Lecture Notes

Big Data Ingestion with Hive and Sqoop

In this module, you will learn about two specific tools — Apache Sqoop and Apache Hive. Sqoop helps you transfer data from RDBMSs to Hadoop, and vice versa. Hive also allows you to process data using a SQL-like language. This document covers the concepts associated with Hive and Sqoop. All the codes and analysis should be referred through the learning platform.

Apache Hive

Hive is a data warehouse software that enables you to query and manipulate data using an SQL-like language. Apache Hive was created to help analysts and engineers query the big data stored in Hadoop (HDFS). For running Hive queries, we use a software called Hue, which is an interactive environment.

The main utility of Hive is that it enables analysts to write SQL-like commands that work with the large scale of data. You write commands in SQL, and Hive translates them into MapReduce code. This prevents the task of writing individual MapReduce tasks for the individual. There are various useful conditional, string, and built-in aggregate functions for easy querying and processing. These functions are similar to SQL (SELECT, UPDATE, INSERT, DELETE, etc.).

Another benefit that Hive has is that Hive is capable of processing unstructured (or rather, semi-structured) logs. Besides, it is also used to parse unstructured data and store it in a query-able structured form. This feature is absent in the conventional RDBMSs and SQL, which require structured data for processing.

The Hive Metastore

The Hive Metastore is the central repository of the information stored. It stores all the metadata about the data stored in the HDFS, such as the names of the tables, the columns, the data types, the dates of creation of the respective tables, etc., but not the data itself. This helps you to understand the structure or the schema about how the data is stored in the HDFS and can be used by external platforms to run queries as well.

Hive stores and queries data using its data models. The purpose of using data models is to make querying convenient and fast. There are four main components in Hive data models:

- Databases
- Tables
- Partitions
- Buckets

Hive Tables

Hive has two types of tables:

- Managed (or internal) table
- External table

A crucial advantage of external tables is that even when you drop a table, the data is still available for use to other programs running on the same cluster, such as Spark, Pig, etc. On the other hand, when you drop (or delete) a managed table, both the metadata and the data are deleted.

You should use external tables when:

- You want to use the data outside of Hive as well. For example, when another existing program is running on the same cluster
- You want the data to remain stored on the HDFS even after dropping tables because Hive does not delete the data stored outside (of the Hive database).
- You do not want Hive to control the storage of your data (location/directories of storage/etc.).

On the other hand, you use managed tables when:

- The data is temporary. So, when the Hive table is dropped, the data stored in the internal table is deleted too.
- You want Hive to manage the life cycle of the data completely, i.e. both store and process it.

SerDes: Serialiser and Deserialiser

SerDes is basically a library that Hive uses to read and write table rows to and from various file formats.
The *deserialiser* is used to read files in a given format into Hive, while the *serialiser* is used to write files back to S3 or the HDFS (in a specified format). In the lectures you came across the JSON files and how they were loaded into a table using the SerDes.

Partitions

Partitioning is a way to segregate data into multiple directories using features present in the table. For example, you can segregate the data into different categories based on the years, months, gender, etc. Partitioning creates subdirectories, which allows relevant subdirectories to be fetched at the time of querying. The entire table is not read. The benefit here is that while querying, Hive will access only the relevant directories, and thus boost querying speed. This also prevents the files or the tables to grow to huge size where querying is slow and inefficient.

However, with the increase in the number of directories, the Hive metastore becomes loaded with information of each directory. Hence, it's not a good practice to create a very large number of directories through partitioning, since that will increase the amount of data the metastore has to store and may rather degrade the performance.

Bucketing

Bucketing (also called clustering) distributes the data uniformly in various buckets. This is fundamentally different from partitioning. Though bucketing is like partitioning in a manner that it divides the data into smaller and manageable parts, the difference between partitioning and bucketing is significant. Indeed, it is

this difference that decides which one of these to use in which case. To compare the two techniques, consider the following example:

- You typically perform partitioning on columns (or groups of columns) such that it produces a manageable number of partitions (directories), such as year, month, state, country etc.
- However, if you partition on date, customer ID, etc., the number of partitions will be enormous and may reduce performance; thus, you can choose to create buckets on such variables

It is important to note that bucketing uses the hash of a column to create multiple 'buckets', whereas partitions are more straightforward - they are simply segmented directories according to the value of a column (year, month etc.)

