International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-4 Issue-6, November 2014

Study of Hadoop Features for Large Scale Data

Dipali Salunkhe, Devendra Bahirat, Neha V. Koushik, Deepali Javale

Abstract*—The data from hospitals around the area (city, state or country) is huge. Handling it through traditional RDBMS will be inefficient and cumbersome because the data from various hospitals won't be in the same format. Another reason is RDBMS doesn't offer an efficient way to handle unstructured data (i.e. Media files).Thirdly, as the data becomes voluminous the time for retrieval increases exponentially. Hadoop has many advantages if used to store all the medical data of the patient and also media files related to it (i.e. X-Ray reports, sonography reports and videos of operation). This paper gives overview of Hadoop and its components and also comparison between Hadoop and RDBMS.

***Keywords*— HDFS, Mapreduce, Hbase**

I. INTRODUCTION

With the rapid development in the field of IT and medical sector, the demand of medical data has increased. The medical data of patient is scattered across various hospitals. So, it becomes really difficult to access this medical history of patients in case of emergency. Also, such medical data has to be shared among doctors and used for analysis purpose. So there is a need of centralized system for storing the medical data of patients and also for their efficient retrieval. There are many health care systems which currently use RDBMS to store, manage and analyze medical data. However, RDBMSs-based framework is not suitable for the requirements of massive health care data storage, management and analysis [6].As number of patients' increases the amount of medical data also goes on increasing which cannot be handled by RDBMS effectively. As a result, storage and retrieval of such large scale data becomes difficult. The framework of Hadoop provides a solution for this problem. Hadoop provides better performance and reliability.

II. HADOOP

Hadoop is an Apache open source project that includes open source implementations of a distributed file system [2] and MapReduce that were inspired by Google's GFS and MapReduce [4] projects. It also includes projects like Apache HBase [3] which is inspired by Google's BigTable. Hadoop framework is designed for processing big data on large cluster built of commodity hardware providing reliability scalability and efficiency.

Manuscript published on 30 November 2014.
*****Correspondence Author(s)
Dipali Salunkhe, MIT College of Engineering, Department of Computer Engineering, Pune, India.
Devendra Bahirat, MIT College of Engineering, Department of Computer Engineering, Pune, India.
Neha V. Koushik, MIT College of Engineering, Department of Computer Engineering, Pune, India.
Deepali Javale, MIT College of Engineering, Department of Computer Engineering, Pune, India.

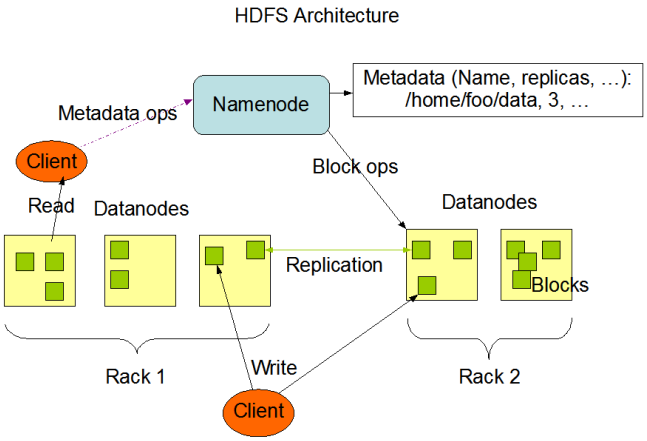
The components of Hadoop covered in this paper:

