

Integration of Big Data and Relational Database

Mrs. Archana Kalia
Lecturer in Information Technology Dept.
VPMs Polytechnic, Thane.

Abstract:-In classical data warehousing terms, organizing data is called data integration. Because there is such a high volume of big data, there is a tendency to organize data at its initial destination location, thus saving both time and money by not moving around large volumes of data. While big data may be massive, very often the amount of data that is relevant to a given query is smaller than the total data volume by an order of magnitude or more. This provides an opportunity for tremendous optimization in query performance. Hadoop is a great tool to help with this task. This paper discusses the various approaches to integrating new data systems into the existing data management and analytic environment. The paper focuses specifically on integrating the two key components of a big data ecosystem – relational database systems and the Hadoop distributed processing framework.

Keywords:-Big Data, HDFS, Map Reduce, Scoop

I. INTRODUCTION

As with data warehousing, web stores or any IT platform, an infrastructure for big data has unique requirements. In considering all the components of a big data platform, it is important to remember that the end goal is to easily integrate any big data with any enterprise data to allow us to conduct deep analytics on the combined data set. Hadoop is a great tool to help with this task. Hadoop was designed to distribute large volumes of multi-structured data across a hardware cluster consisting of hundreds, potentially thousands, of low-cost hardware servers. High performance is achieved using a divide-and-conquer approach that distributes the application processing across these servers. This parallel processing technique is particularly beneficial for applications that sequentially process large data files. Hadoop also includes a programming model, known as MapReduce, which isolates application programmers from the need to know how to code distributed processing programs. Hadoop framework includes following four modules:

- Hadoop Common: These are Java libraries and utilities required by other Hadoop modules. These libraries provide filesystem and OS level abstractions and contains the necessary Java files and scripts required to start Hadoop.
- Hadoop YARN: This is a framework for job scheduling and cluster resource management.
- Hadoop Distributed File System (HDFS): A distributed file system that provides high-throughput access to application data.
- Hadoop MapReduce: This is YARN-based system for parallel processing of large data sets.

II. HADOOP DISTRIBUTED FILE SYSTEM

Hadoop can work directly with any mountable distributed file system such as Local FS, HFTP FS, S3 FS, and others, but the most common file system used by Hadoop is the Hadoop Distributed File System (HDFS). The Hadoop Distributed File System (HDFS) is based on the Google File System (GFS) and provides a distributed file system that is designed to run on large clusters (thousands of computers) of small computer machines in a reliable, fault-tolerant manner. HDFS uses a master/slave architecture where master consists of a single NameNode that manages the file system metadata and one or more slave DataNodes that store the actual data.

A file in an HDFS namespace is split into several blocks and those blocks are stored in a set of Data Nodes. The Name Node determines the mapping of blocks to the Data Nodes. The Data Nodes take care of read and write operation with the file system. They also take care of block creation, deletion and replication based on instruction given by Name Node. HDFS provides a shell like any other file system and a list of commands are available to interact with the file system.

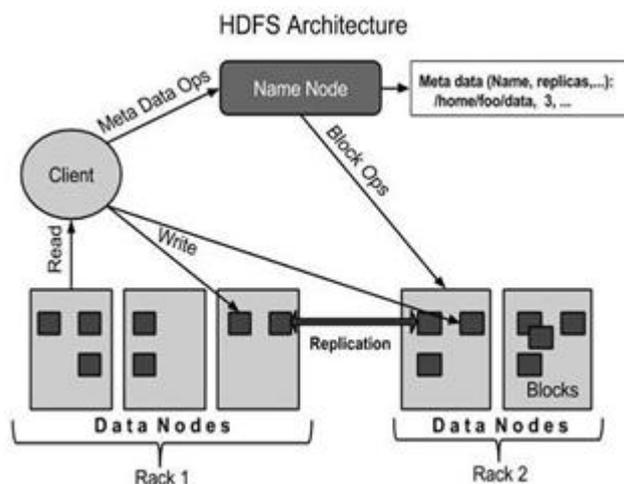


Figure 1.HDFS Architecture

III. MAPREDUCE

Hadoop MapReduce is a software framework for easily writing applications which process big amounts of data in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner. The term MapReduce actually refers to the following two different tasks that Hadoop programs perform:

The Map Task: This is the first task, which takes input data and converts it into a set of data, where individual elements are broken down into tuples (key/value pairs).

