.
- HDFS provides streaming read performance.
- Data can be store in HDFS once and be read many times.
- Processing logic adjusts to the data, rather than the data to the processing logic.

- Portability across heterogeneous hardware and operating systems.
- Economical solution due to distributing data across clusters of commodity personal computers.
- Efficiency by distributing data and logic to process it in parallel on nodes where data is located
- Reliability due to automatically creating and maintaining multiple replicas of data and automatically redeploying processing logic in the event of failure.
- HDFS is a system that consists of blocks of structured files. Each file is broken into blocks of fixed size which are stored in one or more machines with data storage capacity through a node.
- Ability to develop MapReduce programs in Java, a language which even non-computer experts can sufficiently comprehend so as to be able to meet powerful data-processing procedures.
- Ability to rapidly process large data volumes in parallel
- It can be offered in the form of on-demand service, for example as part of Amazon's EC2 cluster computing service

Some of the disadvantages of Hadoop include:

- Abnormal behavior, because the software is under constant development.
- The programming model is very restricted.
- Multiple dataset joins are difficult and slow. There are no indices. Often entire dataset are copied during processing.
- Cluster management is intractable. In the cluster, operations like debugging, distributing software and collection of logs are fairly hard to manage.
- Still single master which require care and may limit scaling.
- Managing job flow is not in the slightest simple when intermediate data should be retained.
- The optimal configuration of nodes is not obvious, e.g. #mappers, #reducers, memory limits

Hadoop's accessibility and simplicity form its significant advantage to create and perform large distributed applications. What is more, its flexibility and scalability make it suitable even for the most demanding jobs (like Yahoo and Facebook). These features make Hadoop widely accepted both in the academia and the industry.

Hadoop consists of four main components:

- Crawler, which downloads pages from web servers.
- WebMap, which creates a web graph and contains all metadata for every web address. It also contains hundreds of attributes for each document, which are utilized by the classification algorithm of empirical systems in order for documents to be classified.
- Indexer, which builds a reverse index in the first pages.

- Runtime, which answers users' queries.

and can be divided into two levels:

- Data storage or the Hadoop Distributed File System (HDFS)
- Data processing or the MapReduce programming model

The typical workflow of the Hadoop technology is:

- Loading of data on clusters
- Data analysis (MapReduce)
- Results storage on cluster (HDFS writes)
- Reading results from cluster (HDFS reads)

4.2 HDFS Architecture

HDFS (Hadoop File System) is a clustered system of grouped files, which is controlled by a central node and is responsible for data storage and management. It is designed to accommodate large volume of imported data (petabytes) – which can be one simple file – per time unit and manage them accordingly. Individual files are allocated in fixed-length blocks and distributed in multiple server nodes. The node retains metadata according to the size and the position of blocks and their replicas. It operates in MapReduce context. A typical Hadoop workflow creates data files and copies them in HDFS. Then, MapReduce applications process this data but, they usually does not read any HDFS files directly. Instead, they are based on MapReduce framework in order to read and analyze HDFS files to individual records (keys/value pairs) which form the data unit upon which MapReduce will be processed. HDFS has been mainly designed for batch processing rather than bidirectional communication with the user. Focus is primarily placed on high access to data rather than fast data recovery. HDFS has a master/slave architecture and consists of a simple NameNode, a central server that manages the file system and regulates access to files from clients. There is also a number of DataNodes, usually one per machine in every cluster, which manage storage procedures in the node they are connected to.
More specifically:

- Clients Machines : Hadoop has been installed on these machines, with all cluster settings, still they are not Master or Slave. The role of client machines is to load data on the cluster, request MapReduce jobs describing how data processing will be done and then recovering the results from these jobs. In smaller clusters (up to 49 nodes) one small server with multiple roles like Job Tracker and NameNode is enough. In medium or large clusters, one functionality-role is usually mapped to one server.
- Master Nodes : Oversee key operational features that compose Hadoop: storage of large volume of data (HDFS) and performance of parallel computations on all data (Map Reduce).

- NameNode oversees, records and coordinates data storage functions (HDFS). It keeps tracks on the way in which the original files are broken into smaller file blocks, which nodes store which blocks, as well as the more general status of HDFS. NameNode functions are memory and I/O intensive. As such, NameNode typically doesn't store any user data or perform any computations for a MapReduce program to lower the workload on a specific machine. This means that the NameNode server doesn't double as a DataNode or a Job Tracker. Unfortunately, this is a negative aspect of the NameNode and a global point of failure of the Hadoop cluster. For any of the other services, if host nodes fail for reasons related to software of hardware, the Hadoop system will probably keep operating normally or restart immediately. This is not the case with NameNode.

The existence of a NameNode in a cluster simplifies the system architecture significantly. NameNode performs control and deposition of all HDFS metadata. The system has been designed in such a way, so that user's data will by no means be distributed through the NameNode.

The Secondary NameNode (SNN) is an assistant service for monitoring the state of the cluster HDFS. Similar to NameNode, each cluster has a SNN. The SNN differs from the NameNode in that it does not receive real-time records for changes in HDFS. Instead, it communicates with the NameNode to take snapshots of HDFS metadata at intervals defined by the cluster configuration. The SNN helps minimize downtime and loss of data.

- Job Tracker is the join between the user application and Hadoop. After loading the application on cluster, JobTracker determines the execution plan defining which files it will process, assigns nodes to different jobs and oversees all current jobs. If a job fails, the JobTracker will automatically relaunch the job, possibly on a different node, for a predefined limit of retries. It also provides the user interface with a MapReduce cluster.

There is only one JobTracker service per Hadoop cluster. It is typically run on a server, such as a master node of the cluster.

- Slave Nodes : Slave Nodes make up the vast majority of machines and do all the dirty work of storing data and execution. Each slave runs Data Node and Job Tracker jobs. Job Tracker is a perpetual (daemon) process that receives instructions from Master Nodes. The Job Tracker is a slave to the Job Tracker and it is responsible for the execution of a specific job on every Slave Node. Although there is a single JobTracker per Slave Node, each JobTracker can replicate multiple JVMs and manage many map or reduce jobs in parallel. JobTracker is also responsible to steadily communicate with the JobTracker for a specific period of time in order to reconfirm that it is still active. Furthermore, it provides updates on the number of available Slots. If the JobTracker does not receive an active signal from them JobTracker within a specific period of time, it assumes that the JobTracker does not operate due to failure and redirects the corresponding jobs to other nodes on the cluster.

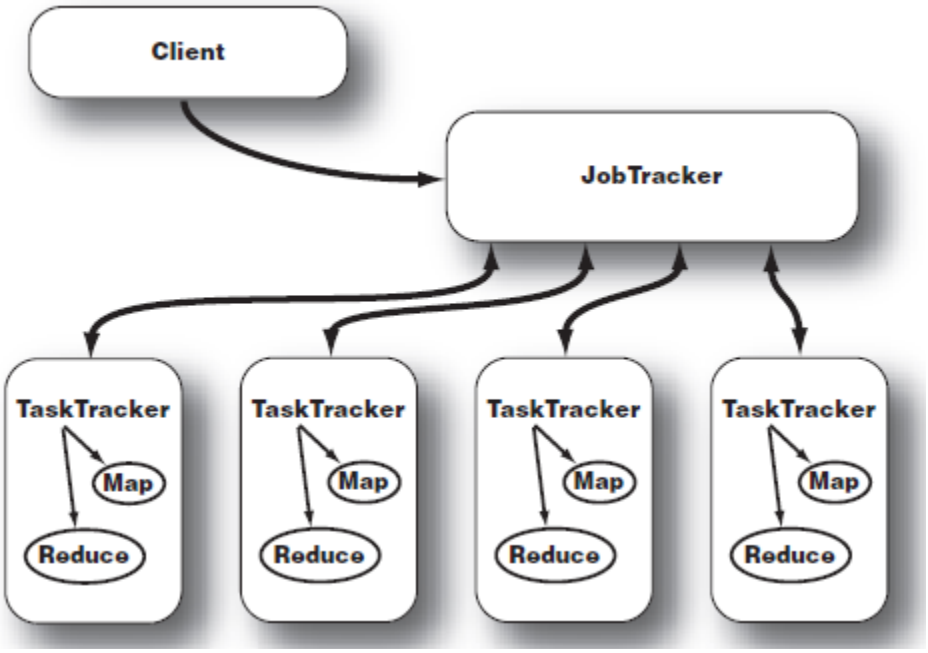


Figure 4.2.1: Interaction between the JobTracker and the JobTracker.

Each slave machine hosts a DataNode service which performs jobs of reading and writing HDFS blocks to actual files on the local file system. When reading an HDFS file, the file breaks into blocks and the NameNode informs the client which DataNode each block resides in The client communicates directly with the DataNode service which will process the local files corresponding to the blocks. Furthermore, a DataNode may communicate with other DataNodes to replicate its data blocks.

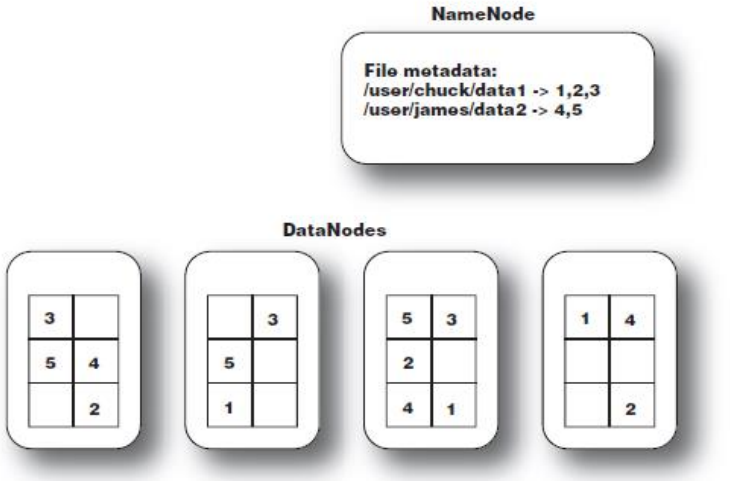


Figure 4.2.2.:NameNode/DataNode interaction in HDFS. NameNode keeps track of metadata files set in the system and each file is distributed in blocks. DataNode provides procedures for maintaining block replicas and reports regularly to NameNode, where current metadata is kept. Finally, it provides information regarding local changes as well as receives instructions to create, transfer or delete blocks from the local drive. The system architecture does not preclude the existence of multiple DataNodes on the same machine, although this is rear in a real context.

Hadoop characteristics: Any data job request, whether defines in HiveQL, Pig Latin or other interfaces, is performed as a batch operation. Every job is scheduled and distributed to as many nodes as possible. In the background, the startup and execution process for the requested jobs is monitored and managed by Hadoop(Figure 4.2.2).

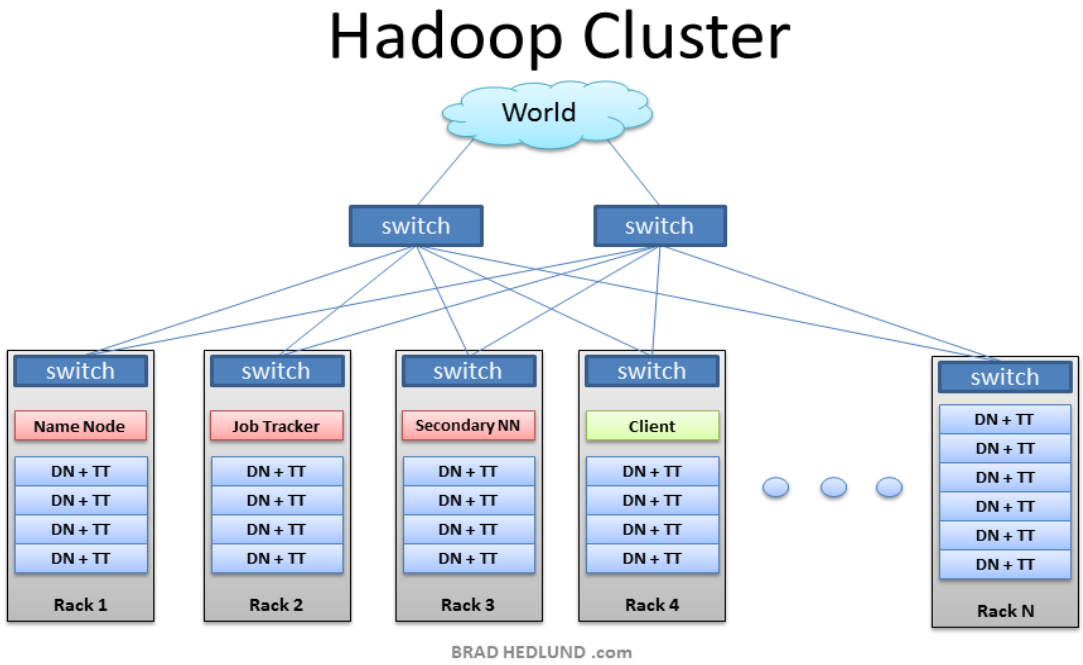


Figure 4.2.2: Hadoop Cluster

This is the typical architecture of a Hadoop cluster. It consists of a set of Servers populated in racks connected to the outside though a top of rack switch. This switch is matched to another tier of switches –connecting all the other racks- with uniform bandwidth. The majority of the servers are slave nodes with lots of local disks and moderate amounts of CPU and DRAM. Some of the machines need to be Master Nodes that might have a slightly different configuration favoring, grater memory and processor capacity and less local storage. Client computer send jobs to the Hadoop Cluster (which can be a computer cloud) and wait for the results.

4.3 Hadoop Data Types

MapReduce has explicitly defined the way in which key/value pair become serialized and transfer through the cluster network, while only classes that support this kind of serialization can function as keys or values in the MapReduce job environment. The two basic types of data are WritableComparable, which is the basic interface for keys, and Writable which is the basic class interface for values:

- Writable Types
  Objects which can be integrated to data files and travel across the network must obey to a particular interface, called Writable, which allows Hadoop to read and write data in a serialized form. Hadoop provides several class categories which implement the Writable interface: Text (which stores string data), IntWritable, LongWritable, FloatWritable, BooleanWritable, and other.

  In addition to these types, the user is free to define their own class which implements the Writable interface. For example, it could be defines as a mapper that contains key-value pairs where the key is the name of an object and value is the coordinated in a 3D area. The key is string data, while value is a structure with the following form:

  Struct point3d {

  Float x;

```
        Float y;
        Float z;
        }
```

The key can be represented as a Text object, however in the case of the value, the Writable interface needs to be implemented, which requires two methods:

```
Public interface Writable {
  void readFields (DataInput in);
  void write(DataOutput out);
}
```

The first of these methods initializes all of the fields of the object included in the binary stream in. The latter writes all the necessary items to reconstruct the object to the binary stream out. The DataInput and DataOutput classes (part of java.io) include methods that serialize most basic types of data. The important admission between the readfields and write methods is that they read and write data from the binary stream in the same order. The following code creates a Point3D class which is used by Hadoop:

```
Public class Point3D implements Writable {
  Public float x;
  Public float y;
  Public float z;
 Public Point3D(float x, float y, float z) {
    This.x=x;
    This.y=y;
    This.z=z;
}
  Public Point3D() {
    This(0.0f, 0.0f,  0.0f);
}
 Public void write(Data Output out) throws IOException{
    Out.writeFloat(x);
    Out.writeFloat(y);
     Out.writeFloat(z);
}
Public void readFields (DataInput in) throws IOException {
  x=in.readfloat();
  y=in.readfloat();
  z=in.readfloat();
}
```

```
Public String toString() {
  Return Float.toString(x) + ",  "
        +Float.toString(y) + ", "
        +Float.toString(z);
  }
}
```

- WritableComparables can be compared to each other through Comparators. Each type that is going to be used as a key in the Hadoop Map-Reduce framework could implement this interface.

```
Public class MyWritableComparable implements WritableComparable {
   //Some Data
Private int counter;
Private long timestamp;
Public void write(DataOutput out) throws IOException {
     Out.writeInt(counter);
     Out.writeLong(timestamp);
}
Public void readFields(DataInput in) throws IOException {
    Counter = in.readInt();
     Timestamp=in.readLong();
}
Public int CompareTo(MyWritableComparable o) {
  int thisValue=this.value
  int thatValue = o.value
 return (thisValue < thatValue ? -1 : (thisValue==thatValue ? 0:1));
}
Public int hascode() {
    Final int prime =31;
    Int result =1;
    Result=prime * result + counter;
    Result = prime * result + (int) ( timestamp ^ (timestamp >>> 32));
   Return result
}
}
```

HashCode is often used in Hadoop for the partition of keys. What is important is that the implementation of hashCode returns the same result in different JVM cases.

Example of a typical Hadoop program:

```
Public class MyJob extends Configured implements Tool {
Public static class MapClass extends MapReduceBase
Implements Mapper<Text, Text, Text, Text> {
Public void map(Text  key, Text  value,
        OutputCollector<Text, Text> output,
        Reporter reporter) throws IOException {
          output.collect(value , key);
```

```
        }
      }
   Public static class Reduce extends MapReduceBase
        Implements Reducer<Text, Text , Text , Text> {
   Public void reduce(Text key, Iterator<Text> values,
         OutputCollector<Text, Text> output,
         Reporter reporter) throws IOException {
         String csv=" " ;
         While (values.hasNext()) {
           If (csv.length() >0) csv+=",";
           csv += values.next().toString();
           }
        Output.collect(key, new Text(csv));
        }
}
   Public int run(String[] args) throws Exception {
      Configuration conf  = getConf();
      JobConf  job = new JobConf(conf, Myjob.class);
     Path in=new Path(args[0]);
    Path out = new Path(args[1]);
    FileInputFormat.setInputPath(job, in);
     FileOutputFormat.setOutputPath(job, out);
     Job.setJobName("MyJob");
    Job.setMapperClass(MapClass.class);
    jobsetReducerClass(Reduce.class);
    job.setInputFormat(KeyValueTextInputFormat.class);
    job.setOutputFormat(TextOutputFormat.class);
    job.setOutputKeyClass(Text.class);
     job.set("key.value.separator.in.input.line", ",");
    JobClient.runJob(job);
  Return 0;
}
Public static void main(String[] args) throws Exception {
      Int res=ToolRunner.run(new Configuration(), new MyJob(), args);
```

```
            System.exit(res)
        }
    }
```

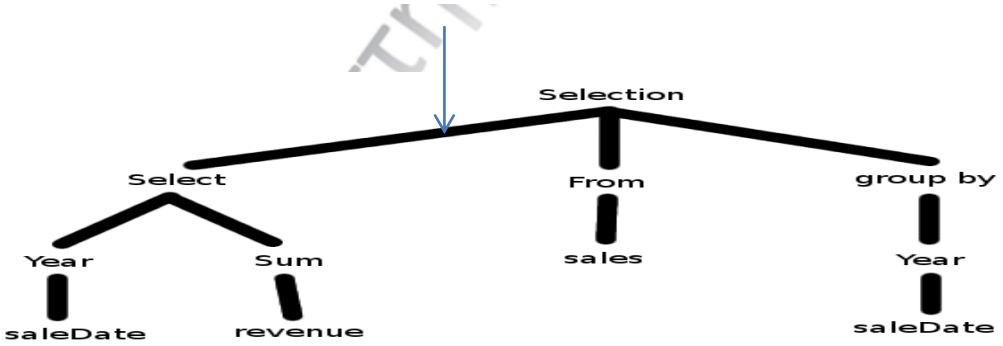
A simple class – called MyJob- defines explicitly every MapReduce job. Hadoop requires Mapper and Reducer to have their own static class. These classes are very small and the example above includes them as internal classes in the MyJob class.

