*Big Data and Hadoop Development*

Session 09: HBase

Assignment 1

**A C A D G I L D Page 1**

*Big Data and Hadoop Development*

**Table of Contents**

Table of Contents .......................................................................................................................................... 2

1. Introduction .......................................................................................................................................... 3

2. Objective ............................................................................................................................................... 3

3. Prerequisites ......................................................................................................................................... 3

4. Associated Data Files ............................................................................................................................ 3

5. Problem Statement ............................................................................................................................... 3

6. Expected output .................................................................................................................................... 3

7. Approximate Time to Complete Task ................................................................................................... 3

**A C A D G I L D Page 2**

*Big Data and Hadoop Development*

**1. Introduction**

In this assignment you need to implement the skills learnt in the session.

**2. Objective**

This assignment will help you to understand the concepts HBase

**3. Prerequisites**

N/A

**4. Associated Data Files**

N/A

**5. Problem Statement**

What is NoSQL data base?

NoSQL databases are highly scalable and flexible database management systems which allow you to

store and process unstructured as well as semi-structured data which is not possible through RDBMS

tools.

The age of digitization has made the world generate massive amounts of data. Most of the

conventional databases are not prepared to handle these data seamlessly which gives rise

to the need to create the kind of databases which do not require modeling and structuring

the data structures before storing it.

NoSQL (Not only SQL) came as a solution to this problem of relational database

management systems and allowed the companies to store massive amounts of structured,

semi-structured and unstructured data in real-time. It does not certainly imply that it restricts

the usage of SQL for these databases. Some of the popular NoSQL databases are HBase,

Cassandra, IBm Informix, MongoDB, Amazon SimpleDB, Cloudata, etc. Today most of the

world famous firms like Google, Facebook, Amazon, etc., are using NoSQL to provide

cloud-based services in real-time.

2. How does data get stored in NoSQl database?

NoSQL stands for Not-Only- SQL and what it gives you is an amazing scalability, lightning fast

performance, ease of use and no schema limitations. Of course, different NoSQL solutions offer a variety

of feature alterations (increased performance with lower ACIDity being the general idea in most cases).

As a developer, while selecting a NoSQL solution you should think of what you need from your data

store and pick one which focuses its feature on addressing your main concerns. In some cases you might

be willing to sacrifice full data consistency in order to improve write speed. In some other you will

prioritize read times over write times or perhaps the other way around.

Columnar Databases – Reads and writes columns of data rather than the rows. Each

column is comparable to a container in RDBMS where a Key defines a row and single row

has multiple columns.

Document Databases – These databases store and retrieve semi-structured data in the

format of documents such as XML, JSON, etc. Some of the popular document databases

like MongoDB provide a rich query language for ease of access and smooth transition of

data models.

Graph Databases – Stores data as entities and relations between them allowing faster

traversal and joining operations to be performed. However these graphs can be built using

SQL as well as NoSQL databases.

In-Memory Key-Value Stores- Suitable for read-heavy workloads and compute-intensive

workloads, these databases store critical data in memory which in turn improves the

performance of the systems.

**3. What is a column family in HBase?**

A column family defines shared features to all columns that are created within them (think of it almost as a sub-table within your larger table). You will notice that HBase columns are composed of combination of the column family and column qualifier (or column key): &#39;family:qualifier&#39;

Columns in Apache HBase are grouped into column families. All column members of a column family have the same prefix. For example, the columns courses:history and courses:math are both members of the courses column family. The colon character (:) delimits the column family from the . The columnfamily prefix must be composed of printable characters. The qualifying tail, the column family qualifier,

can be made of any arbitrary bytes. Column families must be declared up front at schema definitiontime whereas columns do not need to be defined at schema time but can be conjured on the fly while the table is up an running.

Physically, all column family members are stored together on the filesystem. Because tunings andstorage specifications are done at the column family level, it is advised that all column family members

have the same general access pattern and size characteristics.

**4. How many maximum number of columns can be added to HBase table?**

There is no special limit to number of columns in HBase, we can have more than 1 million columns but usually three column families are recommended (not more than three).But there is a potential issue where we could have ‘too wide’ rows (many columns),and if you don’t specify exact qualifiers any scan will result whole rows so you could get much more data than you actually need. Also, it could possibly slow intra-row scanning to get needed column inside row.

