ISSN: 2278 – 1323
International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)
Volume 5, Issue 11, November 2016

[1]

DATA MIGRATION METHODOLOGY FROM SQL TO COLUMN ORIENTED DATABASES (HBase)

Vishal Seshagiri, Manohar Lakshmana Vadaga, Jainam Jatin Shah, Ms. Priya Karunakaran

Abstract— **Big databases are the buzzword of today's tech community, many recognize it but only a few are able to understand and leverage it to practical use. Currently, the field of data migration is very topical. As the number of applications developed rapidly, the ever-increasing volume of data collected has driven the architectural migration from RDBMS to NoSQL DB. This very recent technology is important enough in the field of database management. The main aim of this project is to identify a NoSQL DB store which meets our requirements and chalk out a sound execution strategy to migrate data from the given Oracle 11g database. In this research paper we will be discussing the steps for migrating a Oracle 11g database and delve into the finer details about the choice of NoSQL Database chosen by us for implementing the migration as a part of our capstone engineering project.**

Index Terms—**Big Data, SQOOP, HBase, NoSQL Database.**

I. Introduction

Building on the problem statement of our project the following requirements build a clearer picture of our end user's requirements:

- Transactional capability such as real time reads
- Analytical capability like that of conventional data warehouses
- Scalability using commodity hardware
- Parallel and distributed computing capability
- NoSQL access methods such as python and java API.

These were the reasons which motivated us to consider NoSQL database variants for our datastore.
The basic difference between a traditional RDBMS and NoSQL Database is highlighted in the table below.

[1]*Manuscript received November, 2016*.
Vishal Seshagiri, *Computer Engineering, University of Mumbai, St. Francis Institute Of Technology, Mumbai, India*
Manohar Lakshmana Vadaga, *Computer Engineering, University of Mumbai, St. Francis Institute Of Technology, Mumbai, India*
Jainam Jatin Shah, *Computer Engineering, University of Mumbai, St. Francis Institute Of Technology, Mumbai, India*
Ms. Priya Karunakaran, *Assistant Professor Computer Engineering, University of Mumbai, St. Francis Institute Of Technology, Mumbai, India*

Nosql	RDBMS
No predefined schema/ less rigid schema	Predefined schema
Unstructured and constantly evolving data	Relational data where relationships are stored in separate tables
No Declarative Query Language	Structured Query Language
Weaker Transactional Guarantee	Strong Transactional Guarantee

In our quest to find a suitable database for our project we identified four main types of NoSQL DB variants:

- Key value store : Amazon S3 Dynamo
- Document based store : CouchDB, MongoDB
- Graph Based : Neo4j
- Column Based store : Apache Hbase

The following features are characteristic of a NoSQL database:

- A class of databases that can store unstructured data with ease and also handle evolving data types and schema.
- NoSQL databases are distributed in nature and can scale horizontally.
- Can handle large volumes of data (terabytes/petabytes).
- Since distributed in nature, they are designed to recover from failures of machine.
- More flexible data model.
- Uses clusters of cheap commodity hardware servers to manage the exploding data.

II. Why Column Oriented DATABASES?

The database we have chosen to implement our project is Hbase a column oriented database. Column oriented databases are increasingly being chosen by a large number of organizations to implement their big databases. This section gives a clear perspective as to why and under what scenarios should one opt for a column oriented databases. There are specific use cases where column oriented databases are considered to outperform traditional row oriented databases:

- The first case under consideration is that of data access. For instance, we wanted average marks by college, branch and year_of_exam, where year_of_exam is greater than 1998. For the purposes of estimating population counts, in our imaginary dataset 60% of the records have a year_of_exam post-1998. Consider the implementation for both the column- and row-oriented data structures to generate results.A straightforward method is to create aggregators (sum and count) for [college, branch, and year_of_exam]. One must then iterate over all the rows checking if an aggregator exists for each combination, creating one if it doesn't, and then computing the new sum and incrementing the count. This method is table scanning linearly through the entire 52GB of data, but because it's end to end it is as fast as it can be processed. Now doing the same with column oriented data. One iterates over the year_of_exam, and if the column has a value > 1998, accessing the marks and performing the aggregator operations based upon the data pulled from college and branch. In this example, one is dealing with data that is separated by GBs, which means address jumping that is more expensive both in memory and with the processor, thereby making Column Oriented Databases the preferred mode of storage.