The Reduce Task: This task takes the output from a map task as input and combines those data tuples into a smaller set of tuples. The reduce task is always performed after the map task.

Typically both the input and the output are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks. The MapReduce framework consists of a single master JobTracker and one slave TaskTracker per cluster-node. The master is responsible for resource management, tracking resource consumption/availability and scheduling the jobs component tasks on the slaves, monitoring them and re-executing the failed tasks. The slaves TaskTracker execute the tasks as directed by the master and provide task-status information to the master periodically.

The JobTracker is a single point of failure for the Hadoop MapReduce service which means if JobTracker goes down, all running jobs are halted.

defined by a schema, whereas Hadoop uses key-value pairs as its fundamental unit.

The second major difference is how the data are queried. With Relational databases, users leverage SQL to specify what data they want more than how to obtain it. Hadoop uses Map Reduce programs, which can also be initiated via SQL-like commands, to specify what data to retrieve and also how to retrieve it.

And a third major difference between Hadoop and Relational databases is how they scale. Relational databases typically scale by adding lots of horsepower (RAM and CPU) to a single or small set of database-class servers. Hadoop databases scale by adding far more (often hundreds) but lower power – machines that work in parallel.

IV. MOVING FROM RELATIONAL DATABASE TO HADOOP

Although Apache Hadoop was created to process huge amounts of unstructured data, especially on the Internet, it is found that Hadoop is sitting alongside a relational database. This stems from the prevalence of relational databases. Hadoop cluster often means moving data from existing SQL tables, maybe hosted in SQL Server, MySQL or PostgreSQL, into HDFS or Hive so it can be accessed by all of the MapReduce goodness.

Sqoop is a tool designed to transfer data between Hadoop and relational databases. Sqoop is used to import data from a relational database management system (RDBMS) such as MySQL or Oracle into the Hadoop Distributed File System (HDFS), transform the data in Hadoop Map Reduce, and then export the data back into an RDBMS.

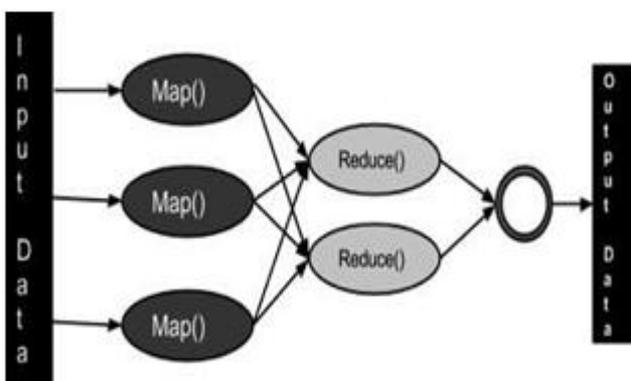


Figure 2. Map Reduce in Hadoop

So, now that there is a basis for the Hadoop technology, how does this compare to traditional relational databases?

For starters, the storage mechanisms are fundamentally different. Relational databases store information in tables

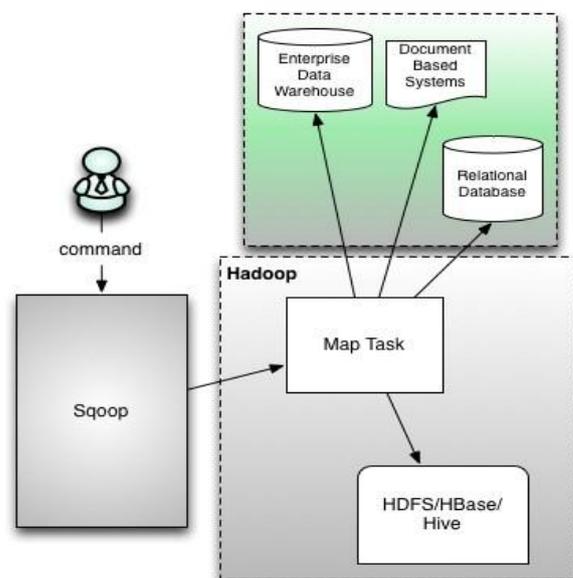


Figure 3. Sqoop architecture

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. Thus it is described below that how Sqoop helps to move data between databases and Hadoop and also provides reference information for the operation of the Sqoop command-line tool suite.