The advantage is that everything fits in one file, simplifying code management. The core of the skeleton lays within this run() method and is also known as a driver. The driver instantiates configures and transfers a JobConf object called JobClient. The runJob initiates the MapReduce job. (The JobClient class will in turn communicate with the JobTracker to start the job that will run across the cluster). The JobConf object will keep all configuration parameters that are necessary for the job to run. It is necessary that the driver specifies in the job all the input and output paths, the Mapper class and the Reducer class, which form the basic parameters for every job.

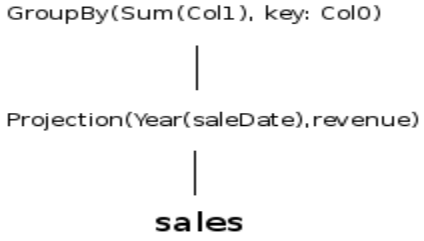
Code development in Hadoop differs from the conventional ways of code writing in two focal points. The first is that Hadoop programs primarily deal with data processing, and the second is that Hadoop programs are executed in distributed groups of computers. Hadoop uses a simple SQL-like language, called HIVEQL and converts SQL queries into MapReduce jobs:

1. *Transform query to AST (abstract syntax tree)*

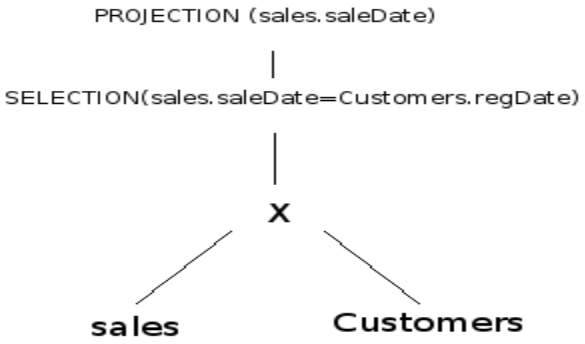
```
SELECT YEAR(saleDate), SUM(revenue)
FROM sales GROUP BY YEAR(saleDate);
```



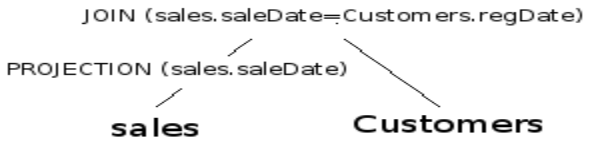
2. *Transform AST to DAG (Logical query Plan):*



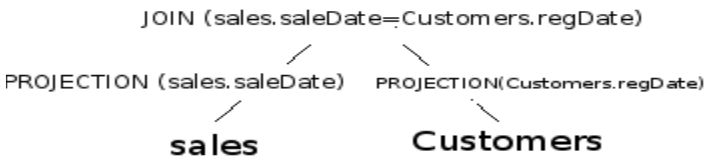
3. *Optimization plan:*



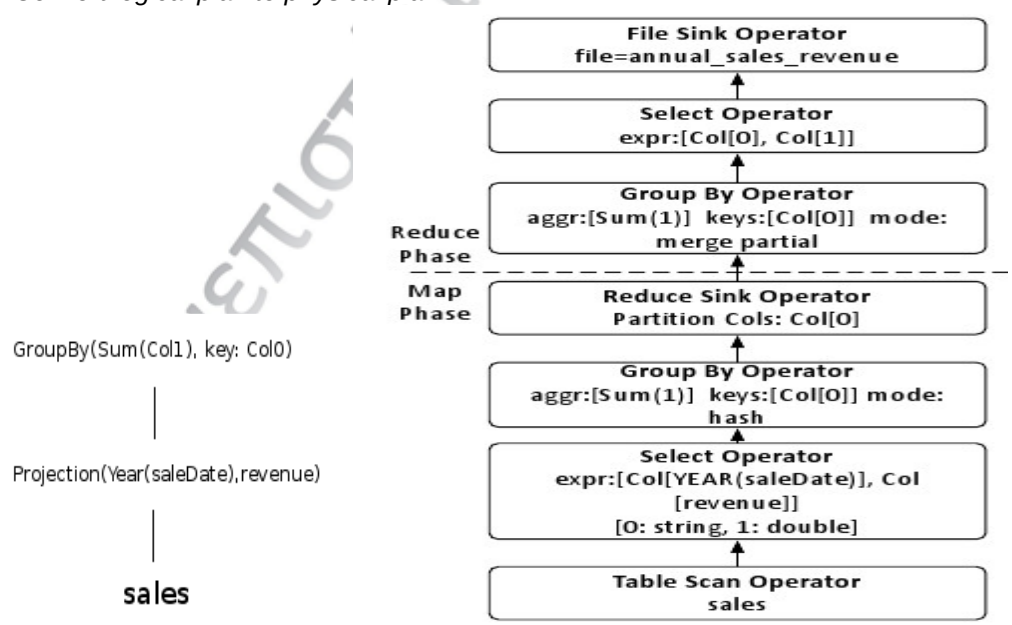
4. *Optimization plan: (Replace cross product by join)*



5. *Optimization plan: (add projection)*



6. *Convert logical plan to physical plan:*



Then the plan is stored in an XML file and the Hadoop process begins.

Hadoop uses a key/value structure in data storing. This indicates that values which compose a logical record are stored as distinct physical records. As a result of this distinction, data structure is particularly flexible. New columns can easily be added during the execution of the jobs. What is more, different records in a table can have different sets of columns. Storage formats are also suitable both for structured data storage and large volume non-structured data storage, like sound files, images and documents.

4.4 Data Distribution

In a Hadoop cluster, data is distributed to all the nodes of the cluster as it is being loaded in. The Hadoop Distributed File System (HDFS) will split large data files into chunks which are managed by different nodes in the cluster. In addition to this each chunk is replicated across several machines, so that a single machine failure does not result in any data being unavailable. An active monitoring system then re-replicates the data in response to system failures which can result in partial storage. Even though the file chunks are replicated and distributed across several machines, they form a single namespace, so their contents are universally accessible.

Data is conceptually record-oriented in the Hadoop programming framework. Individual input files are broken into lines or into other formats specific to the application logic. Each process running on a node in the cluster then processes a subset of these records. The Hadoop framework then schedules these processes in proximity to the location of data/records using knowledge from the distributed file system. Since files are spread across the distributed file system as chunks, each compute process running on a node operates on a subset of the data. Which data operated on by a node is chosen based on its locality to the node: most data is read from the local disk straight into the CPU, alleviating strain on network bandwidth and preventing unnecessary network transfers. This strategy of moving computation to the data, instead of moving the data to the computation allows Hadoop to achieve high data locality which in turn results in high performance.

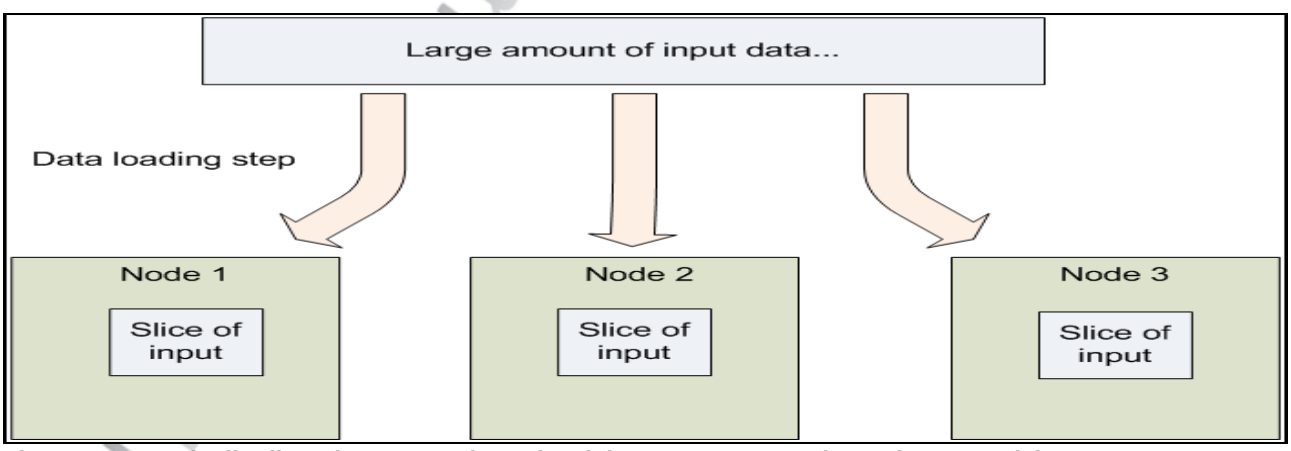


Figure 4.1: Data is distributed across nodes at local time . Source: Apache Hadoop Tutorial Introduction

Separate nodes in a Hadoop cluster still communicate with one another. However, in contrast to more conventional distributed systems where application developers explicitly marshal byte streams from node to node over sockets or through MPI buffers, communication in Hadoop is performed implicitly. Pieces of data can be tagged with key names which inform Hadoop how to send related bits of information to a common

destination node. Hadoop internally manages all of the data transfer and cluster topology issues.

By restricting the communication between nodes, Hadoop makes the distributed system much more reliable. Individual node failures can be worked around by restarting tasks on other machines. Since user-level tasks do not communicate explicitly with one another, no messages need to be exchanged by user programs, nor do nodes need to roll back to pre-arranged checkpoints to partially restart the computation. The other workers continue to operate as though nothing went wrong, leaving the challenging aspects of partially restarting the program to the underlying Hadoop layer.

Hadoop, however, is specifically designed to have a very flat scalability curve. After a Hadoop program is written and functioning on ten nodes, very little--if any--work is required for that same program to run on a much larger amount of hardware. Orders of magnitude of growth can be managed with little re-work required for your applications. The underlying Hadoop platform will manage the data and hardware resources and provide dependable performance growth proportionate to the number of machines available.

Current Hadoop's implementation of MapReduce doesn't support any sort of indexing mechanism. This is not a drawback of MapReduce, it is something that MapReduce has not been designed for. MapReduce has been designed for one time processing of large data sets in batch mode. As a future scope to empower MapReduce with indexing mechanism to make it suitable for real time data analysis.

4.5 Hadoop Example

Distinction based on the columns. Suppose that we want to create two data sets from patent metadata: one containing time-related information (e.g. publication date) for each patent and another one containing geographical information (e.g. country of invention). These two data sets may be of different output formats and different types of data for keys and values.

```
Public class MultiFile extends Configured implements Tool {
    public static class Mapclass extends MapReduceBase
    Implements Mapper<LongWritable, Text, NullWritable, Text> {
    private MultipleOutputs mos;
    private OutputCollector<NullWritable, Text> collector;
Public void configure(JobConf conf)  {
        mos = new MultipleOutputs(conf);
}
  Public void map(LongWritable key, Text value,
            OutputCollector<NullWritable, Text> output,
             Reporter reporter) throws IOException {
```

```
        String[] arr = value.toString().split(“,”,-1);
        String chrono = arr[0] + “,” + arr[1] + “,” + arr[2];
        String geo      = arr[0] + “,” + arr[4] + “,”+ arr[5];
        collector = mos.getCollector(“chrono”, reporter);
        collector.collect(NullWritable.get(), new Text(chromo));
        collector.collect(NullWritable.get(), new Text(geo));
    }
    Public void close() throws IOException {
            mos.close();
    }
    }
    Public int run(String[] args) throws Exception {
            Configuration conf=getConf();
            JobConf  job = new JobConf(conf,  MultiFile.class);
            Path  in = new Path(args[0]);
            Path out= new Path(args[1]);
            FileInputFormat.setInputPath(job, in);
            FileOutputFormat.setOutputPath(job, out);
            job.setJobName(“MultiFile”);
            job.setMapperClass(MapClass.class);
            job.setInputFormat(TextInputFormat.class);
            job.setOutputKeyclass(NullWritable.class);
            job.setOutputValueClass(Text.class);
            job.setNumReduceJobs(0);
            MultipleOutputs.addNamedOutput(job,“chrono”,
            TextOutpuFormat.class, NullWritable.class,
            Text.class);
            MultipleOutputs.addNamedOutput(job,
                    “geo”,
                    TextOutputFormat.class,
                    NullWritable.class,
                    Text.class);
                    JobClient.runJob(job);
                    return 0;
                }
```

```
Public static void main(String[] args) throws Exception {
        Int res = ToolRunner.run(new Configuration(),
                new MultiFile(),
                args)
        System.exit(res)
        }


        }
```

The MultipleOutput class of MapReduce is used, which is used to set up the output collectors it expects that will be used. Their creation involves a call to the addNamedOutput static method of the MultipleOutputs class. *We have created an output collector called "chrono" and second one called "geo".* They have both been created so as to use the TextOutputFormat and have the same key/value types, but we can choose to have different output formats or data types. After that, we call the MultipleOutputs object that tracks the collectors when the mapper becomes active in the call of a configure method. This object has to be available throughout the duration of the job. In the map() routine, the getCollector method is called by the MultipleObjecs object, and then it returns the chrono and geo collectors to output. Then, we will write different data that is appropriate for each of the two collectors. We have given a name to each of the two collectors in the MultipleOutputs routine and then the routine will automatically generate the output file names. In the following text, we can see how MultipleOutputs generates the output names:

ls –l output/

total 101896

-rwxwxrwx 1 Administrator None 9672703 Jul 31 06:28 chrono-m-0000

-rwxwxrwx 1 Administrator None 7752888 Jul 31 06:29 chrono-m-0001

-rwxwxrwx 1 Administrator None 9428951 Jul 31 06:28 geo-m-0000

-rwxwxrwx 1 Administrator None 7464690 Jul 31 06:29 geo-m-0001

-rwxwxrwx 1 Administrator None 0 Jul 31 06:28 part-0000

-rwxwxrwx 1 Administrator None 0 Jul 31 06:28 part-0000

-rwxwxrwx 1 Administrator None 0 Jul 31 06:29 part-0001

-rwxwxrwx 1 Administrator None 0 Jul 31 06:29 part-0001

These files can become records using the OutputCollector routine which is called through the map() method. The records could be:

head output/chrono-m-0000

“PATENT”,”GYEAR”, “GDATE”

3070801,1963,1096

3070802,1963,1096

head output/geo-m-0000

“PATENT”, “COUNTRY”, “POSTATE”

3070801, “BE”, “”

3070802, “US”, “TX”

Looking at the output files, we can see that the columns on the patent data have been successfully extracted into two distinct files.

4.6 Hadoop and FaceBook

Facebook characteristics:

- Three hundred active users
- Thirty million users update their status at least once everyday
- Over one billion photographs are uploaded every month
- Over ten million videos are uploaded every month
- Over one billion posts (web links, news stories, blog posts, notes, photos etc.) are shared every week

Daily statistics:

- Four TB compressed new data is added every day.
- One hundred thirty five TB of compressed data is scanned every day.
- Seven point five thousand Hive Jobs everyday per productive node.
- Eighty K hours of computer time every day.

Where is this data stored?

- Four thousand eight hundred processors, 5.5 Petabytes
- 12 TB per node
- Two level network topology
- 1Gbit/sec from node to rack switch
- 4 Gbit/sec to top level rack switch

Facebook is one of Hadoop and big data’s biggest champions and it claims to operate the largest single Hadoop Distributed Filesystem (HDFS) cluster anywhere, with more than 100 petabytes of disk space in a single system as of July 2012. The sites stores more than 250 billion photos, with 350 million new ones uploaded every day. According to owner of Facebook is used in every Facebook product and in a variety of ways. Users actions such as a “like” or a status update are stored in a highly distributed, customized MySQL database, but applications such as Facebook Messaging run on top of HBase, Hadoop’s NoSQL database framework. All messages sent on desktop or mobile are persisted to HBase. Additionally the company uses Hadoop and Hive to generate reports for third-party developers and advertisers who need to track the success of their applications or campaigns.

Hive, the data warehousing infrastructure Facebook helped develop to run on top of Hadoop, is central to meeting the company's reporting needs. Facebook must balance the need for rapid results in features such as its graph tools with simplicity and ease of reporting, so it is working on another contribution to Hive that will improve the speed of queries. Improving Hive's speed is important, as the scalability that makes the tool central to the social network's needs can come at the expense of low latency.

4.7 Hadoop communication with a conventional database

Although Hadoop is useful for processing large volumes of data, relational databases remain the workhorse of many data processing applications. Hadoop will often need to communicate

with these databases. Although it is better to set up a MapReduce program to take its input by directly querying a database rather than reading a file in HDFS, the performance is less than ideal. Communication can be much more easily achieved with the use of a standard utility that hosts a serial file. Then, this file can be uploaded on HDFS using the put command.

Figure 4.6.1 Simplified system diagram

However, sometimes it is essential to have a MapReduce program which will be able to create records directly onto the database. Many MapReduce programs take large volume data sets and then process them into a manageable size for databases to handle. For example, we often use MapReduce in the ETL-like process of receiving huge volume log data files and then computing a much smaller and more manageable set of statistics which is the desired element for a potential group of analysts.

The DBOutputFormat is the critical class for accessing databases, in which the format of exported data needs to be defined. The configuration for the connection to the database will also need to be specified. This can be achieved through the configured method of the class:

Public static void configureDB(JobConf , job, String driverClass,

String dbUrl, String username , String passwd)

After that, it is important to specify which table and which fields will be edited. This is done with the setOutput method in the DBOutputFormat class.

Public static void setOutput(JobConf job , String tableName,
String….fieldNames)

For example:

```
Conf.SetOutputFormat(DbOutputFormat.class);
DBConfiguration.configureDB(job,
                "com.mysql.jdbc.Driver",
                "username",
               "password")
DBOutputFormat.setOutput(job, "Events", "event_id" , "time");
```

It has to be clear that the process of reading and updating databases from the Hadoop platform is only suitable for datasets that are relatively small for the Hadoop standards. It is often better to bulk load data in the database rather than make direct updates from Hadoop. Specialized solutions for extremely large-scale databases will be necessary.

4.8 Hadoop - Business Intelligence

Business Intelligence (BI) refers to the technologies, tools and practices which regard the collection, integration, analysis and presentation of large amounts of information in such a way as to facilitate and support to best possible degree decision making. Today the BI architecture consists of the database, which unifies data from different sources of information and caters for a large variety of front – end queries, references and analysis tools. Back – end architecture is a data unification conductor for the activation of a database, which usually comes from heterogeneous and distributed sources of primary information. The unification includes procedures of marginal values in the primary structured or unstructured data, integration, conversion and loading of the information in the data repository (Figure 4.8.1). The traditional procedures of data integration are of simple transition and follow batch techniques because BI technologies are used for off-line supporting of decision making strategies. As the degree of procedure automation of organizations increases at geometric progression, and the dependence on the channeling of high quality, continuous flow and real time increases, BI architecture evolves all the time in order to meet the requested needs of supporting decision making (e.g. a bank would like to track down and react immediately to a fraudulent transaction). A consequence of these transformations is the increase and ticklishness of demands, resulting in a complexity in the designing of procedures for quarrying primary information and integrating them. The contemporary objective has to do now with updating the data depository with real time information.