**5. Why columns are not defined at the time of table creation in HBase?**

HBase has the characteristic of **Schema-Less design:** Columns in HBase don’t need to be defined up front so they provide a flexible way of managing evolving schemas. However columns can’t be renamed or assigned easily from one column-family to the other. Making such changes requires creation of new columns, migration of data from existing columns to the new column and then potentially deletion of old columns.

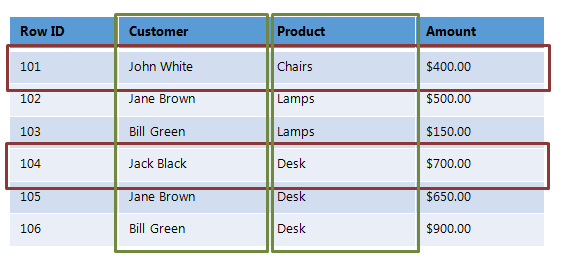
HBase is effectively a map of a map, so it has no schema. The user can define the columns at runtime, and each row can have its own columns. The responsibility is on the application to interpret the values stored in the HBase. This makes HBase very suitable for applications in which the schema is flexible. Also, unlike a database in which a separate metadata table is needed to describe the page identifiers, each column key describes itself in HBase.

**6. How does data get managed in HBase?**

[HBase](http://hbase.apache.org/) is a column-oriented database that’s an open-source implementation of Google’s [Big Table](http://research.google.com/archive/bigtable.html) storage architecture. It can manage structured and semi-structured data and has some built-in features such as scalability, versioning, compression and garbage collection.

Since its uses write-ahead logging and distributed configuration, it can provide fault-tolerance and quick recovery from individual server failures. HBase built on top of Hadoop / HDFS and the data stored in HBase can be manipulated using Hadoop’s MapReduce capabilities.

Let’s now take a look at how HBase (a column-oriented database) is different from some other data structures and concepts that we are familiar with Row-Oriented vs. Column-Oriented data stores. As shown below, in a row-oriented data store, a row is a unit of data that is read or written together. In a column-oriented data store, the data in a column is stored together and hence quickly retrieved.

[](http://www.netwoven.com/wp-content/uploads/2013/10/hbase-12.png)

Row-oriented data stores –

Data is stored and retrieved one row at a time and hence could read unnecessary data if only some of the data in a row is required.

Easy to read and write records

Well suited for OLTP systems

Not efficient in performing operations applicable to the entire dataset and hence aggregation is an expensive operation

Typical compression mechanisms provide less effective results than those on column-oriented data stores

Column-oriented data stores –

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

Introduction Relational Databases vs. HBase

When talking of data stores, we first think of Relational Databases with structured data storage and a sophisticated query engine. However, a Relational Database incurs a big penalty to improve performance as the data size increases. HBase, on the other hand, is designed from the ground up to provide scalability and partitioning to enable efficient data structure serialization, storage and retrieval. Broadly, the differences between a Relational Database and HBase are:

Relational Database –

Is Based on a Fixed Schema

Is a Row-oriented datastore

Is designed to store Normalized Data

Contains thin tables

Has no built-in support for partitioning.

HBase –

Is Schema-less

Is a Column-oriented datastore

Is designed to store Denormalized Data

Contains wide and sparsely populated tables

Supports Automatic Partitioning

HDFS vs. HBase

HDFS is a distributed file system that is well suited for storing large files. It’s designed to support batch processing of data but doesn’t provide fast individual record lookups. HBase is built on top of HDFS and is designed to provide access to single rows of data in large tables. Overall, the differences between HDFS and HBase are