V. SQOOP-IMPORT

The import tool imports an individual table from an RDBMS to HDFS. Each row from a table is represented as a separate record in HDFS. Records can be stored as text files (one record per line), or in binary representation as Avro or Sequence Files.

VI. CONNECTING TO A DATABASE SERVER

Sqoop is designed to import tables from a database into HDFS. To do so, *connect string* must be specified that describes how to connect to the database. The *connect string* is similar to a URL, and is communicated to Sqoop with the `--connect` argument. This describes the server and database to connect to; It may also specify the port. For example:

```
$ sqoop import --connect  
jdbc:mysql://database.example.com/employees
```

This string will connect to a MySQL database named employees on the host database.example.com.

VII. SELECTING THE DATA TO IMPORT

Sqoop typically imports data in a table-centric fashion. The `--table` argument is used to select the table to import. For example, `--table employees`. This argument can also identify a VIEW or other table-like entity in a database.

By default, all columns within a table are selected for import. Imported data is written to HDFS in its "natural order;" that is, a table containing columns A, B, and C result in an import of data such as:

```
A1,B1,C1  
A2,B2,C2  
...
```

Selection of a subset of columns and control their ordering by using the `--columns` argument is possible. This should include a comma-delimited list of columns to import. For example: `--columns "name,employee_id,jobtitle"`.

It is possible to control which rows are imported by adding a SQL WHERE clause to the import statement. By default,

Sqoop generates statements of the form `SELECT <column list> FROM <table name>`. Appending a WHERE clause to this with the `--where` argument is also possible. For example: `--where "id > 400"`. Only rows where the id column has a value greater than 400 will be imported.

VIII. CONTROLLING PARALLELISM

Sqoop imports data in parallel from most database sources. When performing parallel imports, sqoop needs a criterion by which it can split the workload. Sqoop uses a *splitting column* to split the workload. By default, sqoop will identify the primary key column (if present) in a table and use it as the splitting column. The low and high values for the splitting column are retrieved from the database, and the map tasks operate on evenly-sized components of the total range. For example, if a table with a primary key column of id whose minimum value is 0 and maximum value is 1000, and sqoop is directed to use 4 tasks. Sqoop will run four processes in which each execute sql statements of the form `select * from sometable where id >= lo and id < hi`, with (lo, hi) set to (0, 250), (250, 500), (500, 750), and (750, 1001) in the different tasks.

IX. CONTROLLING THE IMPORT PROCESS

Sqoop will import a table named foo to a directory named foo inside a home directory in HDFS. For example, if username is someuser, then the import tool will write to `/user/someuser/foo/(files)`. Adjustment of the parent directory of the import with the `--warehouse-dir` argument is possible. For example:

```
$ sqoop import --connect <connect-str> --table foo --  
warehouse-dir /shared \  
...
```

This command writes to a set of files in the `/shared/foo/` directory. The target directory can also be explicitly chosen like:

```
$ sqoop import --connect <connect-str> --table foo --target-  
dir /dest \  
...
```

This will import the files into the `/dest` directory. `--target-dir` is incompatible with `--warehouse-dir`

X. FILE FORMATS

Data can be imported in one of two file formats: delimited text or Sequence Files.

Delimited text is the default Import format. It is specified explicitly by using the `--as-textfile` argument. This argument will write string-based representations of each record to the output files, with delimiter characters between individual columns and rows. These delimiters may be commas, tabs, or

other characters. The following is the results of an example text-based import:

```
1,here is a message,2010-05-01
2,happy new year!,2010-01-01
3,another message,2009-11-12
```

Delimited text is appropriate for most non-binary data types. It readily supports further manipulation by other tools, such as Hive.

Sequence Files are a binary format that store individual records in custom record-specific datatypes. These data types are manifested as Java classes. Sqoop will automatically generate these data types. This format supports exact storage of all data in binary representations.

XI. SQOOP-IMPORT-ALL-TABLES

The import-all-tables tool imports a set of tables from an RDBMS to HDFS. Data from each table is stored in a separate directory in HDFS.