Business Intelligence (BI) constitutes a collection of data depositories, information quarrying, analytics, references and visualization technologies, tools and practices that accumulate, integrate and remove the information “noises” and finally retrieve those information for the use of an organization, which can facilitate in the greatest possible degree the decision making. Today BI architecture has performed all these adaptations and changes so as to constitute the main ally of strategic procedures for decision making where a group of experienced users analyses historical data, prepares reports or creates models and circular flows of decision makings for the last months or weeks. This architecture can be depicted as a provisional chain of information .

Hadoop forms a fundamental role in the collection, transformation and publication of data. Using tools like Apache Pig, advanced transformation can be performed in Hadoop with little programming effort and since Hadoop constitutes low cost storage space, the data can be kept for months or even years. Since Hadoop is used for “cleaning” and transforming data, they can be loaded in a Data Warehouse or Meter data management system (it processes and manages large volume data using intelligent measuring systems and then disposes them for analysis).

Hadoop is not an Extract-Transform=Load (ETL) tool. It is a platform which supports ETL processes that are performed in parallel. The suppliers of data integration tools do not compete with Hadoop, rather Hadoop is another channel which uses its own models of transforming data.

Given that companies start by gather a large quantity of data, the transferring of them on the internet and their transformation or analysis becomes nonrealistic. The transferring of terabytes from one system to another on a daily basis can bring the network manager to an absolute deadlock. For this reason it is more logical to push the data processing procedure. The transferring to an internet storage area or ELT server is rather unfeasible in cases of large volumes of data as the procedure is going to be either quite slow or limited by the boundaries of the bandwidth. By using Hadoop, the unprocessed data are loaded directly to low cost servers at the same time and only processed outputs of higher value pass into the other systems. It is not surprising that many Hadoop systems are set up next to Data Warehouses systems. These systems serve different purposes and complement each other. For example: A big stock exchange company uses Hadoop to pre-process flows of unprocessed information which are created by clients using the website. The processing of these data offers valuable information about the preferences of the clients which after that enter a Data Warehouse. Then the DW correlates these client preferences with advertising campaigns and suggests mechanisms that offer investment and analysis recommendations to consumers.

Complex Hadoop tasks can use a DW as data source, utilizing at the same time the capabilities of parallel processing of the two systems.

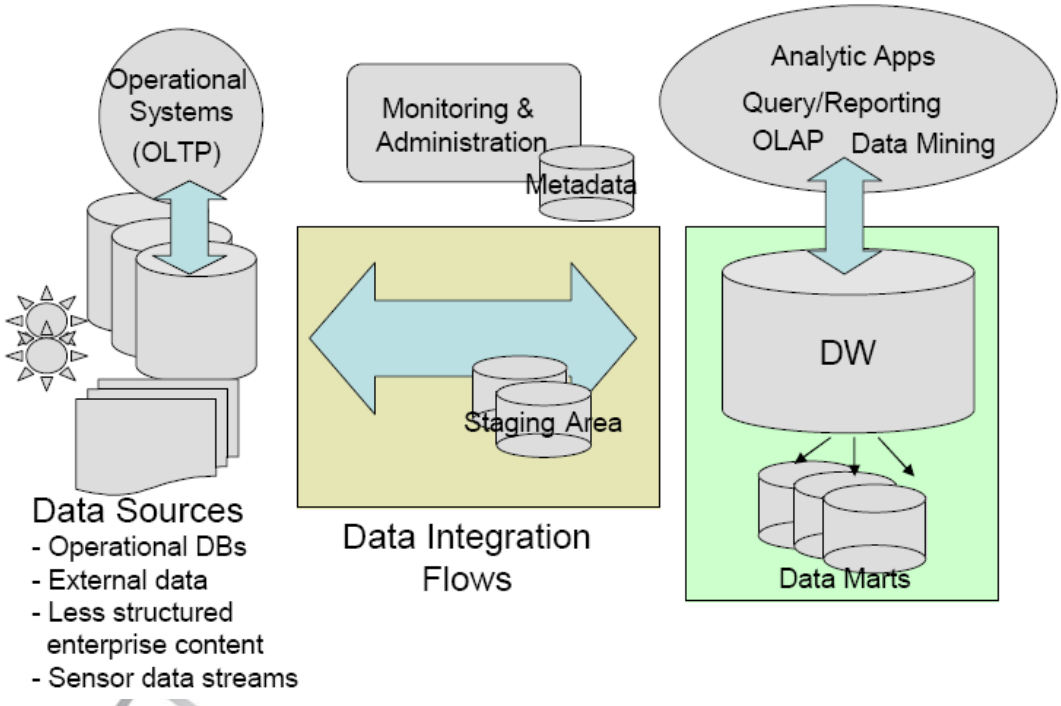


Figure 4.8.1: Traditional BI architecture.

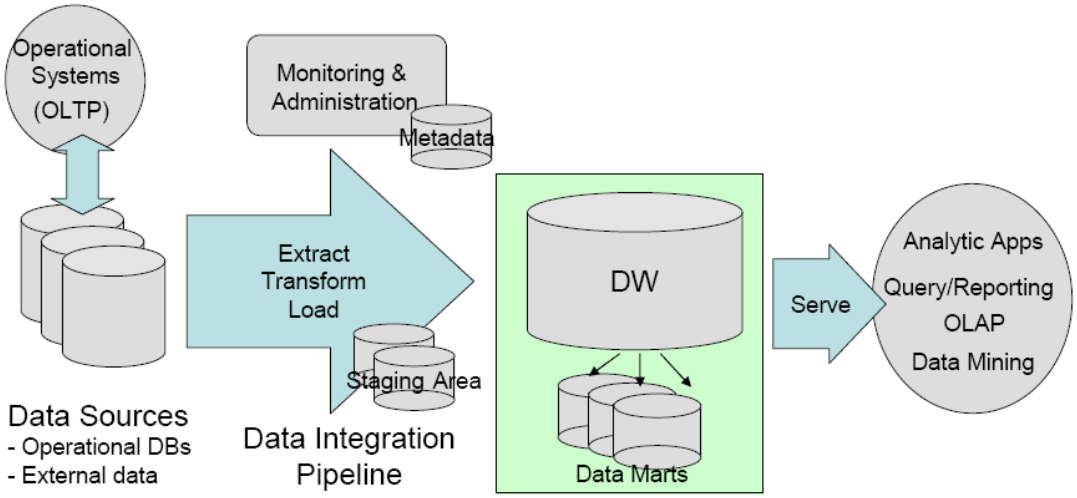


Figure 4.8.2: Next generation of BI Architecture.

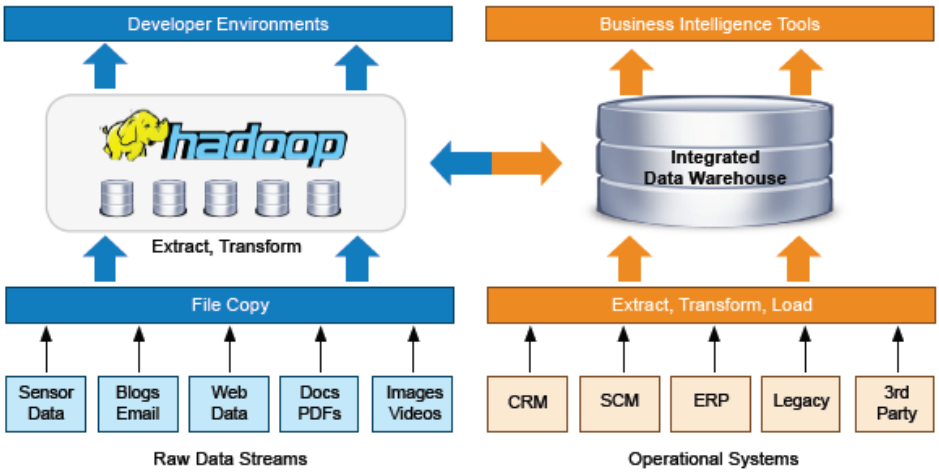


Figure 4.8.3: Example of data architecture of a company.

The following table makes a concise comparison between the two platforms:

Requirement	Data Warehouse	Hadoop
Low latency, interactive reports, and OLAP	●	
ANSI 2003 SQL compliance is required	●	
Preprocessing or exploration of raw unstructured data		●
Online archives alternative to tape		●
High-quality cleansed and consistent data	●	
100s to 1000s of concurrent users	●	●*
Discover unknown relationships in the data	●	●
Parallel complex process logic		●
CPU intense analysis	●	●
System, users, and data governance	●	
Many flexible programming languages running in parallel		●
Unrestricted, ungoverned sand box explorations		●
Analysis of provisional data		●
Extensive security and regulatory compliance	●	
Real time data loading and 1 second tactical queries	●	●*

Hadoop and DW can often work together in a simple chain of information supply. When it is about large volume data Hadoop is superior in the handling of unstructured and complex flows offering great flexibility of programming. DWs also manage large volume structured data offering interactive representation by using BI tools. Obviously a symbiotic relationship is formulated at high speeds. Some differences are clear and the workload or the defining of data that runs better on one or the other platform depends on the organization and the circumstances of use.

Chapter 5 Spatial Data and Hadoop

5.1 Spatial Data and Hadoop

Most spatial queries involve geometric computations which are often compute- intensive. Geometric computation is not only used for extracting measurements or generating new spatial objects, but also used as logical operations for topology relationships. While spatial filtering through minimal boundary rectangles can be accelerated through spatial access methods, spatial refinements such as polygons intersection verification are highly expensive operations.

The last years, MapReduce based systems have emerged as a scalable and cost effective solution for massively parallel processing. Hadoop, the open source implementation of MapReduce, has been successfully applied in large scale internet services to support big data analytics.

However, most of these MapReduce based systems either lack spatial query processing capabilities or have very limited spatial support. While the “map” and “reduce” programming

model fits nicely with large scale problems which are often divided through data partitioning , spatial data is multi-dimensional and spatial queries are intrinsically complex which often rely on effective access methods to reduce search space and alleviate high cost of geometric computations. Consequently there is a significant step required on adapting and redesigning spatial query methods to take advantage of the Map Reduce computing infrastructure.

Hadoop is ill-equipped for supporting spatial data as its core framework is unaware of spatial data properties. Existing attempts to process spatial data on Hadoop focus mainly on specific data types and operations and range queries on trajectories.

5.2 Applications on Spatial Data Storage Based on Hadoop Platform

- *Hadoop-GIS (Emory University)*

The scope of the system is to deliver a scalable, efficient, expressive spatial querying system for efficiently supporting analytical queries on large scale spatial data and to provide an acceptable solution that can for daily operations. Hadoop-GIS integrate a native spatial query engine with MapReduce, where spatial queries are implicitly parallelized through MapReduce through space partitioning, and spatial indexing. This system presented from Emory University at the 39th International Conference on Very Large Databases on August 26th 2013.

By integrating the framework with Hive, Hadoop-GIS provides an expressive spatial query language by extending HIVEQL with spatial constructs, and automate spatial query translation, optimization and execution. Hadoop-GIS supports fundamental spatial queries (selection, join , projection and aggregation), and complex queries such as spatial cross-matching (large scale spatial join) and nearest neighbor queries.

The core components (Figure 5.2.1) of Hadoop-GIS are:

- Spatial Query Translator pares and translates SQL queries into an abstract syntax tree. They extended the HiveQL translator to support a set of spatial query operators, spatial functions and spatial data types.
- Spatial Query Optimizer takes an operator tree as an input and applies rule based optimization such as predicate push down or index-only query processing.
- Spatial Query Engine is a stand-alone spatial query engine which supports following infrastructure operations: 1) spatial relationship comparison , such as intersects, touches, overlaps, contains, within, disjoint, 2)spatial measurements , such as intersection, union, convexHull, distance, centroid, area etc.3)spatial access methods for efficient query processing such as R*-Tree and Voronoi Diagram building and querying these structures. The engine is compiled as a shared library and can be easily shared by all cluster nodes.

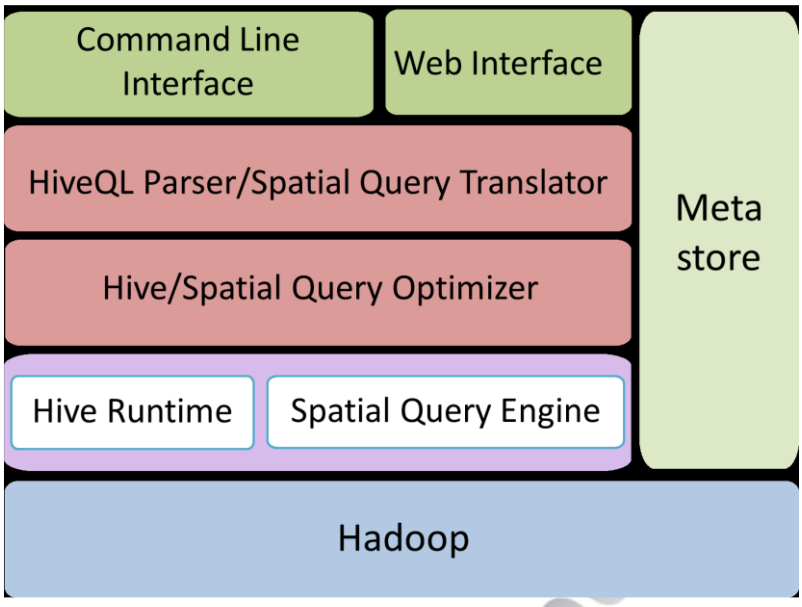


Figure 5.2.1: Architecture of Hadoop-GIS system (Source: Emory University)

Major differences between Hive and Hadoop-GIS in the logical plan generation step. If a query does not contain any spatial operations, the resulting logical query plan is exactly the same as the one from generated from Hive. If the query contains spatial operations, the logical plan is regenerated with special handling of spatial operations.

An interesting project that facilitate the process of spatial data from a Hadoop system. This requires a scalable architecture that can query scientific spatial data at massive scale. Another requirement is to support queries on cost effective architecture such as commodity clusters or cloud environments.

- *SpatialHadoop (University of Minnesota)*

SpatialHadoop is a MapReduce framework with native support for spatial data; available as open-source at http://spatialhadoop.cs.umn.edu/ and presented by Minnesota University at the 39th International Conference on Very Large Databases on August 26th 2013. SpatialHadoop is a comprehensive extension of Hadoop (around 12,000 lines of code inside Hadoop code base. As a result, SpatialHadoop works in a similar way to Hadoop where programs are written in terms of map and reduce functions and hence existing Hadoop programs can run as is on SpatialHadoop pushes its spatial constructs in all layers of Hadoop, namely, language, storage, MapReduce and operations layer (Figure 5.2.2)

- In the language layer, a simple high level language is provided for the simplification the interaction with the system for non-technical users. This language provides a built-in support for spatial data types, spatial primitive functions, and spatial operations (Point, Rectangle , Distance etc).
- In the storage layer, a two-layered spatial index structure is provided where the global index partitions data across nodes while the local index organizes data in each node. The global index is used for preparing the MapReduce job while the local indexes is used for processing map task.

- In the MapReduce layer , two new components are added to allow MapReduce programs to access indexed files as input, namely, SpatialFileSplitter and SpatialRecordReader. The SpatialFileSplitter exploits the global index by pruning partitions that do not contribute to the query answer, while the SpatialRecordReader exploits the local index to efficiently access records within each partition.

- The Operations layer contains a number of spatial operations (range query, kNN and spatial join), implemented using the indexes and new components in the MapReduce layer. Other spatial operations can be added in a similar way.

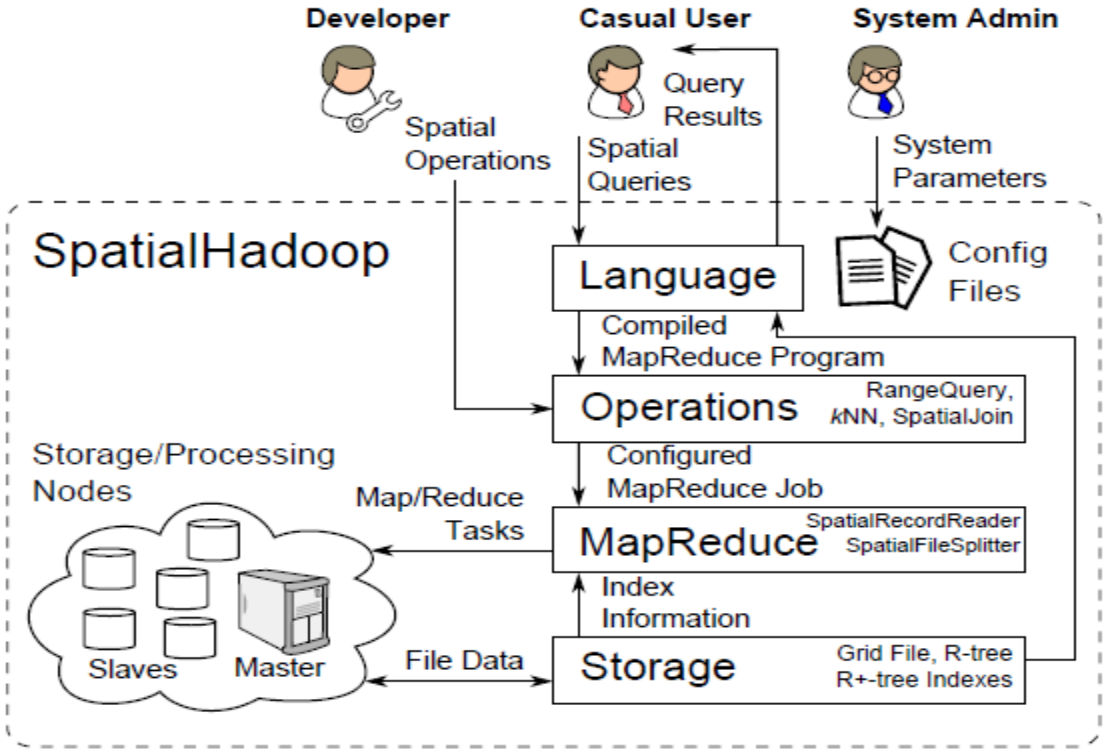


Figure 5.2.2: SpatialHadoop architecture(Source: Minnesota University)

➢ *GIS Tools for Hadoop*

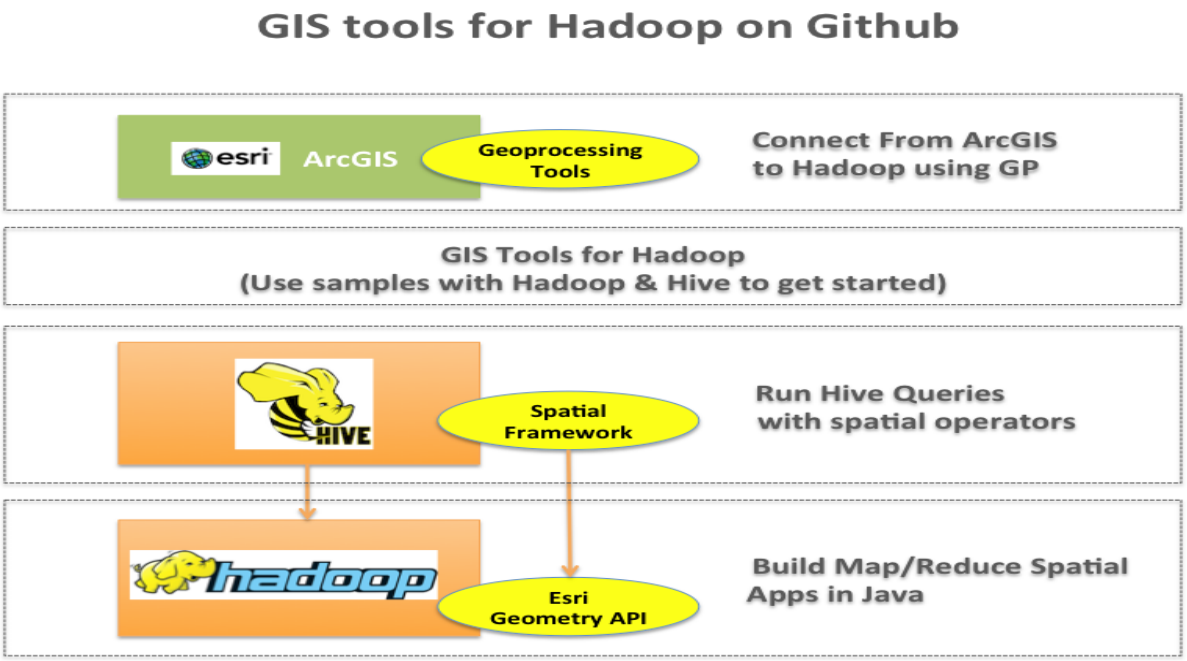
On March 25 , 2013 David Kaiser and his Big Data crew released the GIS Tools for Hadoop Project on GitHub. The goal of this effort is to include to Hadoop spatial data and spatial analysis. The project contains:

- Esri Geometry API for java: A generic geometry library, can be used to extend Hadoop core with vector geometry types and operations, and enables developers to build MapReduce applications for spatial data

- Spatial Framework for Hadoop: extends Hive and is based on the Esri Geometry API, to enable Hive Query Language users to leverage a set of analytical functions and geometry types. In addition to some utilities for JSON used in ArcGIS(A platform for designing and managing solutions through the application of geographic knowledge).