1. Hadoop's Distributed File System (HDFS).
2. The HBase storage system.
3. Hadoop's MapReduce.

III. HADOOP DISTRIBUTED FILE SYSTEM

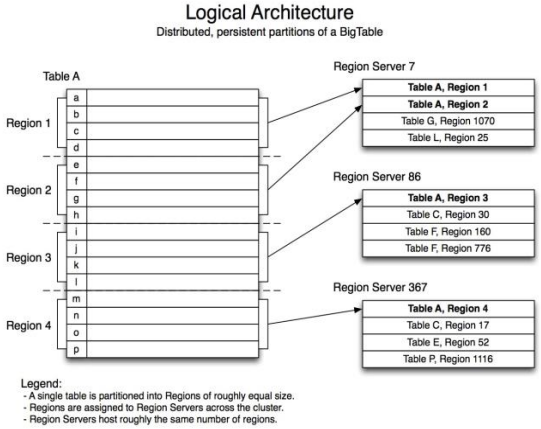
Hadoop provides a distributed file system (HDFS). HDFS is a master-slave system structure. In HDFS, file system metadata and application data are stored separately. It is designed for scalability and fault tolerance. It stores large files across machines in a large cluster. It stores each file by dividing it into blocks (usually 64 or 128 MB).All blocks in a file except the last block are the same size(typical block size is 64 MB).These blocks are then replicated on three or more servers. HDFS has two types of node. Its master node is called Namenode which contains metadata, and slave nodes are called Datanodes which contain application data. Namenode is visible to all cloud nodes and it also provides a uniform global view for file paths in a traditional hierarchical structure. It manages the entire namespace of the file system. The NameNode makes all decisions regarding replication of blocks. It decides the block location in order to meet the load balance and fault tolerance to ensure the high reliability of data. When a new block is needed, the NameNode allocates a block with a unique block ID and determines a list of DataNodes to host replicas of the block. It also ensure the mapping relationship between data blocks and data nodes. File contents are not stored hierarchically, but are divided into low level data chunks and stored in datanodes with replication. Data chunk pointers for files are linked to their corresponding locations by namenode. Namenode is responsible for open, close, rename adjustments while the datanode is responsible for the users to read and write data. HDFS provides APIs for MapReduce applications to read and write data in parallel. When an HDFS client wants to read a file, it will first contact the Namenode for the locations of data blocks comprising the file. It will then read the block contents from the closest Datanode from the Namenode. Similarly, the client contacts Namenode when writing to file. The Namenode allocates three datanodes to host the block replicas. The data is then written to datanodes. The block size and replication factor are configurable per file. An application can specify the number of replicas of a file. The replication factor can be specified at file creation time and can be changed later. Files in HDFS are write-once and have strictly one writer at any time.

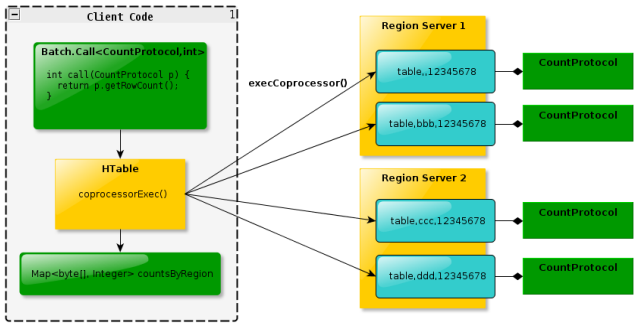
Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)



IV. HBASE

HBase is a distributed, column-oriented data store that runs on top of HDFS. HBase applications are written in Java like a typical MapReduce application. HBase is well suited for sparse data sets which are common in many big data use cases. HBase system consists of tables. Each table is made up of rows and columns. An HBase column represents an attribute of an object. It also allows many attributes to be grouped together called as family. The elements of a column family are stored together. A column family must be created before data can be stored under any column in that family. HBase provides flexibility as new columns can be added to family at any time. It also employs a master-slave topology, where its master maintains a table-like view for users. Tables are split for distributed storage into row-wise regions. These regions are stored on slave machines, using HBase's region servers. Both the master and region servers rely on HDFS to store data. Frequently executed queries and their results can be stored in to HBase, so that whenever the same request has to be fulfilled it will directly show the result to the user without parsing the actual document in HDFS. Hence, it will save lot of time for frequent business queries.





