Integrating Hadoop with Relational Databases using Sqoop

B Nandana Kumar[1]
Assistant Professor
DNR College of Engg. Tech.

L Bujji Babu[2]
Assistant Professor
DNR College of Engg. Tech.

***Abstract*:- Apache Sqoop is a tool designed for efficiently transferring bulk data between Apache Hadoop and external data stores such as relational databases, enterprise data warehouses. Sqoop is used to import data from external data stores into Hadoop Distributed File System or related Hadoop eco-systems like Hive and HBase. Similarly, Sqoop can also be used to extract data from Hadoop or its eco-systems and export it to external data stores such as relational databases, enterprise data warehouses. Sqoop works with relational databases such as Teradata, Netezza, Oracle, MySQL, Postgres etc.**

Performing analytics on large, diverse data sets is a natural fit for Apache Hadoop. The whole point of the Hadoop File System (HDFS) is that it excels at providing a massively scalable, diverse data store that, when combined with the many analytic tools available on the Hadoop platform — from Map Reduce to Mahout and others — gives you a lean, mean, analytics machine when you hitch your data store wagon to Apache Hadoop.

Keywords: Sqoop, Hive, Netezza, HDFS, Map Reduce

INTRODUCTION:

Sqoop is a tool designed to transfer data between Hadoop and relational databases or mainframes. You can use Sqoop to import data from a relational database management system (RDBMS) such as MySQL or Oracle or a mainframe into the Hadoop Distributed File System (HDFS), transform the data in Hadoop MapReduce, and then export the data back into an RDBMS. Sqoop automates most of this process, relying on the database to describe the schema for the data to be imported. Sqoop uses MapReduce to import and export the data, which provides parallel operation as well as fault tolerance. This rosy picture presents a slight problem, however: It turns out that most of the world's structured data is already stored in relational database management systems (RDBMSs), and it's common practice to leverage structured query language (SQL, for short) for data transformation, processing, and analysis — and SQL is decidedly *not* a natural fit for Apache Hadoop. Sqoop was first announced in 2009 by Aaron Kimball as a database import tool for Hadoop, and three years later (March 2012, to be exact), Sqoop became a top-level Apache project. The glory of Sqoop lies in the fact that it not only allows you to import relational data but also provides an export mechanism. The result is that Sqoop can provide an efficient mechanism for loading an RDBMS table by exporting data stored in HDFS, a use case perfectly suited for scenarios where you make use of Hadoop as an enterprise data warehouse (EDW) preprocessing engine. Sqoop has grown a lot since

The Principles of Sqoop Design:

When it comes to Sqoop, a picture is often worth a thousand words, so check out Figure 1-1, which gives you a bird's-eye view of the Sqoop architecture.

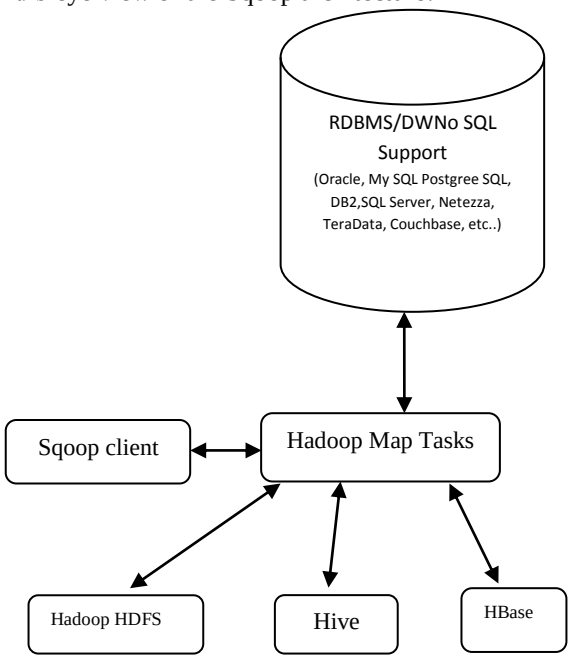

Figure1-1. Sqoop design

The idea behind Sqoop is that it leverages *map* tasks — tasks that perform the parallel import and export of relational database tables right from within the Hadoop MapReduce framework. This is good news because the MapReduce framework provides fault tolerance for import and export jobs along with parallel processing! You'll appreciate the fault tolerance if there is a failure during a large table import or export because the MapReduce framework will recover without requiring you to start the process all over again. Sqoop can import data to Hive and HBase. Note, however, that the arrows to Hive and HBase point in only one direction in Figure1-1. Data stored in any relational database with JDBC support can be directly imported into the Hive or HBase systems with Sqoop. Exports, however, are performed from data stored in HDFS. Therefore, if you need to export your Hive tables, you point Sqoop to HDFS directories that store your Hive

tables. If you need to export HBase tables, you first have to export them to HDFS and then execute the Sqoop export command.