ORC (Optimized Row Columnar) format

An ORC file stores the data in an optimised way. The file contains the data stored in forms of rows (stripes) and footer. The actual information in the file is stored in the stripes and the information or the metadata for it is stored in the footer for decoding the information stored in the stripes. The size of one stripe is 10000 rows. To learn about the structure of an ORC file, you can refer this link.

ORC is one of the most optimised way of storing the data and by using compressed file formats, Hive can improve performance while reading, writing, and processing data. Compression helps when you have to transfer the data to various components of the cluster, such as to the nodes, from the HDFS to Hive, etc. Compression drastically reduces data transfer time. The only thing required to work with ORC is Hive is Serdes (serialisers and deserialisers) associated with the ORC format.

Since everything associated with Hive cannot be covered in the document, here is an exhaustive cheat sheet on Apache Hive for your aid.

Apache Sqoop

Data ingestion is the process of accessing and importing data for immediate use or storage from one system to another. Data ingestion is necessary since traditional systems such as MySQL databases, Teradata, Netezza, Oracle, etc. often become incapable of handling growing volumes of data, and you need to ingest the data from these systems into big data systems.

As more organisations and firms started using big data platforms such as Hadoop, they needed to transfer large amounts of data between Hadoop and their existing RDBMSs. This need resulted in the development of data ingestion tools such as Sqoop and Flume. Note that Sqoop is used to ingest structured data, while Flume is a tool used for semi-structured or unstructured data. Since most of the data stored in RDBMSs is structured, Sqoop is more commonly used than Flume.

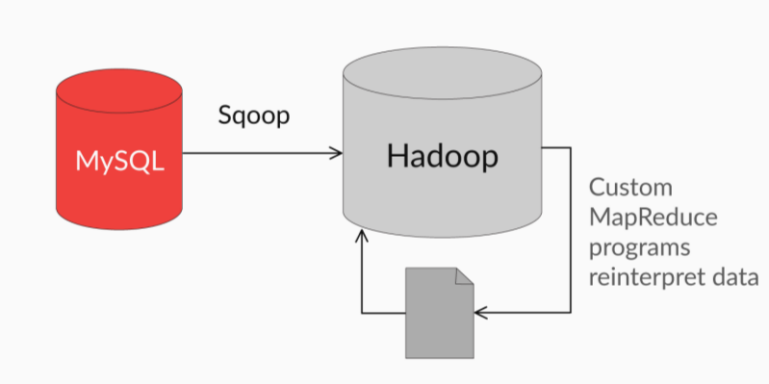

Figure 1: Apache Sqoop: Bridge

There are the three broad use cases of Sqoop in the industry:

- Importing data from RDBMSs
- Exporting data to RDBMSs
- Interactive analytics on RDBMSs

How Sqoop Performs Import

Importing large datasets is not a trivial task and that it can be an extremely time-consuming process. Here, import means bringing data into the Hadoop ecosystem. Imagine ingesting hundreds of gigabytes of data into Hadoop. You do not want a tool to take hours to import data and slow down your analysis. The steps involved in an import command using the Apache Sqoop are:

1. **Making a connection with the database/RDBMS**: In the first step, Sqoop makes a connection with the RDBMS using what is called a 'connector'. Connectors allow Sqoop to ignore the differences in the 'SQL

dialects' of different databases (e.g. Teradata, Netezza, Oracle, MySQL, PostgreSQL) and allow it to treat all of them simply as RDBMSs.

2. **Reading the table to be imported and fetching the metadata**: After making a connection, Sqoop inspects the table to be imported and collects metadata such as the number of rows, number of columns, their names, the data types, etc.
3. **Launching a MapReduce program to perform parallel imports**: In this step, Sqoop divides the data into multiple parts and assigns a mapper to each part. Each of these mappers transfers a part of the table to Hadoop in a parallelised manner.

