**Big Data Assignment 4**

**1. How can we set the block size for HBASE?**

HBase data are stored as StoreFile in the HFile format. StoreFiles are composed of HFile blocks. HFile block is the smallest unit of data that HBase reads from its StoreFiles. It is also the basic element that region server caches in the block cache. The size of the HFile block is an important tuning parameter. To achieve better performance, we should select different block sizes, based on the average Key/Value size and disk I/O speed. Like block cache and Bloom Filter, HFile block size is also configurable at the column family level. The default blocksize is 64 KB. We can use the following guidelines to tune the blocksize size, in combination with testing and benchmarking as appropriate.

* Consider the average key/value size for the column family when tuning the blocksize. You can find the average key/value size using the HFile utility:
  + 
* Consider the pattern of reads to the table or column family. For instance, if it is common to scan for 500 rows on various parts of the table, performance might be increased if the blocksize is large enough to encompass 500-1000 rows, so that often, only one read operation on the HFile is required. If your typical scan size is only 3 rows, returning 500-1000 rows would be overkill. It is difficult to predict the size of a row before it is written, because the data will be compressed when it is written to the HFile. Perform testing to determine the correct blocksize for your data.

**Configuring the Blocksize for a Column Family**

* You can configure the blocksize of a column family at table creation or by disabling and altering an existing table. These instructions are valid whether or not you use Cloudera Manager to manage your cluster.
* hbase> create ‘test\_table′,{NAME => ‘test\_cf′, BLOCKSIZE => '262144'}
* hbase> disable 'test\_table'
* hbase> alter 'test\_table', {NAME => 'test\_cf', BLOCKSIZE => '524288'}
* hbase> enable 'test\_table'
* After changing the blocksize, the HFiles will be rewritten during the next major compaction. To trigger a major compaction, issue the following command in HBase Shell.
* hbase> major\_compact 'test\_table'
* Depending on the size of the table, the major compaction can take some time and have a performance impact while it is running.