SCOOPING UP DATA WITH SQOOP:

Sqoop provides Hadoop with export and import capability to and from any RDBMS or data warehouse (DW) that supports the Java Database Connectivity (JDBC) application programming interface (API) suite. All major RDBMS and DW vendors generally provide JDBC-compliant drivers for their products. In addition, Sqoop releases are bundled with special connector technology for a variety of popular products. As of this writing, Sqoop version 1.4.4 provides special connectors for MySQL, PostgreSQL, Oracle, Microsoft SQL Server, DB2, and Netezza. These special connectors take advantage of specific features within the individual database systems in order to improve import/export performance and functionality. Additionally, third-party connectors are available that aren't bundled with Sqoop for other NoSQL data store and data warehouse providers (Couchbase and Teradata from Cloudera, for example). Sqoop also includes a generic JDBC connector that only supports the Java JDBC API.

CONNECTORS AND DRIVERS:

Sqoop connectors generally go hand in hand with a JDBC driver. Sqoop does not bundle the JDBC drivers because they are usually proprietary and licensed by the RDBMS or DW vendor. So there are three possible scenarios for Sqoop, depending on the type of data management system (RDBMS, DW, or NoSQL) you are trying to interact with. Let's take a look at each one:

✓ **Your data management system is supported by one of the bundled Sqoop connectors listed above.** In this case, you need to acquire the JDBC driver from your data management system provider and install the .jar file associated with it in your $SQOOP_HOME / lib directory. $SQOOP_HOME is an environment variable that refers to the directory pathname on your system where you install Apache Sqoop.

✓ **Sqoop does not include a connector for your database management system.** That means you need to download one from a 3rd party vendor, along with a JDBC driver if the connector requires one. (Couchbase and Teradata both do, for example.)

✓ **Your database management system does not provide a Sqoop connector but a JDBC driver is available.** In this case, you leverage Sqoop's generic JDBC connector and download and install your vendor's JDBC driver.

Importing Data with Sqoop:

It illustrates the steps in a typical Sqoop import operation from an RDBMS or a data warehouse system. Nothing too complicated here, just a typical Products data table from a (typical) fictional company being imported into a typical Apache Hadoop cluster from a typical data management system (DMS).

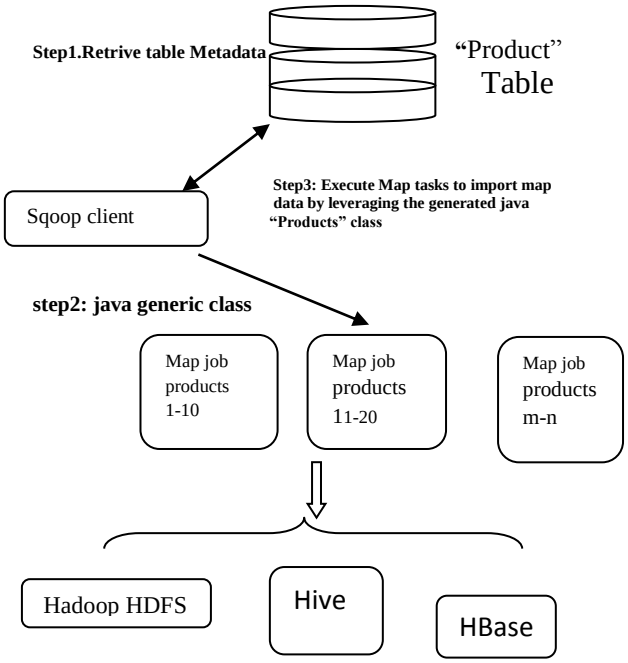

Figure:1-2 The Sqoop import flow of execution

During Step 1, Sqoop uses the appropriate connector to retrieve the Products table metadata from the target DMS. (The metadata is used to map the data types from the Products table to data types in the Java language.)

Step 2 then uses this metadata to generate and compile a Java class that will be used by one or more map tasks to import the actual rows from the Products table. Sqoop saves the generated Java class to temp space or to a directory you specify so that you can leverage it for the subsequent processing of your data records. The Sqoop generated Java code that is saved for you is like the gift that keeps on giving! With this code, Sqoop imports records from the DMS and stores them to HDFS using one of three formats that you can pick: binary Avro data, binary sequence files, or delimited text files. Afterwards, this code is available to you for subsequent data processing. Sequence files are a natural choice if you're importing binary data types and you'll need the generated Java class to serialize and desterilize your data later on perhaps for MapReduce processing or exporting. (More on exporting later - right now, we're focusing on imports.) Avro data based on Apache's own serialization framework is useful if you need to interact with other applications after the import to HDFS. If you choose to store your imported data in delimited text format, you may find the generated Java code valuable later on as you parse and perform data format conversions on your new data. Later in this chapter, you'll see that the generated code also helps you merge data sets after Sqoop import operations and the final example in this chapter illustrates how the generated Java code can help avoid ambiguity when processing delimited text data. Finally, during

Step 3, Sqoop divides the data records in the Products table across a number of map tasks (with the number of mappers optionally specified by the user) and imports the table data into HDFS, Hive, or HBase.

REFERENCES:

[1] HADOOP DUMMIES by Dirk deRoos, Paul C. Zikopoulos, Bruce Brown, Rafael Coss, and Roman B. Melnyk
[2] HADOOP IN PRACTISE By Alex Holmes
[3] HADOOP: The Definitive Guideby Tom White