- Geoprocessing Tools for Hadoop toolkit allows users, who want to leverage the Hadoop Framework, to do spatial analysis on spatial data.

The project supports processing of simple vector data (Points, Lines, Polygons) and basic analysis operations, e.g. relationship analysis on that data running in a Hadoop distributed processing environment.



CHAPTER 6 HADOOPDB

6.1 HadoopDB main components

HadoopDB is an open-source system that connects multiple individual simple database nodes (such as PostgreSQL or MySql), which have been developed in a cluster, using Hadoop as a coordination tool that manages the performance of jobs, as well as network communication. This is the first work that uses the MapReduce Framework together with the RDBMS for big data analytics. HadoopDB stores data in distributed file system (dfs) and , at query processing time , loads the data in the DFS to a local database in each node , and then , processes queries by local DBS. Therefore, HadoopDB resembles a shared-nothing parallel database in which Hadoop provides all the necessary services for escalation to thousands of nodes.

It was created by scientists of the Yale University and was launched for the first time on the 27th of August 2009 by its co-creators Azza Abouzeid and Kamil Bajda-Pawlikowski n Lyon, France. HadoopDB has been created exclusively with open-source components, like Hive which provides its system with a Sql interface.

Business databases are the natural area of implementation for HadoopDB. Due to its superiority in fault tolerance, data escalation is more smooth as compared to the existent parallel databases. Furthermore, the workflow in business database systems is mostly concentrated on the process of reading, which addresses analytical queries to a complex scheme. In order to achieve good performance for a query, the dataset needs to be crucially

prepared, through data partitioning and optimization of queries that refer to table joins. HadoopDB combines Hadoop scalability with high database performance with relation to the analysis of structured information.

HadoopDB extends Hadoop's characteristics providing the following four components:

- Database Connector
  It is the means for communication between independent database systems residing on nodes in the cluster and JobTrackers. Each MapReduce job supplies the Connector with an SQL query and the connection parameters (such as which JDBC driver to use, query fetch size and other query parameters). The Connector connects to the database, executes the query and returns results as key/value pairs. As compared to the framework Hadoop , databases are data sources similar to data blocks in HDFS.
- Data Loader
  It manages the parallel load of data from a file on the databases systems with regard to:
  - Items related to the size and the potential repartitioning of a node.
  - Repartitioning of data on all nodes based on a given partition key upon loading.
  - Partitioning of data on a single node into multiple smaller chunks.
  - Bulk-loading node partitions.

  Data Loader consists of two components:
  - Global Hasher Executes a specific MapReduce job via Hadoop that reads in raw data files stored in HDFS and repartitions them into as many parts as the number of nodes in the cluster
  - Local Hasher Copies a partition from HDFS into the local file system of each node and secondarily further partitions the file into smaller sized parts based on the maximum partition size setting
- Catalog
  The catalog maintains/stores maintain information about the databases and is crucial for the creation of the query. It includes the following:
  - Connection parameters, such as database location, driver class and credentials.
  - Metadata, such data partitioning properties and replica locations. Data sets contained in the cluster.
- SQL MapReduce planner which extends Hive providing an SQL interface on HadoopDB.
- Local hasher Copies data partitions from HDFS to the local file system. It also partitions files into smaller chunks.

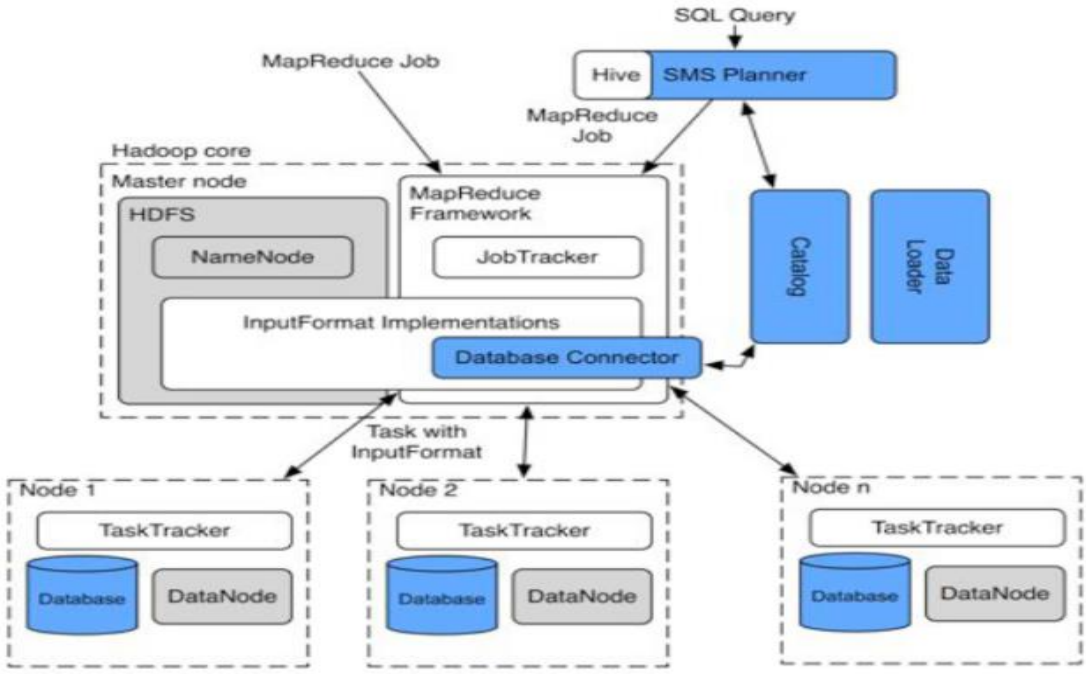


Figure 6.1.1: HadoopDB architecture

One of the biggest drawbacks of HadoopDB that makes it unsuitable in the realm of large scale data processing is the lack of fault tolerance as the data layer. Partitioning of the raw data, uploading them onto individual database nodes is no more supervised by Hadoop framework. While HadoopDB integrates the power of efficient DBMS technology with MapReduce, yet it seems impractical to employ this system to carry out large scale data processing. It shall be a great advancement towards large scale data processing if HadoopDB is improved to possess fault tolerance at data layer too, just like Hadoop do. More, HadoopDB has the following drawbacks because the DBMS that process queries are separated from the DFS that store the data. First, there is storage overhead due to redundant storage of data in both the DFS and local databases. Second HadoopDB causes performances degradation by re-loading the DFS data to local database when processing queries that cannot be processed using the current snapshot of the local databases. Third, HadoopDB, being a shared nothing architecture, does not support queries that require internode communication and does n't integrates the relation DBMS with the DFS as a single system for big data analytics.

6.2 HadoopDB Example

HadoopDB comes with a data partitioner that can partition data into a specified number of partitions. The idea is that a separate partition can be bulk-loaded into a separate database and indexed appropriately. The code to load the data set into our a Single Postgres Note is:

```
Hadoop fs –get /data/part-001 my_file

Postgres>create database grep0;

Postgres> use grep0;

Postgres> create table  grep (key1 char(10), field char(10));

Load data local infile  'my_file' into table grep fields terminated by '|' (key1, field);
```

Above we have loaded data into both HDFS and Postgres. Then in order to generate the XML file and store it in HDFS:

java –cp $HADOOP_HOME/lib/hadoopdb.jar edu.yale.cs.hadoopdb.catalog.SimpleCatalogGenerator > Catalog.properties

hadoop dfs –put hadoopDB.xml hadoopDB.xml

We ‘m using the grep task (class) from the HadoopDB paper (http://hadoopdb.sourceforge.net/doc/) to search for a pattern in the data that we loaded earlier:

Java –cp $CLASSPATH:hadoopdb.jar edu.yale.cs.hadoopdb.benchmark.GrepTaskDB \

-pattern %wo% -output Padraig –hadoop.config.file HadoopDB.xml

10/09/13 18:01:41 INFO exec.DBJobBase: grep_db_job

10/09/13 18:01:41 INFO exec.DBJobBase: SELECT key1, field FROM grep WHERE field LIKE '%%wo%%';

10/09/13 18:01:41 INFO jvm.JvmMetrics: Initializing JVM Metrics with processName=JobTracker,

sessionId=

10/09/13 18:01:41 WARN mapred.JobClient: Use GenericOptionsParser for parsing the arguments.

Applications should implement Tool for the same.

10/09/13 18:01:41 INFO mapred.JobClient: Running job: job_local_0001

10/09/13 18:01:41 INFO connector.AbstractDBRecordReader: Data locality failed for

hadoop1.localdomain

10/09/13 18:01:41 INFO connector.AbstractDBRecordReader: Task from hadoop1.localdomain is connecting

to chunk 0 on host localhost with db url jdbc:mysql://localhost:3306/grep0

10/09/13 18:01:41 INFO connector.AbstractDBRecordReader: SELECT key1, field FROM grep WHERE field

LIKE '%%wo%%';

10/09/13 18:01:41 INFO mapred.MapTask: numReduceTasks: 0

10/09/13 18:01:41 INFO connector.AbstractDBRecordReader: DB times (ms): connection = 245, query

execution = 2, row retrieval = 36

10/09/13 18:01:41 INFO connector.AbstractDBRecordReader: Rows retrieved = 3

10/09/13 18:01:41 INFO mapred.TaskRunner: Task:attempt_local_0001_m_000000_0 is done. And is in the

process of commiting

10/09/13 18:01:41 INFO mapred.LocalJobRunner:

10/09/13 18:01:41 INFO mapred.TaskRunner: Task attempt_local_0001_m_000000_0 is allowed to commit

now

10/09/13 18:01:41 INFO mapred.FileOutputCommitter: Saved output of task

'attempt_local_0001_m_000000_0' to file:/home/hadoop/padraig

10/09/13 18:01:41 INFO mapred.LocalJobRunner:

10/09/13 18:01:41 INFO mapred.TaskRunner: Task 'attempt_local_0001_m_000000_0' done.

10/09/13 18:01:42 INFO mapred.JobClient: map 100% reduce 0%

10/09/13 18:01:42 INFO mapred.JobClient: Job complete: job_local_0001

10/09/13 18:01:42 INFO mapred.JobClient: Counters: 6

10/09/13 18:01:42 INFO mapred.JobClient: FileSystemCounters

10/09/13 18:01:42 INFO mapred.JobClient: FILE_BYTES_READ=115486

10/09/13 18:01:42 INFO mapred.JobClient: FILE_BYTES_WRITTEN=130574

10/09/13 18:01:42 INFO mapred.JobClient: Map-Reduce Framework

10/09/13 18:01:42 INFO mapred.JobClient: Map input records=3

10/09/13 18:01:42 INFO mapred.JobClient: Spilled Records=0

10/09/13 18:01:42 INFO mapred.JobClient: Map input bytes=3

10/09/13 18:01:42 INFO mapred.JobClient: Map output records=3

10/09/13 18:01:42 INFO exec.DBJobBase:

grep_db_job JOB TIME : 1747 ms.

The result is stored in HDFS and in an output directory that just specified.

$cd backup

$cat part-001

~k~MuMq= w0000000000{XSq#Bq6,3xd.tg_Wfa"+woX1e_L*]H-UE%+]L]DiT5#QOS5<

vkrvkB86i0000000000.h9RSz'>Kfp6l~kE0FV"aP!>xnL^=C^W5Y}lTWO%N4$F0 Qu@:]-N4-(J%+Bm*wgF^-{BcP^5NqA

]&{`H%]1{E0000000000Z[@egp'h9! BV8p~MuIuwoP4;?Zr' :!s=,@!F8p7e[9VOq`L4%+3h.*3Rb5e=Nu`>q*{6=7

CHAPTER 7 Hadoop and Partitions

7.1 Strategy to distribute moving data across cluster nodes

Input file fragments distributed by the initial data placement algorithm might be disrupted due to the following reasons: (1) new data is appended to an existing input file; (2) data blocks are deleted from the existing input file; and (3) new data computing nodes are added into an existing cluster. To address this dynamic data load-balancing problem, we implemented a data redistribution algorithm to reorganize file fragments based on computing ratios. The data redistribution procedure is described as the following steps. First, like initial data placement, information regarding the network topology and disk space utilization of a cluster is collected by the data distribution server. Second, the server creates two node lists: a list of nodes in which the number of local fragments in each node exceeds its computing capacity and a list of nodes that can handle more local fragments because of their high performance. The first list is called over-utilized node list; the second list is termed as under-utilized node list. Third, the data distribution server repeatedly moves file fragments from an over-utilized node to an

underutilized node until the data load are evenly distributed. In a process of migrating data between a pair of an over-utilized and underutilized nodes, the server moves file fragments from a source node in the over-utilized node list to a destination node in the underutilized node list. Note that the server decides the number of bytes rather than fragments and moves fragments from the source to the destination node. The above data migration process is repeated until the number of local fragments in each node matches its speed measured by computing ratio.

7.2 Optimizing Data Partitioning for Data-Parallel Computing

Recent developments in distributed data processing systems (MapReduce, Hadoop and Dryad) have significantly simplified the development of large-scale data optimized techniques and data partitioning applications. In these systems, data partitioning is the keystone for successful scalability in a large number of clusters. Still, data partitioning techniques used by systems are very primitive and cause serious performance failure.

- The use of hash functions or a set of a series of equidistant keys for data hash-partitioning often gives asymmetric divisions regarding data or low performance in the computation of results or even failures.
- Balanced workload is not the only factor that needs to be considered in achieving optimal performance. The amount of partitioning is another important factor. There is often a sore point between the amount of computations per partition and the movement of the network which relies on the number of nodes making the detection of an ideal spot difficult.
- On multiple computation levels, data computations can be completed at later stages. It is often hard to predict such skews before running the program.

Even on the same program, imported data often change and present different characteristics (for example, production of statistics based on log files) and they demand partitioning schemes which are adjusted to the change of data towards optimal performance. Obviously, performance in a system of parallel data depends on many significant parameters, like configurations and adjustments of infrastructures, as well as job scheduling.

Therefore, data segmentation affects several aspects with relation to the way in which a job will run on the cluster, including the simultaneous workload for every top, as well as the traffic of the network among tops.

Hence, data partitioning is a crucial factor that can be exploited by users in order to achieve good performance of the system. Many MapReduce jobs fail due to issues within data partitioning. Any improvement in the types of data partitioning contributed significantly to the enhancement of the usability of these systems.

7.3 Referential Partitioning

In general, database system developers spend a lot of time optimizing the performance of joins which are very commonly used and are costly in operation.

Partitioned joins are especially costly for the system performance in Hadoop, because they require an additional MapReduce job for the additional data segmentation on a join key. Typically joins are computed within a database system which involves far fewer readings and writings on the disk than are computed across a MapReduce job on Hadoop. Hence, for reasons related to the performance of the system, HadoopDB chooses to compute join creation processes only inside a database system, while the outcome will be deployed on each node.

In order to be performed exclusively inside the database in HadoopDB, the join must be local, for example each node must join data from tables stored locally, without any need for data to be transferred across the network. When data needs to be sent across a cluster, Hadoop undertakes query processing and the join is not accomplished inside the database system. If two tables are partitioned based on attributes of a join (for example, both employee and department tables are connected through the department key/id), then a local join is possible since each single database node can create a join on the partitioned data without affecting the rest of the data stored on other nodes.

Generally, traditional parallel database systems prefer local joins over repartitioned joins since this approach is less expensive for the system performance. This discrepancy between local and repartitioned joins is even greater in the case of HadoopDB due to the difference in performance regarding the implementation of joins between Hadoop and the database. For this reason, HadoopDB is willing to sacrifice certain performance benefits, such as quick load time, in exchange for local joins.

In order to create the desired joins in a single node database system inside HadoopDB, the method chosen is "aggressive hash-partitioning". Typically, database tables include partitioned data based on an attribute from the original table. However, this method limits the level of co-partitioning, since tables can be connected with each other through multiple primary/foreign-key references. For example, in TPC-H database, the table for line item data contains a foreign-key projected on the order table through the order key, while the order table contains a foreign-key projected on the customer table through the customer key. If the line item table could be partitioned based on the customer who made the order, then any of the join combinations of the customer, order and line item tables could be local to any node. What is more, since the line item table does not contain the customer key, direct partitioning is impossible. HadoopDB has, therefore, been extended to support referential data partitioning. Although a similar technique was recently launched by Oracle 11g, it serves a different purpose with relation to the partitioning data scheme of the HadoopDB platform which facilitates the creation and access of joins to a shared-nothing network.

For tables that are not co-partitioned, joins are generally performed using the MapReduce platform. This usually put into effect in the Reduce phase of the job. The Map phase reads each partitioned table and for each tuple it extracts the join attributes that are going to be used to automatically repartition tables between the Map and Reduce phases. What is more, the same Reduce job is responsible for processing all partitioned data tuples with the same join key.

There are two join optimization methods; direct join and broadcast join. The former is applicable when one of the tables has already been partitioned based on the join key. The join can be processed locally in every node. The broadcast node is used when a table is much larger than the other. The large table could be left in its original position, while the entire small table should to be send to every node in the cluster. Then, each partition of the larger table can be locally connected to the smaller table.