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

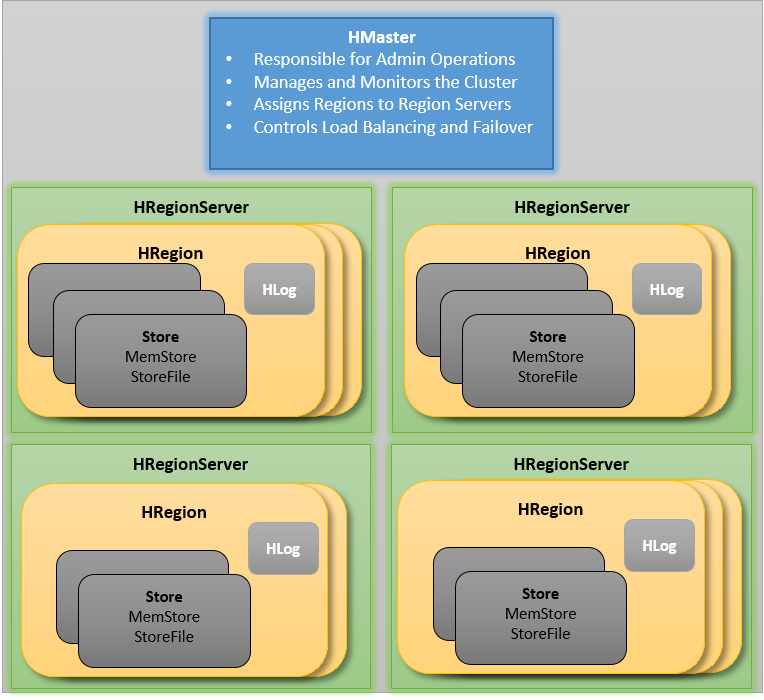
Provides access to single rows from billions of records

Data is accessed through shell commands, Client APIs in Java, REST, Avro or Thrift

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.

[](http://www.netwoven.com/wp-content/uploads/2013/10/hbase-11.png)

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

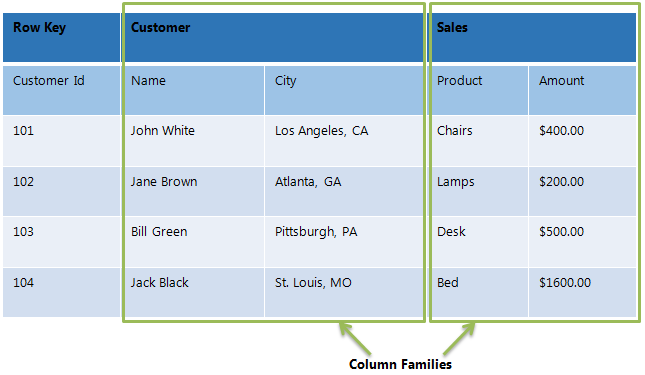
Communicating with the Clients directly

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)

HBase Data 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.

[](http://www.netwoven.com/wp-content/uploads/2013/10/hbase-2.png)

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.

**7. What happens internally when new data gets inserted into HBase table?**

1. The client gets the Region server that hosts the META table from ZooKeeper.
2. The client will query the .META. server to get the region server corresponding to the row key it wants to access. The client caches this information along with the META table location.
3. It will get the Row from the corresponding Region Server.

For future reads, the client uses the cache to retrieve the META location and previously read row keys. Over time, it does not need to query the META table, unless there is a miss because a region has moved; then it will re-query and update the cache.

* MemStore: is the write cache. It stores new data which has not yet been written to disk. It is sorted before writing to disk. There is one MemStore per column family per region.
* Hfiles store the rows as sorted KeyValues on disk.

**HBase Write Steps (1)**

When the client issues a Put request, the first step is to write the data to the write-ahead log, the WAL:

- Edits are appended to the end of the WAL file that is stored on disk.

- The WAL is used to recover not-yet-persisted data in case a server crashes.

**HBase Write Steps (2)**

Once the data is written to the WAL, it is placed in the MemStore. Then, the put request acknowledgement returns to the client.

**HBase MemStore**

The MemStore stores updates in memory as sorted KeyValues, the same as it would be stored in an HFile. There is one MemStore per column family. The updates are sorted per column family.

**HBase Region Flush**

When the MemStore accumulates enough data, the entire sorted set is written to a new HFile in HDFS. HBase uses multiple HFiles per column family, which contain the actual cells, or KeyValue instances. These files are created over time as KeyValue edits sorted in the MemStores are flushed as files to disk.

**HBase HFile**

Data is stored in an HFile which contains sorted key/values. When the MemStore accumulates enough data, the entire sorted KeyValue set is written to a new HFile in HDFS. This is a sequential write. It is very fast, as it avoids moving the disk drive head.