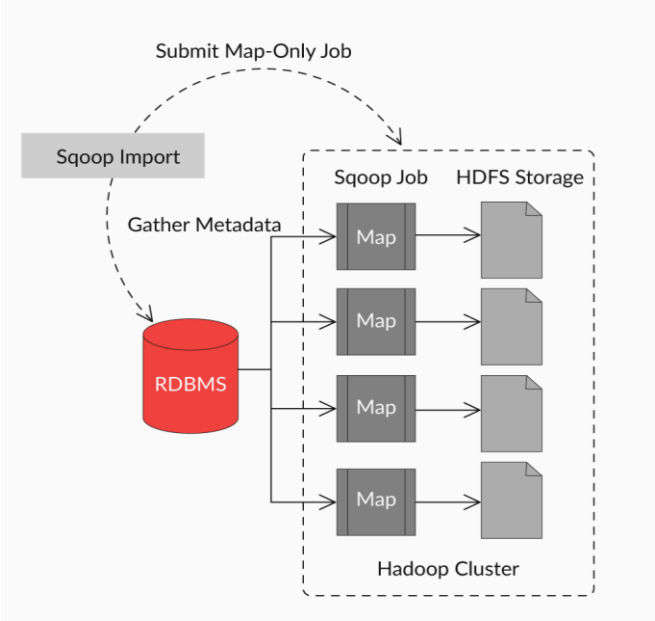

Figure 2: Apache Sqoop: Import from RDBMS into HDFS

The basic idea of importing large datasets is to divide them into multiple parts and import these parts parallelly using a MapReduce-like paradigm. As is the standard case, Sqoop runs the map phase first and divides the data into multiple smaller parts, each of which is assigned to a mapper. However, there is no need for a reduce phase since there is no aggregation required - we are writing into an HDFS type destination!

While performing parallel imports, Sqoop uses a splitting column to split the workload. By default, it identifies the primary key column (if present) in a table and uses it as the splitting column. The highest and lowest values of the splitting column are retrieved from the database, and the map tasks operate on the individual subsets of the table.

For example, if the table has a primary key column 'user_ID' (as integers), whose minimum value is '0' and maximum value is 'n', and Sqoop is directed to use, say, four tasks (the default number of mappers is four), it would run four processes (one for each task) using SQL statements of the form

SELECT * FROM sometab WHERE id ≥ lo AND id < hi

with (lo, hi) set to (0, n/4), (n/4, n/2), (n/2, 3n/4), and (3n/4, n+1) in the four tasks respectively.

To summarise, Sqoop follows the following procedure for data ingestion:

- It looks at the range of the primary key (from the splitting/primary key column).

- It sets the lower value of the primary key to some variable.
- It sets the higher value of the primary key to another variable.
- It generates SQL queries to fetch the data parallelly.

If the values of the primary key column/splitting column are not uniformly distributed across its range, this can result in unbalanced tasks. In such cases, you are advised to choose a different column using the '--split-by' clause explicitly.

For example, to specify that Sqoop should use the 'customer_ID' column for splitting, you can write "--split-by customer_ID". The export operation is similar to the import operation, except that it works in the reverse direction. In the export operation too, the number of maps is specified, and then, the respective data blocks are assigned to each mapper program for transfer.

Choosing the Optimal Number of Mappers

You can also specify the number of mappers manually, in which case, you need to choose an appropriate number of them. In general, using more mappers will lead to a higher degree of parallelisation, leading to a higher speed. However, it will also increase the load on the database, which may affect the other requests your database needs to handle. Also, most databases support only a limited number of parallel connections, and this number is the upper bound you have for the number of mappers. Thus, you need to use a 'hit-and-trial' approach to find the optimal number of mappers, which will be specific to your hardware, database, types of tables, etc. Similarly, Sqoop is also used for the process of exporting data from HDFS to RDBMSs.