Unfortunately, implementing direct and broadcast joins in Hadoop requires computing the join during the Map phase. This is not a trivial job since reading multiple data subsets requires multiple passes and does not fit well into the Map sequential scan model. Furthermore, HDFS does not promise to keep different data subsets co-partitioned between Map-Reduce jobs. In any case, a map job cannot assume that two different partitioned data subsets that have used the same hash function have actually been stored on the same node.

The implementation of the broadcast join in these systems is as follows:

Each Map worker reads smaller tables from HDFS and stores it in a hash table in the memory. This causes the replication of each small table to every local node. What follows is a sequential scan of the larger table. As is the case in a standard simple hash join, the in-memory hash map detects each tuple of the larger table in order to search for a matching key value. Since the join is computed in the Map phase, it is called a Map-side join.

The second type of join (directed join) is enabled by split execution environments. Hadoop runs a standard MapReduce job to repartition the second table. First, we look up in the Hadoop catalog on how the first table was distributed and uses the same function to reform the second table. Each selection operations on the second table are processed in the Map phase of this job. The Output Feature tool of Hadoop is then used to circumvent HDFS and write the output of the data repartitioning directly into the database systems located on each node. Hadoop provides native support for retaining co-partitioned data between jobs. Therefore, since both tables partitioned based on the same attributes inside the Hadoop storage system, the next MapReduce job can compute the join entirely within the database systems.

7.4 Spatial Partition Strategy

7.4.1 Decomposition of the Object Into partitions

For a large random collection of spatial objects, we define universe [4] as the minimum bounding rectangle (MBR) that encloses all objects in the collection. In order to distribute the data sets across shared-nothing database sites following the discussion above, it is required to decompose the universe into smaller regions called partitions. We need to have some metadata first which is essential for decomposition of the universe into partitions. The metadata comprises of the dimensions of the universe which is determined by manual analysis of the data set or through some hand-coded scripts. This is static and permanent information, once computed, need not be computed again through the life time of data set.

Number of partitions into which the universe is to be spatially decomposed depends on the maximum table size a database can process efficiently without using temporary disk buffers (another parameter essential for data partitioning). If the total no of spatial objects in the universe is N, and the average no of objects that can be stored in a database table which avoids disk buffer access during query execution is M, then number of partitions to be made is roughly N/M. The dimensions of partitions boundaries are predicted roughly by dividing the universe into smaller rectangular regions of equal sizes. Partitioning of spatial data sets is done by testing the spatial relationship between partitions and MBR of spatial object as per the predicate condition, say overlap in our case. The spatial objects which qualifies the predicate

with the partition(s) becomes a member of that partition(s). This step produces candidates which are a superset of the actual result.

The Figure 6.4.1 shows the decomposition of spatial into four partition. Each partition consists of the spatial objects belonging to a particular partition resides on a single database site in Distributed DBMS.

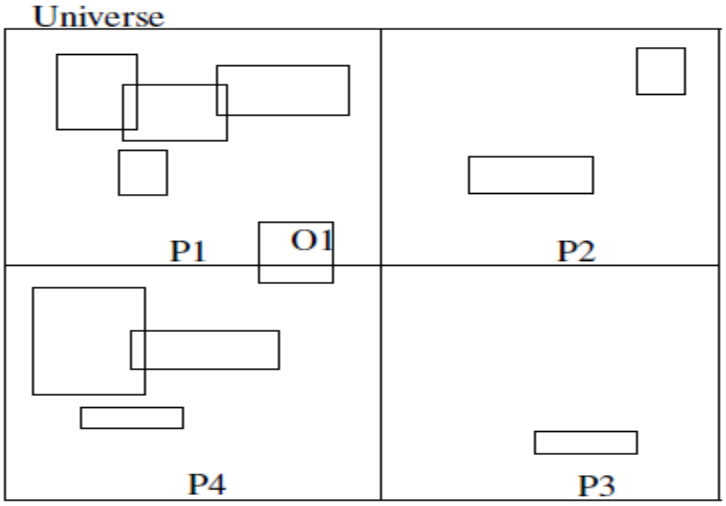


Figure 7.4.1 Decomposition of the Universe into Partitions Source: Large Spatial Data Computation on Shared-Nothing Spatial DBMS Cluster via MapReduce, Abhishek Sagar

7.4.2 Partition Skew & Load Balancing

In practice, the distribution of spatial features over 2D spatial space is generally not even.

For example, there are more roads in cities than in the rural areas. Therefore, the distribution

of spatial objects into partitions may be imbalanced. Figure 6.4.1 1 shows that partition P3 consist of the least number of spatial objects whereas partition P1 and P4 are densely populated. This situation is called Partition Skew and is not uncommon. Since each partition corresponds to the tables residing on the same database site, this uneven distribution results in tables residing on different database sites to vary from each other in size. Consequently, different amount of query computation is carried out on different cluster nodes, thus resulting an increase in the overall job execution time. The overall execution time of the job is decided by the time taken by the cluster node which finishes its share of computation after all cluster nodes have. Therefore, we need Load Balancing for balanced distribution of objects among partitions. To deal with the problem of partition skew, a tile based partitioning method is used for balanced distribution of objects among partitions. This method involves the decomposition of universe into N smaller partitions called Tiles where N >>P (number of partitions). There is also many-to-one mapping between tiles and partitions. All spatial objects that gives positives overlap test with the tile(s) is copied to the partition(s) the tile(s) maps to. Larger the number of tiles the universe is decomposed into, more uniform distribution of objects is among partitions.

In Figure 7.4.2 below, the universe is decomposed into 48 tiles. We have shown the decomposition of only one partition P1 into tiles numbered from 0 to 11. Likewise other partitions are also decomposed in the same manner (not shown in the figure). Tiles are mapped to a partitions in Round Robin fashion. Some spatial objects that are spatially enclosed with in

this partition are now mapped to other partitions. For example, some spatial objects of partition P1 which overlaps with tile 2 and 5, will now be a member of partition P3 and P2 respectively. In the same manner, some spatial objects from other partitions are also mapped to partition P1. This results in the uniform distribution of spatial objects among partitions. Though, this strategy is simple to implement, but balancing the distribution of spatial objects across partitions via this strategy suffer from following drawbacks :

- with the decomposition of each partition further into smaller tiles, the number of spatial objects overlapping across multiple tiles would increase, resulting in mapping of a spatial object to multiple partitions, thereby resulting in profound duplication.
- This strategy is somewhat of adhoc nature and unpredictable.
- Predication of the appropriate value of N (No. of Tiles) is difficult.

9 P2	10 P3	11 P4
6 P3	7 P4	8 P1
3 P4	4 P1	5 P2
0 P1	1 P2	2 P3

Figure 7.4.2: Tile Based Partitioning Scheme. Source: Large Spatial Data Computation on Shared-Nothing Spatial DBMS Cluster via MapReduce, Abhishek Sagar

7.4.3 Spatial Data Partitioning Using Hilbert Curve

Hilbert Space Filling Curve (HSFC) is a continuous fractal space-filling curve first described by German mathematician David Hilbert in 1891 [23]. Hilbert curve are useful in the domain of spatial data partitioning because they give a mapping between 1D and 2D space that fairly well preserves spatial locality. If (x,y) is the coordinates of a point within the unit square, and d is the distance along the curve when it reaches that point, then points that have nearby d values will also have nearby (x,y) values. This is true in the opposite direction also, that is, nearby (x,y) points will also have nearby d values. Figure 4.5 shows the division of a 2D space with the increase in the recursion depth of Hilbert curve.

We first define the term Hilbert Value. Hilbert Value [4] of an arbitrary point p in 2D space is define as Euclidean distance between p and p' where p' is the point lying on Hilbert curve and nearest top. To keep the discussion simple, let us assume we have the following two functions:

- double [ ] GenerateHilbertCurve (m, Universe Dimensions) : The function generate set

of 2D points called Hilbert Points which decompose the universe as shown in figure above.

m is the recursion depth.

- Double ComputeHilbertValue (point p) : The function return the Hilbert Value of a point p in 2D space.

Partitioning of spatial data sets according to this scheme requires to compute the Hilbert value of all spatial entities in the data set. Hilbert Value for spatial object obj is computed as follows:

Objhv = ComputeHilbertValue(Center(MBR(geom(Obj))))

It defines Vg as the average volume of all spatial entities on each disk in bytes. N is the number of disks in the system and Bj is the volume of spatial entities on jth disk, initially initialized to 0 for all j = 0 to N-1; n is the number of all spatial entities to be partitioned; Vi is the size of the ith spatial entity in bytes; i takes value from 0 to n-1.

Procedure for data partitioning using HSFC approach is presented as follows:

- Construct Hilbert Curve within the universe, for each spatial entity in the data set, compute Hilbert value. As a general rule, if n is the number of spatial entities in the universe, then n <22m, where m is the recursion depth of the Hilbert curve to be generated.
- Sort the spatial entities in increasing order according to Hilbert value.
- Beginning from i=0, j=0, put the ith spatial object to jth disk. Do Bj = Bj + Vi
- Compare Bj and Vg. If Bj is smaller than Vg, then i = i+1, otherwise j = j+1 ; repeat this step until i = n-1.

In this approach of data partitioning, each spatial entity is assigned to one and only one disk, therefore there is no duplication in this case. Spatial locality is preserved due to assignment of disk to spatial entities in increasing order of Hilbert value

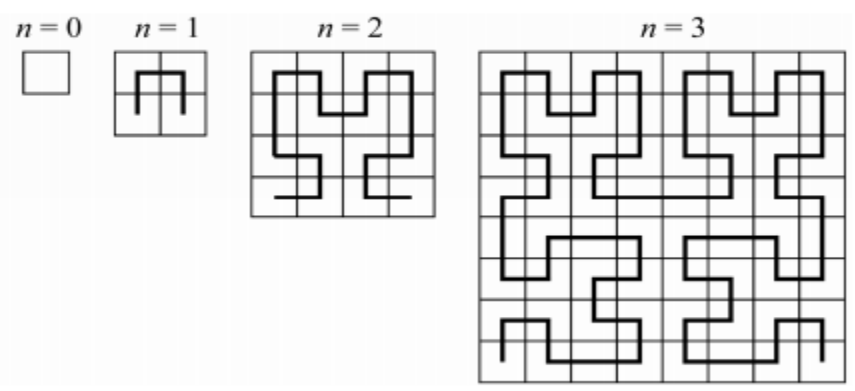


Figure 6.4.3: Hilbert Curve Space Filling Curve Source: Large Spatial Data Computation on Shared-Nothing Spatial DBMS Cluster via MapReduce, Abhishek Sagar

7.4.4 Spatial Index and Hadoop

One essential requirement for spatial queries is fast response. This is important for both exploratory studies on massive amounts of spatial data with a large space of parameters and algorithm, and decision making in enterprise or healthcare applications. However, the mismatch between the large data blocks in HDFS for batch processing and the page based random access of spatial indexes makes it difficult to pre-store spatial indexes on HDFS and retrieves it later for queries. To support indexing based spatial queries, we combine two approaches: i) global spatial indexes for regions and tiles and on demand indexing for objects in tiles. Global region indexes are small due to their large data granularity and ii) local spatial index that organizes data inside each node. They can be pre-generated and stored in a binary format in

HDFS and shared across cluster nodes through Hadoop distributed cache mechanism.. The global index is kept in main memory of the master node while each local index is stored as one file block (typically 64MB) in ever slave node.

An index is constructed through a MapReduce job that runs in three phases, namely, partitioning, local indexing, and global indexing. During local indexing phase a local index is created for each partition separately and flushed to a file with one HDFS block which is annotated y the MBR of the partition. Since each partition has a fixed size (64 MB) , the local index is constructed in memory before it is written to disk in one shot. In case of global indexing where the files containing local indexes are concatenated into one big file and a global index is constructed to index all partitions using their MBRs (Map Balanced Reduce) as keys and stored in main memory of master node. In case of system failure, the global index is reconstructed from block MBRs only when needed.

CHAPTER 8 Hive

8.1 Hive Introduction

The MapReduce programming platform used by Hadoop is certainly of very low level and the lack of SQL-like commands and routines forces developers to spend a lot of time developing programs for even simple analysis. What is more, Hadoop is not very easy to use for developers who are not familiar with map-reduce concepts. It requires writing specialized code for each operation, even for a simple job like the recovery of the number of files from a log or the average from certain columns. Finally, the code generated is hard to maintain and reuse in different applications.

Apache Hive, a NoSQL database system that runs over Hadoop, overcomes these limitations and provides a simple, satisfactory interface tool. Hive efficiently meets the challenges of storing, managing and processing large volume data, whose management with traditional RDBMS solutions can be tricky. Hive facilitates data summarization, ad-hoc queries and the analysis of large volume datasets in Hadoop compatible file systems. It provides a mechanism to project and expand this data structure using an SQL-like language called HiveQL. It addresses data analysts with good knowledge of SQL language, who need to create ad hoc queries, summaries and data analyses on a Hadoop platform.

8.2 Hive components

The three different Hive components are

- Hadoop cluster
  The cluster of low cost commodity computers on which large volume data set is stored and all processes are performed.
- Metadata Store
  The location where the description of the data structure is kept. The database can be either local or remote, that is accessible to multiple users through the network. The hive metastore data is persisted in an ACID database such as Oracle or MySQL.
- Warehouse Directory

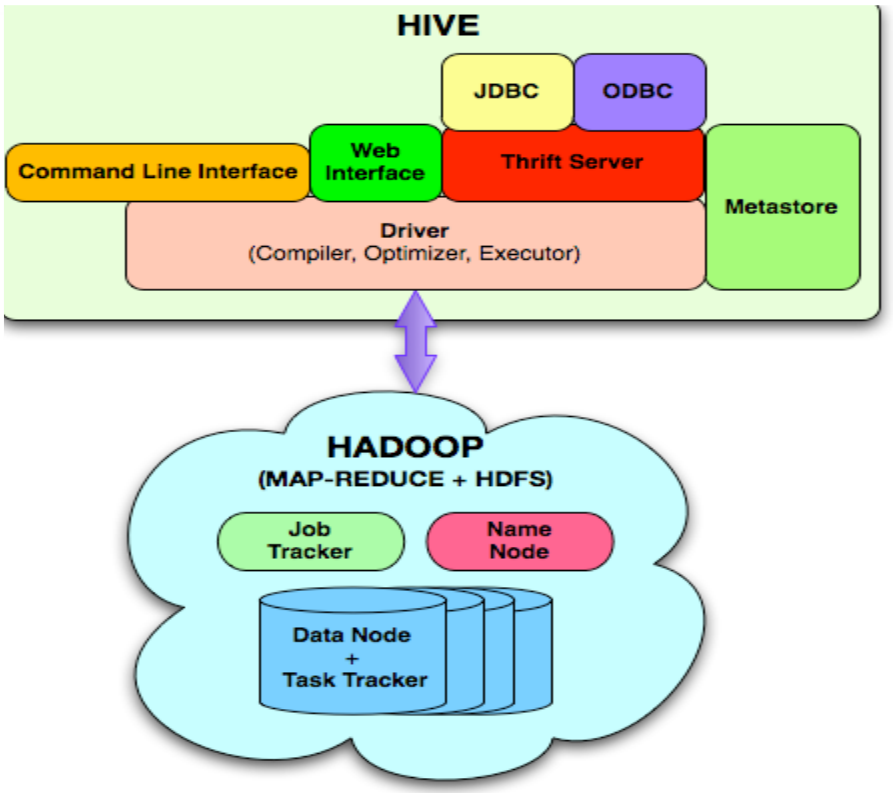


Figure 8.2: Hive Components. Source: Hive A Petabyte Scale Data Warehouse Using Hadoop, Facebook Data Infrastructure Team

It is a location called scratch-pad storage location where Hive permits to store used files. More specifically, it includes

- Newly created tables
- Temporary results from user queries.

For efficient processing/communication, the Warehouse Directory is stored on an HDFS on the Hadoop cluster.

Hive does not have an HDFS format for data storage. Users can write store files on HDFS with any tools/mechanisms that are able to parse the file in a way that can be recognized by Hive. Commonly used file formats include comma delimited text files. Hive provides a dynamic and flexible mechanism for parsing the data file in order to be used by Hadoop and it is called a Serializer or Deserializer.In Hive, tables are stored in the form of files. A table is basically a file directory with the data files. In order to define a table, Hive covers two main concepts which are stored in the Metadata database:

- Where the folder that includes the data files is located.
- How the data for reading the file are parsed.

Depending on the Hive configuration, simply adding more tables to the table folder adds this data to the file system table. In special cases where the table has been partitioned, each partition on the table is a sub-folder within the table's folder.

8.3 Hive architecture

The main components of Hive are:

- External Interfaces Hive provides both user interfaces , such as command line (CLI) and web UI, and application interfaces (API), like JDBC and ODBC.
- Hive Thrift Server extracts a very simple API to execute HiveQL statements. Thrift is a platform for cross-language services, where a service written in one programming language (Java) can also support requests that have been created in other languages. Thrift Hive client is generated in different programming languages and it is used to build common drivers, like JDBC (java) and ODBC (C++).
- Metastore . It is the system catalog that contains metadata from all tables stored in Hive. The metadata is specified during the creation of the table and it is reused every time the table is referenced in HiveQL.
  Metastore consists of the following entities:
  - Database is a namespace for tables. Database 'default' is used for tables for which users have not supplied the name of the database they belong to.
  - Table Metadata contains a list of columns and their types, owner for every table, as well as SerDe information. It also contains all keys supplied by the user. All these attributes offer the ability to store statistics for each table.
  - Partition Each partition can have its own columns and SerDe, as well as storage information. This can be used in the future to support the schema evolution on the Hive database.

The storage system for metastore could be optimized for online transactions and storage information with random access and update mechanisms. A file system like HDFS is not suitable since it is optimized for sequential scanning rather than random access. So, metastore uses either a traditional relational database (such as MySQL, ORACLE) or a file system (like local, NFS, AFS) and not HDFS. Therefore, HiveQL statements which can only access metadata objects are executed with big delays. In any case, Hive explicitly maintains consistency between metadata and data.