- Data is stored and retrieved in columns and hence can read only relevant data if only some data is required
- Read and Write are typically slower operations
- Well suited for OLAP systems
- Can efficiently perform operations applicable to the entire dataset and hence enables aggregation over many rows and columns
- Permits high compression rates due to few distinct values in columns
- In data warehouses, for instance, data is populated into dimensions and facts, with the fact components stored in a specific order such that the values used in computations can often be found in sequential ranges. e.g. facts 1027 – 2087 are the daily sales of widgets from a specific store. Financial systems, such as a kdb+ HFT system, are generally column-oriented because each set of data is a specific, linear set of values, such as streaming bids for AAPL.

Features of column oriented datastores :

- Data is stored and retrieved in columns and hence can read only relevant data if only some data is required.
- Read and Write are typically slower operations.
- Well suited for OLAP systems.
- Can efficiently perform operations applicable to the entire dataset and hence enables aggregation over many rows and columns.
- Permits high compression rates due to few distinct values in columns.

III. HBase

HBase is a distributed column-oriented database built on top of the Hadoop file system. It is an open-source project and is horizontally scalable.
HBase is a data model that is similar to Google's big table designed to provide quick random access to huge amounts of structured data. It leverages the fault tolerance provided by the Hadoop Distributed File System (HDFS).
It is a part of the Hadoop ecosystem that provides random real-time read/write access to data in the Hadoop File System.
One can store the data in HDFS either directly or through HBase. Data consumer reads/accesses the data in HDFS randomly using HBase. HBase sits on top of the Hadoop File System and provides read and write access.
As mentioned Hbase stores its data on the HDFS, it is essential to highlight the differences between Hbase and Hadoop since the former is database while the latter is a datastore system.

HDFS –

- Is suited for High Latency operations batch processing
- Data is primarily accessed through MapReduce
- Is designed for batch processing and hence doesn't have a concept of random reads/writes

HBase –

- Is built for Low Latency operations.
- Provides access to single rows from billions of records.
- Data is accessed through shell commands, Client APIs in Java, REST, Avro or Thrift.

Features of HBase:

- Is Schema-less.
- Is a Column-oriented datastore.
- Is designed to store non-normalized Data.
- Supports Automatic Partitioning.

IV. HBase Architecture

The HBase Physical Architecture consists of servers in a Master-Slave relationship as shown below. Typically, the HBase cluster has one Master node, called HMaster and multiple Region Servers called HRegionServer. Each Region Server contains multiple Regions – HRegions.

Just like in a Relational Database, data in HBase is stored in Tables and these Tables are stored in Regions. When a Table becomes too big, the Table is partitioned into multiple Regions. These Regions are assigned to Region Servers across the cluster. Each Region Server hosts roughly the same number of Regions.

The HMaster in the HBase is responsible for

- Performing Administration
- Managing and Monitoring the Cluster
- Assigning Regions to the Region Servers
- Controlling the Load Balancing and Failover

On the other hand, the HRegionServer perform the following work

- Hosting and managing Regions
- Splitting the Regions automatically
- Handling the read/write requests
- Communicating with the Clients directly

Each Region Server contains a Write-Ahead Log (called HLog) and multiple Regions. Each Region in turn is made up of a MemStore and multiple StoreFiles (HFile). The data lives in these StoreFiles in the form of Column Families (explained below). The MemStore holds in-memory modifications to the Store (data).

The mapping of Regions to Region Server is kept in a system table called .META. When trying to read or write data from HBase, the clients read the required Region information from the .META table and directly communicate with the appropriate Region Server. Each Region is identified by the start key (inclusive) and the end key (exclusive)

Zookeeper is to HBase as YARN is to the HDFS. Zookeeper maintains the server state in a cluster. It uses consensus to maintain which servers are alive and available and provides notification in case of server failure. The Zookeeper maintains ephemeral nodes for active sessions via heartbeats.

V. HBase Model

The Data Model in HBase is designed to accommodate semi-structured data that could vary in field size, data type and columns. Additionally, the layout of the data model makes it easier to partition the data and distribute it across the cluster. The Data Model in HBase is made of different logical components such as Tables, Rows, Column Families, Columns, Cells and Versions.