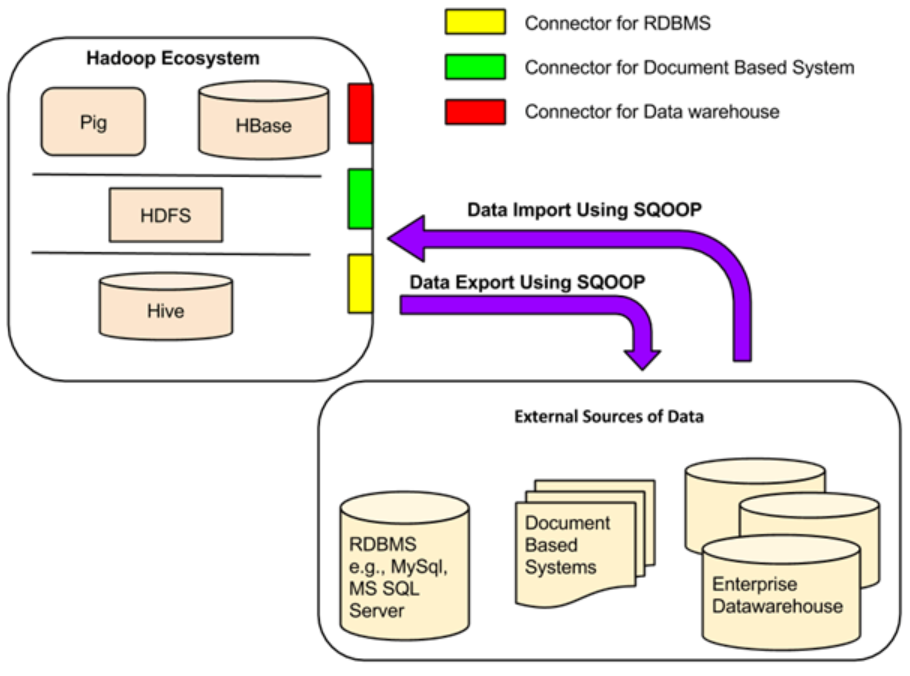
**2. What is Zookeeper's purpose?**

* HBase uses ZooKeeper as a distributed coordination service for region assignments and to recover any region server crashes by loading them onto other region servers that are functioning.
* ZooKeeper is a centralized monitoring server that maintains configuration information and provides distributed synchronization. Whenever a client wants to communicate with regions, they have to approach Zookeeper first.
* HMaster and Region servers are registered with ZooKeeper service, client needs to access ZooKeeper in order to connect with region servers and HMaster.
* In case of node failure within an HBase cluster, ZooKeeper will trigger error messages and start repairing failed nodes.
* ZooKeeper service keeps track of all the region servers that are there in an HBase cluster- tracking information about how many region servers are there and which region servers are holding which DataNode. HMaster contacts ZooKeeper to get the details of region servers. Various services that Zookeeper provides include –
  + Establishing client communication with region servers.
  + Tracking server failure and network partitions.
  + Maintain Configuration Information
  + Provides ephemeral nodes, which represent different region servers.

**3. Explain the architecture of Sqoop in detail.**

* SQOOP, which stands for “SQL-to-Hadoop”, is a tool designed to transfer data between relational database(s) and Hadoop.
* It facilitates bidirectional exchange of data between relational databases (RDBMS) such as MySQL or Oracle and the Hadoop Distributed File System (HDFS)
* SQOOP can also import data into Hive.
* It uses MapReduce to read data from source tables.