- Driver. It manages the life cycle of a HiveQL query during compilation, optimization and execution. After receiving the HiveQL statement, by the thrift server or other interfaces, the driver creates a session handle which is later used to store statistics regarding the process of query execution, like the time of execution, the number of exported documents, etc.
- The Compiler is invoked by the driver upon receipt of an HiveQL statement. The compiler translates this statement into a plan which consists only of metadata operations in case of DDL statements and HDFS operations in case of LOAD statements. For insert data queries, the plan consists of a directed acyclic graph (DAG) of the map-reduce job.
  - The Parser transforms a set of characters (query) into a parse representation
  - The Semantic Analyzer transforms the parse tree into a block-separated internal query representation. The later retrieves schema information of the input table from the metastore directory. Using this information it verifies column names and checks the types of data
  - The Logical Plan Generator converts the internal query representation into a logical plan, which consists of three logical operators
  - The Optimizer performs multiple passes onto the logical plan and rewrites it in several ways:
    - ✓ It combines multiple joins which share the join key into a single multi-way joins and hence a single map-reduce job.
    - ✓ It adds repartition operators (also known as ReduceSinkOperators) for joins, aggregations, as well as custom map-reduce operators. These

repartition operators define the boundaries between the map and reduce process during physical plan generation

- ✓ In the case of partitioned tables, partitions that are not necessary to the query are deleted

➢ Physical Plan Generator converts the logical plan into a physical plan, which consists of a DAG of the map-reduce job. It creates a new map-reduce job for each one of the marker operators, repartitions and connects all in the logical plan. Then it assigns parts of the logical plan enclosed between the markers to map and reduce platforms so as to run map-reduce jobs

- Execution Engine The driver submits the individual map-reduce job from the DAG to the execution engine in topological order. Each dependent task is only executed if all of its prerequisites have been executed. First, a map/reduce job converts the part of the plan into an xml plan. This file is then added to the job cache for the task and instances of ExecMapper and ExecReducers which are built using Hadoop. Each of these classes deserializes the plan.xml file and executes the relevant part of the DAG operator. The final results are stored in a temporary location. At the end of the entire query, the final data is moved to the desired location in the case of DMLS. In the case of queries the data is served as such from the temporary location.

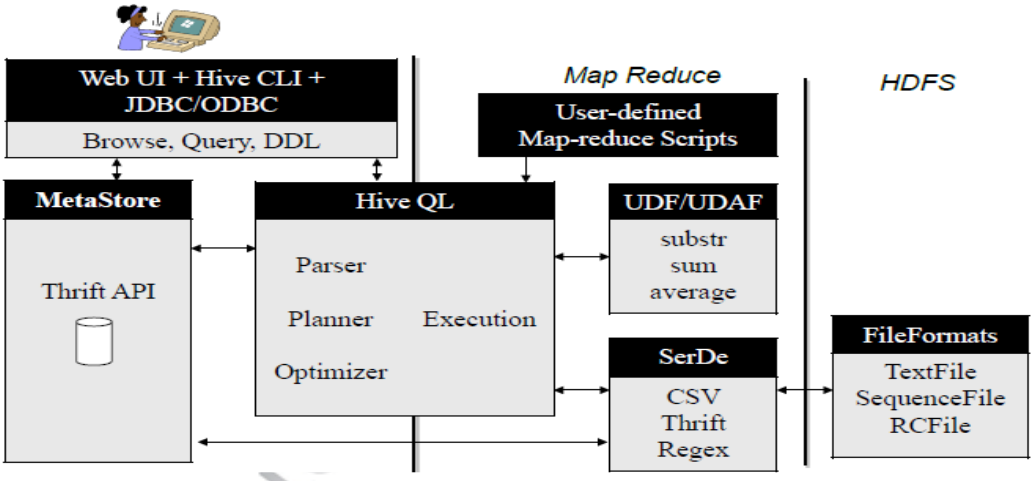


Figure 8.3: Hive open architecture Source: Hadoop Architecture and its Usage at Facebook. Dhruba Borthakur

8.4 Types of HIVE data

Similar to the traditional databases, Hive stores data in tables, where each table consists of the number of rows and each row consists of a specific number of columns. Each column has an associated type of data. This type can be either primitive or complex. At the moment, the following primitive types can be supported:

- Integers –bigint(8 bytes), int(4 bytes), smallint(2 bytes), tinyint(1 byte).
- Floating Point Numbers – float(single precision), double(double precision)
- String
- Bit (ver 1.0.10)

Hive also supports the following complex types:

- Associative arrays-map<key-type, value-type>
- Lists-list<element-type>

- Structs - struct<file-name: field-type, …>

These complex types can generate new types of arbitrary complexity. For example, list<map<string, struct<p1:int, p2:int>> represents a list of associative arrays that map strings to structs that in turn contain two integers: p1 and p2. All these can be put together in a statement to create tables with the desired schema. For example, the following statement creates a table t1 with a complex schema:

CREATE TABLE t1(st string, fl float , li list<map<string,struct<p1:int, p2:int>>);

Queries can access fields within a structured schema using an '.' operator. Values in the associative tables and lists can be accessed with the use of the '[]'operator. In the previous example, t1.li[0] gives the first element of the list, while t1.li[0]['key'] gives the struct associated with table through the key. Finally, the p2 field of this struct can be accessed by the t1.li[0]['key'].p2 command. With these structures Hive is able to support complex types of data defined by the user.

The Hive query language consists of a subset of SQL and some extensions that are useful in the specific context. Hive does not support the import of data in a table or data partition and all data imports overlay existent data.

For example: INSERT OVERWRITE TABLE t1 SELECT * FROM t2;

In fact, this limitation does not cause serious problems since most of the data is loaded on the data base on a daily basis or per hour. It is possible to import data to the new table partition using a key on a specific day or time. Of course in cases of frequent data loads, the number of partitions can be large and hard to manage. On the other hand, the lack of INSERT INTO, UPDATE and DELETE commands in HIVE allows us to use very simple mechanisms in order to manage reading and writing mechanisms without using complex locking protocols. Apart from these restrictions, HiveQL has extensions to support analysis operations expressed as MapReduce programs. This allows advanced users to express complex logic in terms of map reduce programs that are seamlessly connected to HiveQL. The word count example on a table of texts can be expressed using the MapReduce platform in the following manner:

```
FROM (
   MAP doctext USING 'python wc_mapper.py' AS (word, cnt)
FROM docs
CLUSTER BY word
) a
REDUCE word , cnt USING 'python wc_recude.py'.
```

The MAP clause indicates how the input columns (doctext in this case) can be transformed using a user program (in this case 'python wc_mapper.py') in output columns (word and cnt). The CLUSTER BY clause in the sub-query specifies the output columns that are hashed on to distributed the data to the reducers and finally the REDUCE clause specifies the user program to invoke (python wc_reduce.py) on the output columns of the sub-query. Hive allows users to interchange the order of the FROM and SELECT/MAP/REDUCE clauses within a sub-query. Furthermore, HiveQL supports inserting different transformation results into different tables, partitions, hdfs or local directories as part of the same query. This helps in reducing the number of scans that need to be performed on the input data, as shown in the following example:

```
FROM t1
 INSERT OVERWRITE TABLE t2
```

```
 SELECT t3.c2, count(1)
 FROM t3
WHERE t3.c1<=20
GROUP BY t3.c2

INSERT OVERWRITE DIRECTORY '/output_dir'
SELECT 3.c2, avg(t3.c1)
FROM t3
WHERE  t3.c1 > 20 AND t3.c1<=30
INSERT OVERWRITE LOCAL DIRECTORY '/home/dir'
SELECT t3.c2, sum(t3.c1)
FROM t3
WHERE t3.c1 >30
GROUP BY t3.c2
```

In this example, different portions of table t1 are aggregated and used to generate a table t2, an hdfs directory (/output_dir) and a local directory (/home/dir on the user's machine).

Additional of basic types Hive comes with a set of pre-defined User Defined Functions (UDFs) available for use. The hive community has made an effort to make most of the commonly used UDFs available to users. However, often times, the existing UDFS are not good enough for users and they want to write their custom UDF.

Example:

```
Class UDFExample extends UDF {
    Public Text evaluate(Text input){
    Return new Text("Hello " + input.toString());
   }
 }
```

and the use in Hive:

```
hive> ADD JAR hive/lib//hive-extensions-1.0-SNAPSHOT-jar-with-dependencies.jar;
hive> CREATE TEMPORARY FUNCTION helloworld as 'UDFExample';
hive> select helloworld(name) from people limit 10;
```

8.5 Hive data model

Hive supports tables as a fundamental data model. Of course, Hive stores tables as file directories under the /user/hive/warehouse folder. Hive data (not metadata) is spread across Hadoop HDFS DataNodes servers. Typically, each block of data is stored on three different

DataNodes. Relational databases use indexes on columns to speed up the execution of queries on those columns. Hive, instead, uses a concept of partitioned columns. For example, a column whose name field is "state" could partition a table into 50 partitions, one for each field value. In cases of log files, the column that includes time data is the one that is mainly used for the table partitioning. Hive manages partitioned columns differently in comparison to regular data columns as it includes the partitioned columns in the execution of the query, showing greater efficiency and speed. This happens because Hive stores different partitions of the table in different file directories. For example, assuming that there is a table named "users" that has two partitioned columns "name" and "state" (plus the regular columns). The file directory structure will be:

/user/hive/warehouse/users/date=20090901/state=CA

/user/hive/warehouse/users/date=20090901/state=NY

/user/hive/warehouse/users/date=20090901/state=TX

…

/user/hive/warehouse/users/date=20090901/state=CA

/user/hive/warehouse/users/date=20090901/state=NY

/user/hive/warehouse/users/date=20090901/state=TX

…

/user/hive/warehouse/users/date=20090901/state=CA

/user/hive/warehouse/users/date=20090901/state=NY

/user/hive/warehouse/users/date=20090901/state=TX

All data for users in California on September 1, 2009 resides in one directory, while the data for other partitions resides in other file directories. If a query asks about California users on September, 2009, Hive only processes data in the specific directory and ignores data the users table that have been stored in other partitions.

Moreover, the Hive data model includes another possibility, that of buckets, which provide efficient management to queries that can operate well in cases of random samples of data. (For example, in computing the average of a column, a random sample of data could provide a satisfactory approximation. Bucketing divides data into a specified number of files based on the hash value of the bucket column. If we specify 32 buckets based on a user id in our *users table, the complete structure of the file will be:*

/user/hive/warehouse/users/date=20090901/state=CA/part-00000
….
/user/hive/warehouse/users/date=20090901/state=CA/part-00031
/user/hive/warehouse/users/date=20090901/state=NY/part-00000
….
/user/hive/warehouse/users/date=20090901/state=NY/part-00031
/user/hive/warehouse/users/date=20090901/state=TX/part-00000
…..

Each partition will have 32 buckets. Each file in part-00000 ... part-00031 has a random sample of users. The computation of many aggregate statistics remains fairly accurate on a sampled data set. Bucketing is particularly useful for accelerating queries execution.

The query based on which the ability for partition is configured for the specific table is show in the following command:

CREATE TABLE page view(view Time INT, userid BIGINT,

```
                    Page_url STRING, referrer_url STRING,
                    Ip STRING COMMENT 'IP Address of the User')
    COMENT 'This is the page view table'
    PARTIONED BY (dt STRING, country STRING)
    CLUSTERED by (userid) INTO 32 BUCKES
    ROW FORMAT DELIMIED
        FIELDS TERMINATED BY '\t'
        LINES TERMINATED BY \n'
    STORED AS SEQUENCEFILE;
```

8.6 Hive Indexing

If we want to apply indexing using Hive then first expectation might be that with indexing it should take less time to fetch records and it should not launch a map reduce job. Whereas in practice a map reduce job would still be launched on Hive query even though an index is created on hive table. MapReduce jobs runs on the table that holds the index data to get all the relevant offsets into the main table and then using those offsets it figures out which blocks to read from the main table. The biggest advantage of having index is that it does not require a full table scan and it would query only the HDFS blocks required.

We use the following commands to create an index:

```
Hive> create INDEX test_column_index ON TABLE test_table(test_column) as
'org.apach.hadoop.hive.ql.index.compact.CompactIndexHandler' WITH DEFERRED REBUILD;
```

Indexes are built in two phases. In the first phase , index layout and variables are validated at the index creation time. In the second phase , the index table is loaded with relevant data, which takes a major time of building an index.

If PARTITIONED BY clause is omitted from the CREATE INDEX, index spans all the table partitions. Index partitions and table partitions do not have necessarily the same level of granularity.

In the relation databases context, there are a number of structures that can hold the index. They include:

- Primary indexes on sorted file . Key-pointer pairs in an index file
- Secondary Indexes. Find the locations of the desired records in an un-sorted file
- B-trees. Organizes data in a balanced tree.
- Hash Tables. A hash function takes a search key as an argument and computes an integer in the range 0 to B-1, where B is the number of buckets.

Current Hive index structure is pretty analogous to the secondary indexes or sometimes the primary index once the data is sorted on the search key/index as will be explained shortly. Presently, Hive supports two types of indexes, compact and bitmap. The corresponding handles in both index types store the index in a tabular format, i.e, indexes are stored in tables.The compact index, builds a compressed index separated from the data. This means that rather than storing the HDFS location of each occurrence of particular value, , it only stores the addresses of HDFS blocks containing that value.

Bitmap indexes are created for columns with few distinct values, such as gender, Bitmap operations are than used to quickly identify rows that satisfy a combination of conditions on such columns. Bitmap index has the same columns as the compact index in addition to a number of binary bit vectors used to represent the value of the indexed column. Each value in

the vector represents a row and is set to 1 if the value is present in the row , and set to 0 otherwise.

8.7 The use of Hive on Facebook

Hive and Hadoop are used extensively on Facebook for different data processing stages. Currently the database has 700TB of data (which come from 2.1PB of raw data on Hadoop after accounting for the 3 way replication). 5TB of compressed data is added on a daily basis (15TB after replication). Typically, the compression ratio is 1:7 and sometimes even more than that. On any particular day, more than 7,500 jobs are submitted to the cluster and more than 75TB of compressed data is processed. With the continuous growth in the Facebook, the continuous growth in data is evident. At the same time, the cluster has to scale with the continuously growing number of users.

More than half of the total workload is related to adhoc queries, while the rest is related to reporting dashboards. Hive allows this kind of workload on the Hadoop cluster in Facebook because of the simplicity with which adhoc analysis can be done. However, sharing the same resources by adhoc users and reporting users emerges significant challenges due to the unpredictable behavior of adhoc jobs. Often these jobs are not properly tuned and consume valuable cluster resources. As a result, the performance of reporting queries processing, many of which are time critical, is degraded. Resource scheduling by Hadoop is not satisfactory and the only viable solution at present seems to be maintaining separate clusters for adhoc queries and reporting queries.

There is also a wide variety of jobs running on Hive on a daily basis. These may range from simple numeric calculations and generation processes of different kinds of rollups and cubes to more advanced machine learning algorithms. The system is used by beginners as well as advanced users who are able to use it immediately or after several hours of training.

An outcome of heavy usage is the generation of a big number of tables on the database and this has in turn tremendously increased the need for data discovery tools, especially for new users. Generally, the system provides data processing services to software engineers at low cost as compared to the traditional databases infrastructure. If on top of that we add the ability of Hadoop to scale to thousands of nodes, then there is great confidence that the specific infrastructure will have a very positive evolvement.

Figure 8.1 shows the plan of a data insert query in multiple tables. The nodes in the plan are physical operators and their edges represent the flow of data between operators. The last line in each node represents the output schema of each operator. The plan has three map-reduce jobs. The first map-reduce job stores data in two temporary files on HDFS tmp1 and tmp2, which are used by the second and third map-reduce jobs accordingly. So, the second and third map-reduce jobs wait until the first is completed.

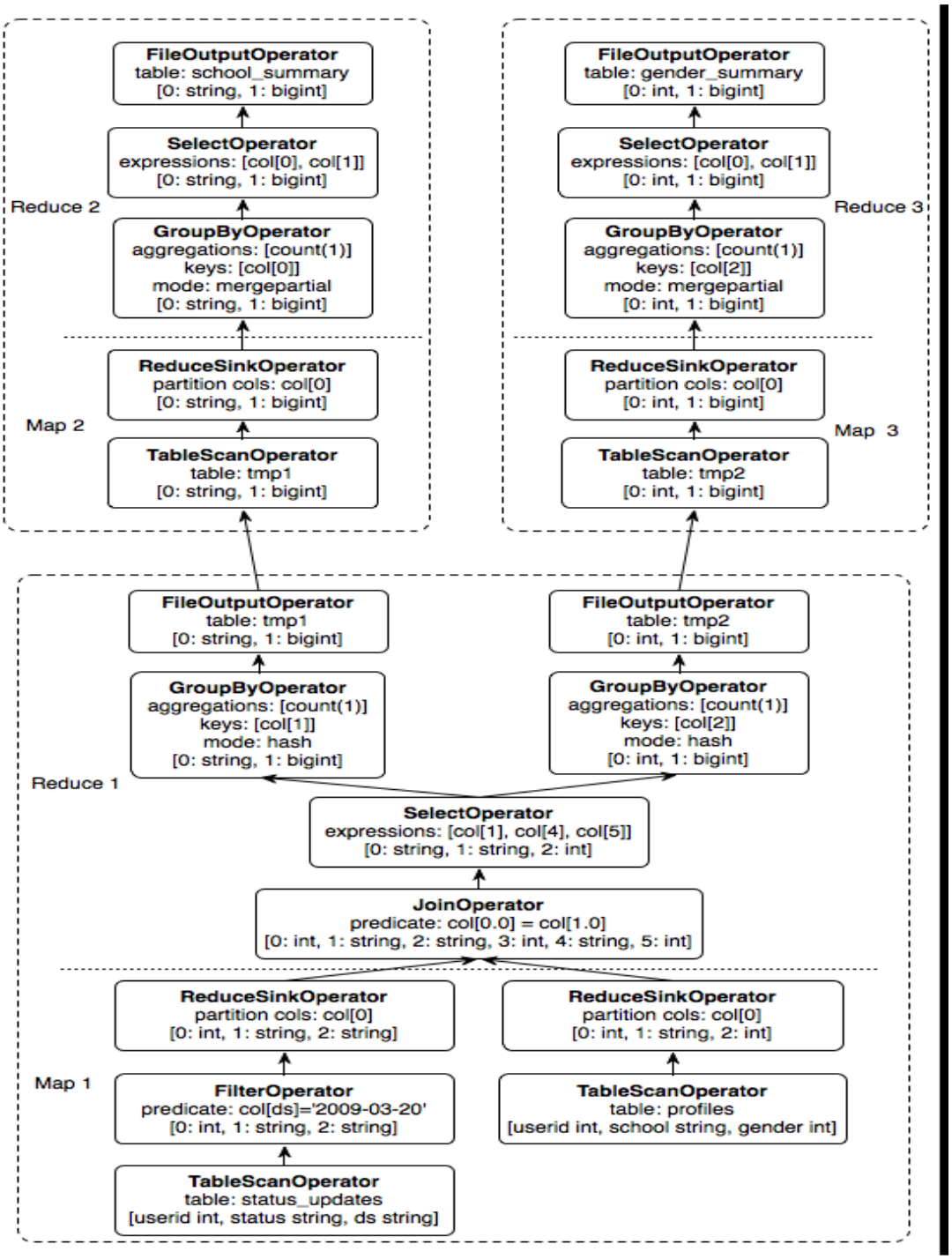


Figure 8.1: Query plan for multiple table data insert query with three map-reduce jobs

8.7 Future work

Hive is the first step in building an open-source database on a web-scale map-reduce data processing system (Hadoop). The distinct characteristics of the specific way of data storage, as well as the execution engine, have forced computer scientists to revisit the techniques of query processing. It was considered crucial to either modify or rewrite several query algorithms so that they perform efficiently in the new shape of conditions.