- *Tables* – The HBase Tables are more like logical collection of rows stored in separate partitions called Regions. As shown above, every Region is then served by exactly one Region Server. The figure above shows a representation of a Table.
- *Rows* – A row is one instance of data in a table and is identified by a *rowkey*. Rowkeys are unique in a Table and are always treated as a byte[].
- *Column Families* – Data in a row are grouped together as Column Families. Each Column Family has one more Columns and these Columns in a family are stored together in a low level storage file known as HFile. Column Families form the basic unit of physical storage to which certain HBase features like compression are applied. Hence it's important that proper care be taken when designing Column Families in table. The table above shows Customer and Sales Column Families. The Customer Column Family is made up 2 columns – Name and City, whereas the Sales Column Families is made up to 2 columns – Product and Amount.
- *Columns* – A Column Family is made of one or more columns. A Column is identified by a Column Qualifier that consists of the Column Family name concatenated with the Column name using a colon – example: columnfamily:columnname. There can be multiple Columns within a Column Family and Rows within a table can have varied number of Columns.
- *Cell* – A Cell stores data and is essentially a unique combination of *rowkey*, Column Family and the Column (Column Qualifier). The data stored in a Cell is called its value and the data type is always treated as byte[].
- *Version* – The data stored in a cell is versioned and versions of data are identified by the timestamp. The number of versions of data retained in a column family is configurable and this value by default is 3.

VI. Sqoop

Apache Sqoop is a tool which is designed for efficient export or import of bulk data between Apache Hadoop and structured datastores. Presently Sqoop is a Top-Level Apache project which is a command line interface application written in java.

PURPOSE:
1) Sqoop is designed efficiently for the purpose of transferring huge amount of data between Apache hadoop and structured datastores such as relational.
2) It copies data quickly from external systems to hadoop.
3) It enables data imports from external data stores and enterprise data warehouses into hadoop.
4) It ensures fast performance by parallelizing data transfer and utilizes optimal system.
5) Sqoop supports analyses of data efficiently.
6) It even mitigates excessive loads to external systems.

Working:
1) Sqoop runs in hadoop cluster. It has access to hadoop core.Sqoop use mappers to slice the incoming data.
2) Sqoop will communicate with the database store for fetching information called metadata from relational datastore. This metadata is being used for initiating java class by the Sqoop
3) Sqoop gets the metadata from DB store.
4) Sqoop will internally create a JAVA class using JDBC API. Sqoop will compile the java class using JDK and compare jar files.
5) Sqoop tries again to communicate with database store,once the jar files are created in order to find the split column which will enable Sqoop to fetch data from the database.
6) Finally Sqoop will place the retrieved data into HDFS.

VII. Steps to Migrate from Oracle 11g to HBase

- Setup Hadoop on the system.
- Use SQOOP to migrate data (tables) from Oracle 11g to Hadoop Distributed File System.
- Convert the data stored in HDFS to a designated datastore format such as XML or CSV etc.
- Setup HBase on top of the Hadoop framework.
- Map the data onto tables created on the HBase – column oriented database based on the data access needs of the applications.

VIII. Steps to Install Hadoop on Ubuntu16:

- Download and unzip the Hadoop binary files from http://hadoop.apache.org/releases.html.
- Provide ownership to the downloaded folders.
- Set the hadoop path variables on the bashrc file and source it.

export JAVA_HOME=/usr/lib/jvm/java-8-oracle (path to your jdk installation)
export HADOOP_HOME=/home/hduserjune2016/hadoop
export PATH=$PATH:$HADOOP_HOME/bin

```
export PATH=$PATH:$HADOOP_HOME/sbin
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export
HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_H
OME/lib/native
export                          HADOOP_OPTS="-
Djava.library.path=$HADOOP_HOME/lib"
```

- Make the following changes to the configuration files inside the /etc/hadoop folder to run hadoop locally on standalone mode

1. hadoop-env.sh

```
JAVA_HOME=<<path to the jdk installation>>
```

2. core-site.xml

```
<configuration>
        <property>
                <name>fs.default.name</name>
                <value>hdfs://localhost:9005</value>
        </property>
        <property>
                <name>hadoop.tmp.dir</name>
                <value>/home/hduserjune2016/hadoop_t
        mp</value>
        </property>
</configuration>
```