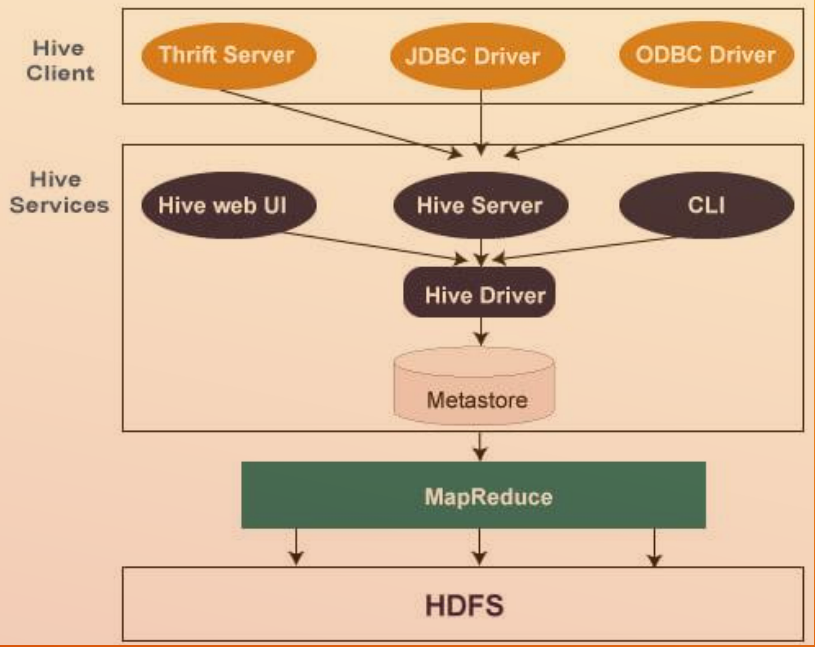
**Sqoop Architecture:**

* ****
* Data transfer between Sqoop Hadoop and external storage system is made possible with the help of Sqoop’s connectors.
* Sqoop has connectors for working with a range of popular relational databases, including MySQL, PostgreSQL, Oracle, SQL Server, and DB2. Each of these connectors knows how to interact with its associated DBMS. There is also a generic JDBC connector for connecting to any database that supports Java’s JDBC protocol. In addition, Sqoop Big data provides optimized MySQL and PostgreSQL connectors that use database-specific APIs to perform bulk transfers efficiently.
* In addition to this, Sqoop in big data has various third-party connectors for data stores, ranging from enterprise data warehouses (including Netezza, Teradata, and Oracle) to NoSQL stores (such as Couchbase). However, these connectors do not come with Sqoop bundle; those need to be downloaded separately and can be added easily to an existing Sqoop installation.
* Apache Sqoop provides the command-line interface to its end users. We can also access Sqoop via Java APIs. The Sqoop commands which are submitted by the end-user are read and parsed by the Sqoop. The Sqoop launches the Hadoop Map only job for importing or exporting data.
* No Reduce job is launched because the Reduce phase is needed only when the aggregations are performed. Apache Sqoop just imports and exports data, and hence it does not perform any aggregations due to which we don’t require a Reduce phase.
* Apache Sqoop parses the arguments which are provided in the command line and launches the Map only job. The Map only job launches multiple mappers depending on the number defined by the user in the command line.
* **Sqoop Import:**
  + For import, each mapper task is assigned with the part of data that is to be imported on the basis of the key defined in a command line. For getting higher performance, Sqoop distributes input data equally amongst all the mappers.
  + Each mapper then creates a connection with the database by using the JDBC and fetches part of the data assigned by the Sqoop. They then write those data into HDFS or HBase or Hive on the basis of the option provided in the command line.
* **Sqoop Export:**
  + Sqoop Export also works in the same way. The Sqoop Export tool exports the set of files from the Hadoop Distributed File System back to the Relational Database. The files which are given as an input to the Sqoop contain records. These records are called as rows in a table.
  + When the user submits it Job, then it is mapped into the Map Tasks that bring chunks of data from the Hadoop Distributed File System. These chunks are then exported to any structured data destination.
  + By combining all these chunks of data, the user receives the entire data at destination, which is generally an RDBMS such as MYSQL, SQL Server, Oracle, etc.

**4. In Hadoop, write the difference between Sqoop and HDFS.**

SQOOP, is a tool designed to transfer data between relational database(s) and Hadoop. It facilitates bidirectional exchange of data between relational databases (RDBMS) such as MySQL or Oracle and the Hadoop Distributed File System (HDFS). SQOOP can also import data into Hive. It uses MapReduce to read data from source tables. The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware. It is used to scale a single Apache Hadoop cluster to hundreds (and even thousands) of nodes. HDFS is one of the major components of Apache Hadoop, the others being MapReduce and YARN. HDFS is a key part of the many Hadoop ecosystem technologies. It provides a reliable means for managing pools of big data and supporting related big data analytics applications.

**5. Describe the Hive architecture's essential components.**

* A data warehouse solution built on top of Hadoop - by Facebook that can create interaction between user and HDFS. It provides an SQL dialect (called as Hive Query Language) for querying data stored in the HDFS. Automatically uses HDFS for storage, but stores all the meta information about database and table in metadata DB locally to Hive. The Hive architecture is shown below:
* 

The essential components of Hive are explained below:

* **Hive Client:**Hive allows writing applications in various languages, including Java, Python, and C++. It supports different types of clients such as:-
  + **Thrift Server**- It is a cross-language service provider platform that serves the request from all those programming languages that supports Thrift.
  + **JDBC Driver**- It is used to establish a connection between hive and Java applications. The JDBC Driver is present in the class org.apache.hadoop.hive.jdbc.HiveDriver.
  + **ODBC Driver** - It allows the applications that support the ODBC protocol to connect to Hive.

* **Hive Services:**The following are the services provided by Hive:-
  + **Hive CLI -**The Hive CLI is a shell where we can execute Hive queries and commands.
  + **Hive Web UI -** The Hive Web UI is just an alternative of Hive CLI. It provides a web-based GUI for executing Hive queries and commands.
  + **Hive MetaStore -**It is a central repository that stores all the structure information of various tables and partitions in the warehouse. It also includes metadata of column and its type information, the serializers and de-serializers which is used to read and write data and the corresponding HDFS files where the data is stored.
  + **Hive Server -** It is referred to as Apache Thrift Server. It accepts the request from different clients and provides it to Hive Driver.
  + **Hive Driver -** It receives queries from different sources like web UI, CLI, Thrift, and JDBC/ODBC driver. It transfers the queries to the compiler.
  + **Hive Compiler -**The purpose of the compiler is to parse the query and perform semantic analysis on the different query blocks and expressions. It converts HiveQL statements into MapReduce jobs.
  + **Hive Execution Engine -**Optimizer generates the logical plan in the form of DAG of map-reduce tasks and HDFS tasks. In the end, the execution engine executes the incoming tasks in the order of their dependencies.