Hive is part of an Apache project, with an active user and developer community both within and outside Facebook. The Hive database instance on Facebook contains more than 700 terabytes of data and supports 5,000 queries on a daily basis. There are many important avenues that need to be made towards the optimization on the HIVE platform

- HiveQL currently accepts only a subset of SQL as valid queries. The Hive team is working towards integrating SQL syntax as a whole.
- At the moment, Hive's optimization library is based on a small number of simple rules. The plan is to build a cost-based optimizer and adaptive optimization techniques based on conditions that aim at more effective plans.
- There is optimization on columnar storage and more intelligent data placement with the aim to improve scan performance.
- In the preliminary experiments conducted by the Hive team, the possibility for improvisation on the performance of Hadoop by 20% compared to current calculations. Optimization involves faster Hadoop data structures, for example the use of Text instead of String.
- Optimization and enhancement of JDBC and ODBC drivers in Hive for integration with traditional BI tools which currently only work with traditional database systems.
- Optimization of methods and techniques for multi-query cases and performing generic n-way joins processing in a single map-reduce job.

CHAPTER 9 Pig

Pig is a Hadoop extension that simplifies Hadoop code development by providing a high-level data processing language while retaining Hadoop's advantages regarding scalability and reliability and can describe jobs executions and data flows. It also enables the execution of several jobs from one script. Yahoo, one of the most significant users of Hadoop and by extension of Pig, runs 40% of all its Hadoop jobs with Pig. Twitter is also another well-known user of Pig.

Pig consists of two major components:

• A high-level data processing language called Pig Latin

• A compiler that compiles and runs the Pig Latin code in a choice of evaluation mechanisms. The main evaluation mechanism is Hadoop.

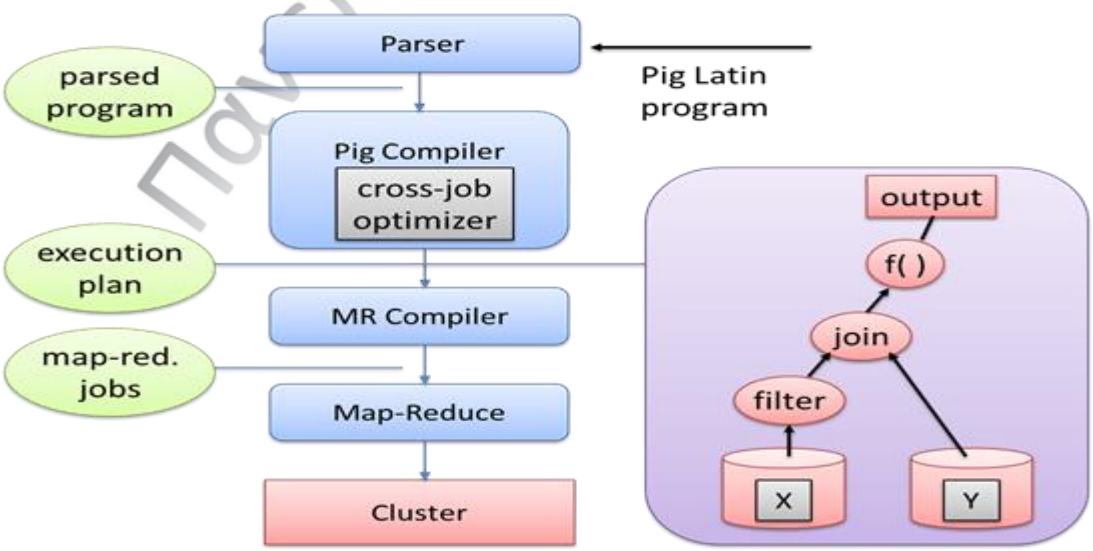


Figure 9.1.1: Ping architecture

Pig simplifies programming because of the ease of expressing methods and variables in Pig Latin. The compiler helps to automatically exploit optimized opportunities in a piece of code. This frees the user from having to tune your program manually, since when the Pig compiler improves, the code is automatically executed faster.

Code writing in Pig follows a specific sequence of steps where each step is a single high-level data transformation task. The transformation supports relational-style operators, such as filter, union, group, and joins. An simple example of the Pig Latin program that processes a search query in a log file looks like this:

```
Log = LOAD ‘excite-small.log’ AS (user, time, query);

Grpd = GROUP log BY user;

Cntd = FOREACH grpd GENERATE group COUNT(log);

DUMP cnt;
```

9.2 Pig Data Types

Input data can be have any format. Known data formats, such as tab-delimited text, are fully supported. Furthermore, users can add functions to support other data formats. Pig does not require metadata or a data representation schema, but it can make use of these advantages if they are provided. Pig can operate on relational, nested, semistructured, or unstructured data. Since it supports diversity of data, Pig also supports complex data types, such as bugs and tuples, which can form sophisticated data structures. A field with a tuple format or a value in a map can be null, a simple or a complex type of data, offering the potential for the formation of complex data structures.

Appendage I

Installment Hadoop Framework, Hive Warehouse System and GIS Tools for Hadoop:

The job takes place in single machine with Linux (Debian) operating system.

Hadoop Framework:

- root@cslab2:/# addgroup hadoop
- root@cslab2:/#adduser --ingroup hadoop hadoop
- Created an ssh RSA authentication key (be able to log into a remote site from my account, without having to type my password)
  root@cslab2:/# ssh-keygen –t rsa –P “”
  root@cslab2:/# cat /home/hadoop/.ssh/id_rsa.pub >>
  /home/hadoop/.shh/authorized_keys
- root@cslab2:/# cd /usr/local
  root@cslab2:/# wget
  http://apache.com/communilink.net/hadoop/core/hadoop/hadoop-1.0.4/hadoop-1.0.4.tar.gz
  root@cslab2:/#tar -xvf hadoop-1.0.4.tar.gz
- root@cslab2:/usr/local/# chown –R hadoop: hadoop hadoop-1.0.4.
  root@cslab2:/usr/local/# ln –s hadoop-1.0.4/hadoop (create shortcut)
- /etc/hosts :
  127.0.0.1 localhost
  195.251.230.18 localhost
  # The following lines are desirable for IPv6 capable hosts
  ::1 localhost ip6-localhost ip6-loopback
  ff02::1 ip6-allnodes
  ff02::2 ip6-allrouters
- Hadoop Configuration:
  Hadoop-site.xml:

```
<!-- mandatory -->
<configuration>
  <property>
    <name>hadoopdb.config.file</name>
    <value>HadoopDB.xml</value>
    <description>The name of the HadoopDB cluster configuration
file</description>
</property>
<property>
    <name>hadoopdb.fetch.size</name>
   <value>1000</value>
   <description>The number of records fetched from JDBC ResultSet at
     once</description>
  </property>
 <!-- optional, default false -->
<property>
     <name>hadoopdb.config.replication</name>
      <value>false</value>
      <description>Tells HadoopDB Catalog whether replication is enabled.
       Replica locations need to be specified in the catalog.
```

```
        False causes replica information to be ignored.</description>
</property>
<property>
      <name>hadoop.tmp.dir</name>
       <value>/tmp/hadoop-${user.name}</value>
</property>
<property>
      <name>fs.default.name</name>
      <value>hdfs://localhost:54310</value>
 </property>
<property>
     <name>mapred.job.tracker</name>
     <value>hdfs://localhost:54311</value>
  </property>
  <property>
      <name>dfs.replication</name>
       <value>8</value>
    </property>
    <property>
     <name>mapred.child.java.opts</name>
     <value>-Xmx512m</value>
    </property>
  </configuration>
```

Core-site.xml:

```
  <configuration>
    <property>
          <name>hadoop.tmp.dir</name>
          <value>/home/hadoop/single/hadoop-${user.name}</value>
    </property>
    <property>
         <name>fs.default.name</name>
         <value>hdfs://localhost:9000</value>
          <final>true</final>
    </property>
    <property>
          <name>hadoopdb.config.file</name>
          <value>HadoopDB.xml</value>
          <description>The Name of the HadoopDB cluster configuration
          File</description>
       </property>
     </configuration>
```

Hdfs-site.xml:

```
   <configuration>
          <property>
                  <name>dfs.replication</name>
                  <value>3</value>
          </property>


     </configuration>
```

Map-site.xml:

```
        <configuration>

           <property>
```

```
<name>mapred.job.tracker</name>
<value>localhost:9001</value>
<final>true</final>
</property>
<property>
<name>mapred.job.tracker.http.address</name>
<value>0.0.0.0:9002</value>
</property>
</configuration>
```

- NameNode Format:

```
hadoop@cslab2:/usr/local/hadoop# sudo bin/hadoop namenode –format
hadoop@cslab2:/usr/local/hadoop$ bin/start-all.sh
Warning: $HADOOP_HOME is deprecated.
starting namenode, logging to /usr/local/hadoop-1.0.4/libexec/../logs/hadoop-hadoop-namenode-cslab2.out
localhost: starting datanode, logging to /usr/local/hadoop-1.0.4/libexec/../logs/hadoop-hadoop-datanode-cslab2.out
localhost: starting secondarynamenode, logging to /usr/local/hadoop-1.0.4/libexec/../logs/hadoop-hadoop-secondarynamenode-cslab2.out
starting jobtracker, logging to /usr/local/hadoop-1.0.4/libexec/../logs/hadoop-hadoop-jobtracker-cslab2.out
localhost: starting tasktracker, logging to /usr/local/hadoop-1.0.4/libexec/../logs/hadoop-hadoop-tasktracker-cslab2.out

hadoop@cslab2:/usr/local/hadoop$ bin/hadoop dfsadmin -report
Warning: $HADOOP_HOME is deprecated.

13/10/27 16:58:05 WARN conf.Configuration: DEPRECATED: hadoop-site.xml found in the classpath. Usage of hadoop-site.xml is deprecated. Instead use core-site.xml, mapred-site.xml and hdfs-site.xml to override properties of core-default.xml, mapred-default.xml and hdfs-default.xml respectively
Configured Capacity: 488700362752 (455.14 GB)
Present Capacity: 454157680640 (422.97 GB)
DFS Remaining: 454157631488 (422.97 GB)
DFS Used: 49152 (48 KB)
DFS Used%: 0%
Under replicated blocks: 2
Blocks with corrupt replicas: 0
Missing blocks: 0
-------------------------------------------------
Datanodes available: 1 (1 total, 0 dead)
Name: 127.0.0.1:50010
Decommission Status : Normal
Configured Capacity: 488700362752 (455.14 GB)
DFS Used: 49152 (48 KB)
Non DFS Used: 34542682112 (32.17 GB)
DFS Remaining: 454157631488(422.97 GB)
DFS Used%: 0%
DFS Remaining%: 92.93%
Last contact: Sun Oct 27 16:58:05 EET 2013
```

Besides Hadoop comes with the several web interfaces:

By default, it's available at http://localhost:50070/. :

NameNode 'localhost:9000'

Started: Sun Oct 27 16:54:33 EET 2013

Version: 1.0.4, r1393290

Compiled: Wed Oct 3 05:13:58 UTC 2012 by hortonfo

Upgrades: There are no upgrades in progress.

Browse the filesystem

Namenode Logs

Cluster Summary

33 files and directories, 5 blocks = 38 total. Heap Size is 58.25 MB / 888.94 MB (6%)

Configured Capacity : 455.14 GB

DFS Used : 7.56 MB

Non DFS Used : 32.39 GB

DFS Remaining : 422.74 GB

DFS Used% : 0 %

DFS Remaining% : 92.88 %

Live Nodes : 1

Dead Nodes : 0

Decommissioning Nodes : 0

Number of Under-Replicated Blocks : 5

NameNode Storage:

Storage Directory Type State

tmp/hadoop-hadoop/dfs/name IMAGE_AND_EDITS Active

This is Apache Hadoop release 1.0.4

Hive Warehouse System:

- root@cslab2:/# tar –xzvf hive-0.10.0.tar.gz

  root@cslab2:/# cd hive

  root@cslab2:/# ant clean package

Compile hive on Hadoop:

root@cslab2:/# ant –Dhadoop.version=2.0.1 very-clean package tar test –Dinclude –postgress=true –Doverwrite=true –Dtest.silent=true

Create Data Directory:

root@cslab2:/# $HADOOP_HOME/bin/hadoop fs –mkdir /tmp

root@cslab2:/# $HADOOP_HOME/bin/hadoop fs -mkdir /user/hive/warehouse

root@cslab2:/# $HADOOP_HOME/bin/hadoop fs –chmod g+w /tmp

root@cslab2:/# $HADOOP_HOME/bin/hadoop fs –chmod g+w user/hive/warehouse

GIS Tools For Hadoop:

- Intro in Hive Environment:
  root@cslab2:/# $HIVE_HOME/bin/hive

  hive> add jar

  > /hive/lib/esri-geometry-api.jar

  > /hive/lib/spatial-sdk-hadoop.jar ;

  Added /hive/lib/esri-geometry-api.jar to class path

  Added resource: /hive/lib/esri-geometry-api.jar

  Added /hive/lib/spatial-sdk-hadoop.jar to class path

  Added resource: /hive/lib/spatial-sdk-hadoop.jar

  hive> create temporary function ST_Point as

  'com.esri.hadoop.hive.ST_Point';

  OK

  Time taken: 0.184 seconds

  hive> create temporary function ST_Contains as

  'com.esri.hadoop.hive.ST_Contains';

  OK

  Time taken: 0.004 seconds

- Create Tables:
  hive> CREATE EXTERNAL TABLE IF NOT EXISTS earthquakes (earthquake_date STRING, latitude DOUBLE, longitude DOUBLE, magnitude DOUBLE)
  ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
  LOCATION '${env:HOME}/esri-git/gis-tools-for-hadoop/samples/data/earthquake-data';
  hive> CREATE EXTERNAL TABLE IF NOT EXISTS counties (Area string, Perimeter string,
  State string, County string, Name string, BoundaryShape binary)
  ROW FORMAT SERDE 'com.esri.hadoop.hive.serde.JsonSerde'
  STORED AS INPUTFORMAT
  'com.esri.json.hadoop.EnclosedJsonInputFormat'
  OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'
  LOCATION '${env:HOME}/esri-git/gis-tools-for-hadoop/samples/data/counties-data';
- Load Data Into Tables:

```
hive> load data local inpath '/hive/earthquakes.csv' into table earthquakes;
    Copying data from file:/hive/earthquakes.csv
    Copying file: file:/hive/earthquakes.csv
    Loading data to table default.earthquakes
    Table default.earthquakes stats: [num_partitions: 0, num_files: 0,
    num_rows: 0, total_size: 0, raw_data_size: 0]
    OK
    Time taken: 0.399 seconds
hive> load data local inpath '/hive/california-counties.json' into table
    counties;
    Copying data from file:/hive/california-counties.json
    Copying file: file:/hive/california-counties.json
    Loading data to table default.counties
    Table default.counties stats: [num_partitions: 0, num_files: 0,
    num_rows: 0, total_size: 0, raw_data_size: 0]
    OK
    Time taken: 0.355 seconds
```

- Run a HiveQL statement:

```
hive> SELECT counties.name, count(*) cnt FROM counties
    > JOIN earthquakes
    > WHERE ST_Contains(counties.boundaryshape,
      ST_Point(earthquakes.longitude, earthquakes.latitude))
    > GROUP BY counties.name
    > ORDER BY cnt desc;
Total MapReduce jobs = 3
Launching Job 1 out of 3
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapred.reduce.tasks=<number>
Starting Job = job_201310271654_0004, Tracking URL =
http://localhost:9002/jobdetails.jsp?jobid=job_201310271654_0004
```

Kill Command = /usr/local/hadoop-1.0.4/libexec/../bin/hadoop job -kill job_201310271654_0004

Hadoop job information for Stage-1: number of mappers: 2; number of reducers: 1

2013-10-28 01:18:07,552 Stage-1 map = 0%, reduce = 0%

2013-10-28 01:18:13,573 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 4.16 sec

2013-10-28 01:18:14,578 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 4.16 sec

2013-10-28 01:18:15,582 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 4.16 sec

2013-10-28 01:18:16,586 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 4.16 sec

2013-10-28 01:18:17,595 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 4.16 sec

2013-10-28 01:18:18,598 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 4.16 sec

2013-10-28 01:18:19,601 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 4.16 sec

2013-10-28 01:18:20,605 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 4.16 sec

2013-10-28 01:18:21,608 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 4.16 sec

2013-10-28 01:18:22,612 Stage-1 map = 100%, reduce = 17%, Cumulative CPU 4.16 sec

2013-10-28 01:18:23,616 Stage-1 map = 100%, reduce = 17%, Cumulative CPU 4.16 sec

2013-10-28 01:18:24,619 Stage-1 map = 100%, reduce = 17%, Cumulative CPU 4.16 sec

2013-10-28 01:18:25,622 Stage-1 map = 100%, reduce = 72%, Cumulative CPU 4.16 sec

2013-10-28 01:18:26,625 Stage-1 map = 100%, reduce = 72%, Cumulative CPU 4.16 sec

2013-10-28 01:18:27,629 Stage-1 map = 100%, reduce = 72%, Cumulative CPU 4.16 sec

2013-10-28 01:18:28,633 Stage-1 map = 100%, reduce = 73%, Cumulative CPU 4.16 sec

2013-10-28 01:18:29,637 Stage-1 map = 100%, reduce = 73%, Cumulative CPU 4.16 sec

2013-10-28 01:18:30,641 Stage-1 map = 100%, reduce = 73%, Cumulative CPU 4.16 sec

2013-10-28 01:18:31,646 Stage-1 map = 100%, reduce = 74%, Cumulative CPU 4.16 sec

2013-10-28 01:18:32,650 Stage-1 map = 100%, reduce = 74%, Cumulative CPU 4.16 sec

2013-10-28 01:18:33,653 Stage-1 map = 100%, reduce = 74%, Cumulative CPU 4.16 sec

2013-10-28 01:18:34,656 Stage-1 map = 100%, reduce = 76%, Cumulative CPU 4.16 sec

2013-10-28 01:18:35,660 Stage-1 map = 100%, reduce = 76%, Cumulative CPU 4.16 sec

2013-10-28 01:18:36,663 Stage-1 map = 100%, reduce = 76%, Cumulative CPU 4.16 sec

2013-10-28 01:18:37,666 Stage-1 map = 100%, reduce = 77%, Cumulative CPU 4.16 sec

2013-10-28 01:18:38,669 Stage-1 map = 100%, reduce = 77%, Cumulative CPU 4.16 sec

2013-10-28 01:18:39,674 Stage-1 map = 100%, reduce = 77%, Cumulative CPU 4.16 sec

2013-10-28 01:18:40,678 Stage-1 map = 100%, reduce = 78%, Cumulative CPU 24.49 sec

2013-10-28 01:18:41,681 Stage-1 map = 100%, reduce = 78%, Cumulative CPU 24.49 sec

2013-10-28 01:18:42,686 Stage-1 map = 100%, reduce = 78%, Cumulative CPU 24.49 sec

2013-10-28 01:18:43,690 Stage-1 map = 100%, reduce = 79%, Cumulative CPU 24.49 sec