3. hdfs-site.xml

```
<configuration>
        <property>
                <name>dfs.replication</name>
                <value>1</value>  //The value 1 here is
used because this is a pseudo distributed hdfs therefore no
replication there value 1
        </property>
        <property>
                <name>dfs.namenode.name.dir</name
                <value>file:/home/hduserjune2016/hadoo
        p_tmp/hdfs/namenode</value>
        </property>
        <property>
                <name>dfs.datanode.data.dir</name>
                <value>/home/hduserjune2016/hadoop_t
        mp/hdfs/datanode</value>
        </property>
</configuration>
```

4. yarn-site.xml

```
<configuration>
        <property>
                <name>yarn.nodemanager.aux-
                services</name>
                        <value>mapreduce_shuffle</valu
                e>
        </property>
        <property>
                <name>yarn.resourcemanager.hostename
        </name>
                <value>localhost</value>
        </property>
</configuration>
```

5. mapred_site.xml

```
<configuration>
        <property>
                <name>mapreduce.framework.name</na
        me>
                <value>yarn</value>
        </property>
</configuration>
```

- On the terminal type the following commands to start yarn, dfs and the history server

1. start-yarn.sh
2. start-dfs.sh
3. mr-jobhistory-daemon.sh start historyserver

- Open these links to view the hdfs file system.

http://localhost:8088/cluster
http://localhost:50070/dfshealth.html#tab-datanode

IX. Steps to Install HBase on Top of Hadoop

Install HBASE – Standalone mode

- ../hbase-installation-folder/conf/hbase-env.sh
- export JAVA_HOME=<<path to appropriate jdk>>
- ../hbase-installation-folder/conf/hbase-site.xml
- hbase.rootdir : <<path to the directory on the local file system>>
- hbase.zookeeper.property.dataDir : <<path to the directory on the local file system>>

Run Hbase Server

- ~/.bashrc(environment variables)
- HBASE_HOME=<<path to the hbase installation>>
- PATH=$PATH:$HBASE_HOME/bin
- start-hbase.sh(starts the hbase server)
- stop-hbase.sh(stops the hbase server)
- Open the HBase shell using the command ‘hbase shell'

X. Conclusion

Undertaking this project has given us the opportunity to apply the Hadoop skills that we picked up on a course during the summer. We have had the opportunity to interact with people from the industry and have been entrusted with a real world problem of migrating from an Oracle 11g database to HBase. The research performed to author this review paper has paved the way for us to examine and zero in on a methodology for the migration and expedite the execution process.

REFERENCES

[1] Yishan Li and Sathiamoorthy Manoharan, “A performance comparison of SQL and NoSQL databases”, Conference: Communications, Computers and Signal Processing (PACRIM), 2013 IEEE Pacific Rim Conference, DOI: 10.1109/PACRIM.2013.6625441.

[2] Steve Ataky,Tsham Mpinda, Luís Gustavo Maschietto, Patrick Andjasubu Bungama, “From Relational Database to Column-Oriented

NoSQL Database: Migration Process”, *International Journal of Engineering Research & Technology (IJERT) ISSN: 2278-0181 IJERTV4IS050021 Vol. 4 Issue 05, May-2015.*

[3] Mr S. S. Aravinth,Ms A. Haseenah Begam, Ms S. Shanmugapriyaa, Ms S. Sowmya, “An Efficient HADOOP Frameworks SQOOP and Ambari for Big Data Processing”, IJIRST –*International Journal for Innovative Research in Science & Technology| Volume 1 | Issue 10 | March 2015 ISSN (online): 2349-6010.*

Vishal Seshagiri currently pursuing final year of Bachelor ofComputerEngineering at St. Francis Institute Of Technology from University of Mumbai.

Manohar Lakshmana Vadaga currently pursuing final year of Bachelor ofComputerEngineering at St. Francis Institute Of Technology from University of Mumbai.

Jainam Jatin Shah currently pursuing final year of Bachelor ofEngineering at St. Francis Institute Of Technology from University of Mumbai.

Ms. Priya Karunakaran currently Assistant Professor of the Department of Computer Engineering at St. Francis Institute Of Technology affiliated to University Of Mumbai.