V. MAPREDUCE

MapReduce is a programming model and an implementation for processing and generating large data sets [4]. The main implementation of MapReduce is done Hadoop. The MapReduce model is a programming paradigm for designing parallel and distributed algorithms [8]. It comes with a simple interface that makes it easy for a programmer to design a parallel program that can efficiently perform data-intensive computations. We have to specify a Map function that generates a set of intermediate key/value pairs and also a Reduce function that merges all intermediate values associated with the same intermediate key. Many real world problems can be expressed with the help of the MapReduce model.

A. MapReduce Model

The MapReduce model specifies a computation as a sequence of map, shuffle, and reduce steps that operate on a given set of values ($X = \{x_1, x_2 \ldots x_n\}$)

- The map step executes a function, μ, to each value in given set of values, xi, which produces a finite set of key/value pairs (k, v) [8] .This key/value pair is called the intermediate key/value pair. To support parallel execution, the computation of the function μ(x) must depend only on x.
- The shuffle step collects all the intermediate key/value pairs produced in the map step to produce a set of lists, (for example, $L_k = \{k; v_1, v_2 \ldots\}$), where each such list contains all the values, v_j, such that $k_j = k$ for a particular key, k, assigned in the map step.
- The reduce step executes a function, ρ, for each list $L_k = \{k; v_1, v_2 \ldots\}$, formed in the shuffle step, to produce a set of values, $\{y_1, y_2 \ldots\}$ [8]. The reduce function, ρ, should be independent of other lists.

The reduce phase starts after all the Mappers have finished, and hence, this model is an example of Bulk Synchronous Processing (BSP). The shuffle phase is typically implemented by writing all the data that comes to a destination computer to disk. [9] The parallelism of the MapReduce framework comes from the fact that each map or reduce operation can be executed on a separate processor independently of others.

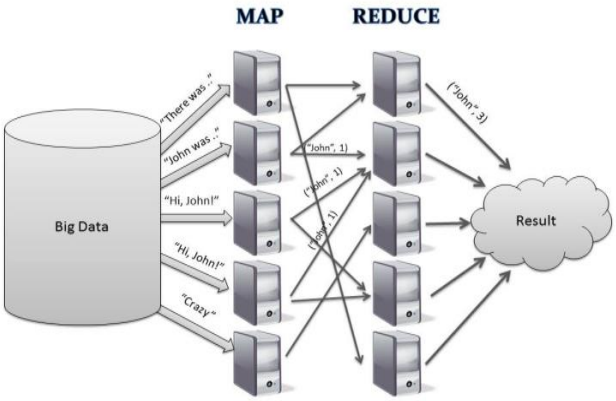
B. Execution Overview

The sequence of actions occurring when the user program calls the MapReduce function is as given below

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

1. The MapReduce library in the user program first splits the input files into N number of pieces, typically of sizes 16 MB to 64 MB per piece [4]. It then starts up many copies of the same program on all of the machines in the cluster of distributed file system.
2. The original copy of the program is the master. The rest of the copies are workers that are assigned some work (viz. map or reduce) by the master. There are M number of map tasks and R number of reduce tasks to assign. The master picks the idle workers and assigns each one a task.
3. A worker who is assigned a map task starts reading the contents of the corresponding input piece. It then parses all the key/value pairs from the input data and passes each pair to the user-defined Map function [4]. The intermediate key/value pairs are produced by this function and are buffered in memory of the worker.
4. After certain time interval, the buffered key/value pairs are written to local disk, partitioned into R regions. The locations of these buffered pairs on the local disk are passed back to the master [4]. The master then forwards these locations to the reduce workers.
5. On receiving the notification about these locations from the master, a reduce worker then reads the buffered data from the local disks of the map workers.
6. When a reduce worker finishes reading all intermediate data, it then sorts the data according to the intermediate keys so that all occurrences of the same key are clubbed together one after the other.
7. The reduce worker now starts iterating over the sorted intermediate data and for each unique intermediate key, it passes the key and the corresponding set of intermediate values to the user-defined Reduce function [4] .
8. The output of the Reduce function is then appended to a final output file.
9. When all map and reduce tasks have been completed, the master wakes up the user program indicating that the work is over. At this point, the MapReduce function call in the user program returns back to the user code which follows next to the MapReduce function call.[4]