2013-10-28 01:18:44,693 Stage-1 map = 100%, reduce = 79%, Cumulative CPU 24.49 sec
2013-10-28 01:18:45,697 Stage-1 map = 100%, reduce = 79%, Cumulative CPU 24.49 sec
2013-10-28 01:18:46,701 Stage-1 map = 100%, reduce = 80%, Cumulative CPU 24.49 sec
2013-10-28 01:18:47,705 Stage-1 map = 100%, reduce = 80%, Cumulative CPU 24.49 sec
2013-10-28 01:18:48,709 Stage-1 map = 100%, reduce = 80%, Cumulative CPU 24.49 sec
2013-10-28 01:18:49,712 Stage-1 map = 100%, reduce = 81%, Cumulative CPU 24.49 sec
2013-10-28 01:18:50,716 Stage-1 map = 100%, reduce = 81%, Cumulative CPU 24.49 sec
2013-10-28 01:18:51,719 Stage-1 map = 100%, reduce = 81%, Cumulative CPU 24.49 sec
2013-10-28 01:18:52,728 Stage-1 map = 100%, reduce = 82%, Cumulative CPU 24.49 sec
2013-10-28 01:18:53,732 Stage-1 map = 100%, reduce = 82%, Cumulative CPU 24.49 sec
2013-10-28 01:18:54,735 Stage-1 map = 100%, reduce = 82%, Cumulative CPU 24.49 sec
2013-10-28 01:18:55,738 Stage-1 map = 100%, reduce = 83%, Cumulative CPU 24.49 sec
2013-10-28 01:18:56,741 Stage-1 map = 100%, reduce = 83%, Cumulative CPU 24.49 sec
2013-10-28 01:18:57,746 Stage-1 map = 100%, reduce = 83%, Cumulative CPU 24.49 sec
2013-10-28 01:18:58,749 Stage-1 map = 100%, reduce = 84%, Cumulative CPU 24.49 sec
2013-10-28 01:18:59,752 Stage-1 map = 100%, reduce = 84%, Cumulative CPU 24.49 sec
2013-10-28 01:19:00,756 Stage-1 map = 100%, reduce = 84%, Cumulative CPU 24.49 sec
2013-10-28 01:19:01,761 Stage-1 map = 100%, reduce = 85%, Cumulative CPU 24.49 sec
2013-10-28 01:19:02,769 Stage-1 map = 100%, reduce = 85%, Cumulative CPU 24.49 sec
2013-10-28 01:19:03,773 Stage-1 map = 100%, reduce = 85%, Cumulative CPU 24.49 sec
2013-10-28 01:19:04,776 Stage-1 map = 100%, reduce = 86%, Cumulative CPU 24.49 sec
2013-10-28 01:19:05,780 Stage-1 map = 100%, reduce = 86%, Cumulative CPU 24.49 sec
2013-10-28 01:19:06,784 Stage-1 map = 100%, reduce = 86%, Cumulative CPU 24.49 sec
2013-10-28 01:19:07,789 Stage-1 map = 100%, reduce = 88%, Cumulative CPU 24.49 sec
2013-10-28 01:19:08,792 Stage-1 map = 100%, reduce = 88%, Cumulative CPU 24.49 sec
2013-10-28 01:19:09,796 Stage-1 map = 100%, reduce = 88%, Cumulative CPU 24.49 sec
2013-10-28 01:19:10,799 Stage-1 map = 100%, reduce = 89%, Cumulative CPU 24.49 sec
2013-10-28 01:19:11,802 Stage-1 map = 100%, reduce = 89%, Cumulative CPU 24.49 sec
2013-10-28 01:19:12,808 Stage-1 map = 100%, reduce = 89%, Cumulative CPU 24.49 sec
2013-10-28 01:19:13,811 Stage-1 map = 100%, reduce = 89%, Cumulative CPU 24.49 sec
2013-10-28 01:19:14,815 Stage-1 map = 100%, reduce = 89%, Cumulative CPU 24.49 sec
2013-10-28 01:19:15,818 Stage-1 map = 100%, reduce = 89%, Cumulative CPU 24.49 sec
2013-10-28 01:19:16,821 Stage-1 map = 100%, reduce = 90%, Cumulative CPU 24.49 sec
2013-10-28 01:19:17,826 Stage-1 map = 100%, reduce = 90%, Cumulative CPU 24.49 sec

2013-10-28 01:19:18,829 Stage-1 map = 100%, reduce = 90%, Cumulative CPU 24.49 sec
2013-10-28 01:19:19,832 Stage-1 map = 100%, reduce = 92%, Cumulative CPU 24.49 sec
2013-10-28 01:19:20,836 Stage-1 map = 100%, reduce = 92%, Cumulative CPU 24.49 sec
2013-10-28 01:19:21,839 Stage-1 map = 100%, reduce = 92%, Cumulative CPU 24.49 sec
2013-10-28 01:19:22,844 Stage-1 map = 100%, reduce = 93%, Cumulative CPU 24.49 sec
2013-10-28 01:19:23,847 Stage-1 map = 100%, reduce = 93%, Cumulative CPU 24.49 sec
2013-10-28 01:19:24,850 Stage-1 map = 100%, reduce = 93%, Cumulative CPU 24.49 sec
2013-10-28 01:19:25,854 Stage-1 map = 100%, reduce = 94%, Cumulative CPU 24.49 sec
2013-10-28 01:19:26,857 Stage-1 map = 100%, reduce = 94%, Cumulative CPU 24.49 sec
2013-10-28 01:19:27,861 Stage-1 map = 100%, reduce = 94%, Cumulative CPU 24.49 sec
2013-10-28 01:19:28,865 Stage-1 map = 100%, reduce = 95%, Cumulative CPU 24.49 sec
2013-10-28 01:19:29,870 Stage-1 map = 100%, reduce = 95%, Cumulative CPU 24.49 sec
2013-10-28 01:19:30,873 Stage-1 map = 100%, reduce = 95%, Cumulative CPU 24.49 sec
2013-10-28 01:19:31,876 Stage-1 map = 100%, reduce = 96%, Cumulative CPU 24.49 sec
2013-10-28 01:19:32,881 Stage-1 map = 100%, reduce = 96%, Cumulative CPU 24.49 sec
2013-10-28 01:19:33,884 Stage-1 map = 100%, reduce = 96%, Cumulative CPU 24.49 sec
2013-10-28 01:19:34,888 Stage-1 map = 100%, reduce = 97%, Cumulative CPU 24.49 sec
2013-10-28 01:19:35,892 Stage-1 map = 100%, reduce = 97%, Cumulative CPU 24.49 sec
2013-10-28 01:19:36,895 Stage-1 map = 100%, reduce = 97%, Cumulative CPU 24.49 sec
2013-10-28 01:19:37,898 Stage-1 map = 100%, reduce = 98%, Cumulative CPU 24.49 sec
2013-10-28 01:19:38,902 Stage-1 map = 100%, reduce = 98%, Cumulative CPU 24.49 sec
2013-10-28 01:19:39,905 Stage-1 map = 100%, reduce = 98%, Cumulative CPU 24.49 sec
2013-10-28 01:19:40,910 Stage-1 map = 100%, reduce = 99%, Cumulative CPU 86.15 sec
2013-10-28 01:19:41,915 Stage-1 map = 100%, reduce = 99%, Cumulative CPU 86.15 sec
2013-10-28 01:19:42,918 Stage-1 map = 100%, reduce = 99%, Cumulative CPU 86.15 sec
2013-10-28 01:19:43,921 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 86.15 sec
2013-10-28 01:19:44,925 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 86.15 sec
2013-10-28 01:19:45,929 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 86.15 sec
2013-10-28 01:19:46,934 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 89.53 sec
2013-10-28 01:19:47,938 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 89.53 sec
2013-10-28 01:19:48,941 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 89.53 sec
2013-10-28 01:19:49,945 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 89.53 sec
2013-10-28 01:19:50,949 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 89.53 sec
2013-10-28 01:19:51,952 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 89.53 sec

2013-10-28 01:19:52,956 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 89.53 sec

MapReduce Total cumulative CPU time: 1 minutes 29 seconds 530 msec

Ended Job = job_201310271654_0004

Launching Job 2 out of 3

Number of reduce tasks not specified. Estimated from input data size: 1

In order to change the average load for a reducer (in bytes):

set hive.exec.reducers.bytes.per.reducer=<number>

In order to limit the maximum number of reducers:

set hive.exec.reducers.max=<number>

In order to set a constant number of reducers:

set mapred.reduce.tasks=<number>

Starting Job = job_201310271654_0005, Tracking URL = http://localhost:9002/jobdetails.jsp?jobid=job_201310271654_0005

Kill Command = /usr/local/hadoop-1.0.4/libexec/../bin/hadoop job -kill job_201310271654_0005

Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1

2013-10-28 01:20:01,519 Stage-2 map = 0%, reduce = 0%

2013-10-28 01:20:07,535 Stage-2 map = 100%, reduce = 0%, Cumulative CPU 0.45 sec

2013-10-28 01:20:08,540 Stage-2 map = 100%, reduce = 0%, Cumulative CPU 0.45 sec

2013-10-28 01:20:09,543 Stage-2 map = 100%, reduce = 0%, Cumulative CPU 0.45 sec

2013-10-28 01:20:10,547 Stage-2 map = 100%, reduce = 0%, Cumulative CPU 0.45 sec

2013-10-28 01:20:11,551 Stage-2 map = 100%, reduce = 0%, Cumulative CPU 0.45 sec

2013-10-28 01:20:12,554 Stage-2 map = 100%, reduce = 0%, Cumulative CPU 0.45 sec

2013-10-28 01:20:13,558 Stage-2 map = 100%, reduce = 0%, Cumulative CPU 0.45 sec

2013-10-28 01:20:14,561 Stage-2 map = 100%, reduce = 0%, Cumulative CPU 0.45 sec

2013-10-28 01:20:15,564 Stage-2 map = 100%, reduce = 0%, Cumulative CPU 0.45 sec

2013-10-28 01:20:16,569 Stage-2 map = 100%, reduce = 0%, Cumulative CPU 0.45 sec

2013-10-28 01:20:17,572 Stage-2 map = 100%, reduce = 0%, Cumulative CPU 0.45 sec

2013-10-28 01:20:18,576 Stage-2 map = 100%, reduce = 0%, Cumulative CPU 0.45 sec

2013-10-28 01:20:19,579 Stage-2 map = 100%, reduce = 100%, Cumulative CPU 1.36 sec

2013-10-28 01:20:20,583 Stage-2 map = 100%, reduce = 100%, Cumulative CPU 1.36 sec

2013-10-28 01:20:21,588 Stage-2 map = 100%, reduce = 100%, Cumulative CPU 1.36 sec

2013-10-28 01:20:22,592 Stage-2 map = 100%, reduce = 100%, Cumulative CPU 1.36 sec

2013-10-28 01:20:23,595 Stage-2 map = 100%, reduce = 100%, Cumulative CPU 1.36 sec

2013-10-28 01:20:24,598 Stage-2 map = 100%, reduce = 100%, Cumulative CPU 1.36 sec

2013-10-28 01:20:25,602 Stage-2 map = 100%, reduce = 100%, Cumulative CPU 1.36 sec

MapReduce Total cumulative CPU time: 1 seconds 360 msec

Ended Job = job_201310271654_0005

Launching Job 3 out of 3

Number of reduce tasks determined at compile time: 1

In order to change the average load for a reducer (in bytes):

set hive.exec.reducers.bytes.per.reducer=<number>

In order to limit the maximum number of reducers:

set hive.exec.reducers.max=<number>

In order to set a constant number of reducers:

set mapred.reduce.tasks=<number>

Starting Job = job_201310271654_0006, Tracking URL = http://localhost:9002/jobdetails.jsp?jobid=job_201310271654_0006

Kill Command = /usr/local/hadoop-1.0.4/libexec/../bin/hadoop job -kill job_201310271654_0006

Hadoop job information for Stage-3: number of mappers: 1; number of reducers: 1

2013-10-28 01:20:35,198 Stage-3 map = 0%, reduce = 0%

2013-10-28 01:20:41,215 Stage-3 map = 100%, reduce = 0%, Cumulative CPU 0.43 sec

2013-10-28 01:20:42,219 Stage-3 map = 100%, reduce = 0%, Cumulative CPU 0.43 sec

2013-10-28 01:20:43,222 Stage-3 map = 100%, reduce = 0%, Cumulative CPU 0.43 sec

2013-10-28 01:20:44,225 Stage-3 map = 100%, reduce = 0%, Cumulative CPU 0.43 sec

2013-10-28 01:20:45,228 Stage-3 map = 100%, reduce = 0%, Cumulative CPU 0.43 sec

2013-10-28 01:20:46,231 Stage-3 map = 100%, reduce = 0%, Cumulative CPU 0.43 sec

2013-10-28 01:20:47,234 Stage-3 map = 100%, reduce = 0%, Cumulative CPU 0.43 sec

2013-10-28 01:20:48,237 Stage-3 map = 100%, reduce = 0%, Cumulative CPU 0.43 sec

2013-10-28 01:20:49,242 Stage-3 map = 100%, reduce = 0%, Cumulative CPU 0.43 sec

2013-10-28 01:20:50,245 Stage-3 map = 100%, reduce = 0%, Cumulative CPU 0.43 sec

2013-10-28 01:20:51,248 Stage-3 map = 100%, reduce = 0%, Cumulative CPU 0.43 sec

2013-10-28 01:20:52,251 Stage-3 map = 100%, reduce = 0%, Cumulative CPU 0.43 sec

2013-10-28 01:20:53,257 Stage-3 map = 100%, reduce = 100%, Cumulative CPU 1.31 sec

2013-10-28 01:20:54,263 Stage-3 map = 100%, reduce = 100%, Cumulative CPU 1.31 sec

2013-10-28 01:20:55,266 Stage-3 map = 100%, reduce = 100%, Cumulative CPU 1.31 sec

2013-10-28 01:20:56,269 Stage-3 map = 100%, reduce = 100%, Cumulative CPU 1.31 sec

2013-10-28 01:20:57,273 Stage-3 map = 100%, reduce = 100%, Cumulative CPU 1.31 sec

2013-10-28 01:20:58,276 Stage-3 map = 100%, reduce = 100%, Cumulative CPU 1.31 sec

MapReduce Total cumulative CPU time: 1 seconds 310 msec

Ended Job = job_201310271654_0006

MapReduce Jobs Launched:

Job 0: Map: 2 Reduce: 1 Cumulative CPU: 89.53 sec HDFS Read: 7800137 HDFS Write: 541 SUCCESS

Job 1: Map: 1 Reduce: 1 Cumulative CPU: 1.36 sec HDFS Read: 996 HDFS Write: 541 SUCCESS

Job 2: Map: 1 Reduce: 1 Cumulative CPU: 1.31 sec HDFS Read: 996 HDFS Write: 201 SUCCESS

Total MapReduce CPU Time Spent: 1 minutes 32 seconds 200 msec

OK

Kern 72

San Bernardino 70

Imperial 56

Inyo 40

Los Angeles 36

Monterey 28

Riverside 28

Santa Clara 24

Fresno 22

San Benito 22

San Diego 14

Santa Cruz 10

San Luis Obispo 6

Ventura 6

Orange 4

San Mateo 2

Time taken: 180.489 seconds

Sources:

[1] **Abhishek Sagar,** Large Spatial Data Computation on Shared-Nothing Spatial DBMS Cluster via MapReduce, 2010

[2] **Ahmed Eldaway**, Mohamed F. Mokbel (University of Minesota) A Demonstration of Spatial Hadoop: An Efficient MapReduce Framework for Spatial Data 2013

[3] **Alex Kalinin**, OLAP in MapReduce, 2008

[4] **Anand Rajaraman , Dan Weld** MapReduce Architecture , 2012

[5] **Atta F.**, Implementation and Analysis of join Algorithms to handle skew for the Hadoop Map/Reduce Framework, 2010

[6] **Benjamin Guinebertiere , Philippe Beraud , Remi Olivier**, Leveraging a Hadoop cluster from SQL Server Integration Services (SSIS) , 2013

[7] **Brad HedLund**, Understanding Hadoop Clusters and the Network, 2009

[8] **Chuck Lam**, Hadoop in Action, 2011

[9] **Christos Doulkeridis, Kjetil Norvag**, On Saying “Enough Already!” in MapReduce, 2012

[10]**Dan Weld’s** MapReduce: Simplified Data Processing On Large Clusters

[11] **Diamanos Chatziantonioy** ,ASSET Queries: A Declarative Alternative to MapReduce, 2011

[12] **Dhruba Borthakur**, Hadoop Architecture and its Usage at Facebook

[13]**Emory University Team**, Hadoop-GIS: A Spatial Data Warehousing System Over MapReduce 2013

[14] **Eugene Ciurama**, Why Should You care About MapReduce, 2010

[15] **Facebook Data Infrastructure Team**, Hive – A Warehousing Solution Over a Map-Reduce Framework, 2009

[16] **Gery Menegaz**, What is NoSQL, and why do you need it? , 2009

[17] **Git Enterpise**,GIS Tools For Hadoop

[18] **Guy Harrison**, 10 things you should know about NoSQL databases, 2011

[19] **Hung-chihYangm Ali Dasdan** , MapReduce-Merge: Simplified Relational Data Processing on Large Clusters, 2007

[20] **Inneke Ponnet**, HadoopDB

[21] **Jiong Xie, Shu Yin, Xiaojum Ruan, Zhiyang Ding, Yun Tian, James Majors, Adam Manzanares, and Xiao Qin.,** Improving MapReduce Performance through Data Placement in Heterogeneous Hadoop Clusters, 2012

[22] **Kim Rose**, How Facebook uses Hadoop and Hive, 2013

[23] **Mahsa MofidPoor**, Index-Based Join Operations In Hive 2013

[24] **Mark Grower**, How to write a Hive UDF, 2012

[25] **Matthew Rathbone,** Hadoop Hive UDF Tutorial Extending Hive Custom Functions

[26] **McCreadie , Macdonald, Ounis**, Comparing Distributed indexing: To MapReduce or Not? (2009)

[27] **McKinsey Global Institute**, BigData: The next frontier for innovation, competition, productivity, 2010

[28] **Michael G. Noll**, Running Hadoop on Ubuntu Linux , 2009

[29] **Microsoft Research Silicon Valley , Columbia University**, Optimizing Data Partitioning for Data Parallel Computing

[30] **Padraig O'Sulivan** , Up and Running with HadoopDB, 2008

[31] **Philip Russon** , Big Data Analytics , 2011

[32] **Ricky Ho**, Hadoop Map/Reduce Implementation, 2011

[33] **Rick F. van der Lans**. Using SQL-MapReduce for Advanced Analytical Queries,, 2011

[34] **Technical University of Berlin Germany**, MapReduce and PACT- Comparing Data Parallel Programming Models , 2010

[35] **Tom White**, Hadoop The Definitive Guide. 2009

[36] **University of Glasgow**, Comparing Distributed Indexing: To MapReduce or Not, 2009

[37] **Vikas JaDhav, Jagannath Aghav, Sunil Dorwani,** Join Algorithms Using MapReduce: A survey , 2011

[38] **Yahoo Developer Network**, Apache Hadoop Tutorial Network, 2012

[39] **Yale University**, HadoopDB in Action: Building Real World Applications, 2011

[40] **Yale University, Brown University**, HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for analytical Workloads , 2009

[41] **Yale University , University of Wisconsin-Madison** Efficient Processing of Data Warehousing Queries in a Split Executing Environments, , 2011

[42] **Yogesh Sachwani** ,Hive Indexing, 2012

[43] **Zheng Shao**, Hadoop at Facebook,2011

[44] Apache Hadoop Documentation, 2013

[45] Apache Hive Documentation ,2013

[46] WikiPedia