After successful completion, the output of the MapReduce execution is available in the R output files (one per reduce task, with file names as specified by the user).The working of the MapReduce programming model is best shown in the following diagram:



C. Complexity Measures

The complexity of MapReduce model is measured with the help of Key Complexity and Sequential Complexity
Key Complexity consists of three parts:

- The maximum size of a key/value pair
- The maximum running time for a key/value pair.
- The maximum memory used to process a key/value pair.[9]

To get Sequential Complexity we sum over all Mappers and Reducers as opposed to looking at the worst.

- The size of all key/value pairs
- The total running time for all Mappers and Reducers for all key/value pairs.[9]

Notice that we do not consider the total memory for sequential complexity measure, because it depends on the number of Reducers operating at any given time and is a property of deployment model as opposed to the algorithm.

D. Fault Tolerance

The MapReduce programming model was basically built to process large data sets in parallel on various machines. Therefore it is highly fault tolerant. The basic fault tolerance is achieved with the following simple technique. The master pings every worker after a particular interval of time. If no response is received from a worker in a certain amount of time, the master marks the worker as failed.If the master has a failure point then it is advisable to regularly create checkpoints and restore them whenever the master fails. The detailed fault tolerance is given in [4].

VI. COMPARISON BETWEEN HADOOP AND RDBMS

Hadoop	RDBMS
It is Open Source.	It is mostly proprietary.
It is a Framework. An Eco System Suite of java based (mostly) projects.	It is one project with multiple components.
It is designed to support distributed architecture(HDFS)	It is designed with idea of server client Architecture.
It is designed to run on commodity hardware.	It expects high end server for high usage.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

It is Cost efficient.	It is Costly.
It has high fault tolerance	It has legacy procedure.
It is based on distributed file system like GFS, HDFS..	It Relies on OS file system.
Very good support of unstructured data.	Needs structured data
Flexible, evolvable and fast	Needs to follow defined constraints
It is still evolving	It has lots of very good products like oracle, sql.
It is suitable for Batch processing	It is suitable for Real time Read/Write
It supports sequential write.	It supports arbitrary insert and update

VII. CONCLUSION

By looking at the facilities given by Hadoop and its added advantages over RDBMS it can be said that Hadoop is much more efficient and cost effective when it comes to storing large data sets which have structured as well as unstructured data.

REFERENCES

1. Apache Hadoop Available at http://hadoop.apache.org
2. Apache HDFS Available at http://hadoop.apache.org/hdfs
3. Apache HBase. Available at http://hbase.apache.org
4. MapReduce Simplified Data Processing on Large Clusters Available at http://labs.google.com/papers/mapreduceosdi04.pdf
5. T. White, Hadoop: The Definitive Guide. O'Reilly Media, Yahoo! Press, June 5, 2009.
6. Hongyong Yu ; Deshuai Wang" Research and Implementation of Massive Health Care Data Management and Analysis Based on Hadoop " Computational and Information Sciences (ICCIS), 2012.
7. Wang F, Qiu J, Yang J, et al. Hadoop high availability through metadata replication [C].Proceeding of the First International Workshop on Cloud Data Management_Hong Kong_China_ November 2009.
8. Sorting, Searching, and Simulation in the MapReduce Framework by Michael T. Goodrich from University of California, Nodari Sitchinava from Aarhus University, Qin Zhang from Aarhus University
9. Complexity Measures for Map-Reduce, and Comparison to Parallel Computing* by Ashish Goel Stanford University and Twitter, Kamesh Munagala Duke University and Twitter in November 11, 2012

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)