For the import-all-tables tool to be useful, the following conditions must be met:

- Each table must have a single-column primary key.
- It is intended to import all columns of each table.
- It must not intend to use non-default splitting column, nor impose any conditions via a WHERE clause.

```
$ sqoop import-all-tables (generic-args) (import-args)
```

```
$ sqoop-import-all-tables (generic-args) (import-args)
```

Although the Hadoop generic arguments must precede any import arguments, the import arguments can be entered in any order with respect to one another.

XII. INCREMENTAL IMPORTS

Sqoop provides an incremental import mode which can be used to retrieve only rows newer than some previously-imported set of rows.

Sqoop supports two types of incremental imports: Append and Lastmodified. The incremental argument is used to specify the type of incremental import to perform.

Append mode is specified when importing a table where new rows are continually being added with increasing row id values. The column containing the row's id with --check-column is also specified. Sqoop imports rows where the check column has a value greater than the one specified with --last-value.

An alternate table update strategy supported by Sqoop is called lastmodified mode. This is used when rows of the

source table may be updated, and each such update will set the value of a last-modified column to the current timestamp. Rows where the check column holds a timestamp more recent than the timestamp specified with --last-value are imported.

At the end of an incremental import, the value which should be specified as --last-value for a subsequent import is printed to the screen. When running a subsequent import, it is specified as --last-value to ensure import only the new or updated data. This is handled automatically by creating an incremental import as a saved job, which is the preferred mechanism for performing a recurring incremental import.

XIII. IMPORTING DATA INTO HIVE

Sqoop's import tool's main function is to upload data into files in HDFS. If a Hive metastore is associated with HDFS cluster, Sqoop can also import the data into Hive by generating and executing a CREATE TABLE statement to define the data's layout in Hive. Importing data into Hive is as simple as adding the --hive-import option to Sqoop command line.

XIV. SQOOP-EXPORT

The export tool exports a set of files from HDFS back to an RDBMS. The target table must already exist in the database. The input files are read and parsed into a set of records according to the user-specified delimiters.

The default operation is to transform these into a set of INSERT statements that inject the records into the database. In "update mode," Sqoop will generate UPDATE statements that replace existing records in the database, and in "call mode" Sqoop will make a stored procedure call for each record.

```
$ sqoop export (generic-args) (export-args)
```

```
$ sqoop-export (generic-args) (export-args)
```

Although the Hadoop generic arguments must precede any export arguments, the export arguments can be entered in any order with respect to one another.

XV. EXPORTS AND TRANSACTIONS

Exports are performed by multiple writers in parallel. Each writer uses a separate connection to the database; these have separate transactions from one another. Sqoop uses the multi-row insert syntax to insert up to 100 records per statement. Every 100 statements, the current transaction within a writer task is committed, causing a commit every 10,000 rows. This ensures that transaction buffers do not grow without bound, and cause out-of-memory conditions. Therefore, an export is

not an atomic process. partial results from the export will become visible before the export is complete.

XVI. SQOOP-JOB

The job tool allows to create and work with saved jobs. Saved jobs remember the parameters used to specify a job, so they can be re-executed by invoking the job by its handle.

If a saved job is configured to perform an incremental import, state regarding the mostrecently importedrows isupdated in the saved job to allow the job to continually import only the newest rows.

```
$ sqoop job (generic-args) (job-args) [-- [subtool-name] (subtool-args)]
```

```
$ sqoop-job (generic-args) (job-args) [-- [subtool-name] (subtool-args)]
```

Although the hadoop generic arguments must precede any job arguments, the job arguments can be entered in any order with respect to one another.

XVII. CONCLUSION AND FUTURE WORK

Integrating relational databases with Hadoop may be necessary to get data flowing in organizations. Although it needs some fine tuning and has a few limitations, Sqoop is the right tool for the database job. Though Sqoop can be really helpful to move data from RDBMS to Hadoop and back again, but several issues could crop up during the process.

- Jdbc drivers: Sqoop works best with relational databases that support jdbc. This means the right drivers need to be installed on every machine in the hadoop cluster.

- Data types: Different databases support different data types. These data types convert to java types when exporting with sqoop, a conversion that may not always run smoothly

- Multiple Processes: Sqoop exports data from the DataBase in parallel using several processes. This is great since it speeds up conversion, but it also means the DB may be under strain while Sqoop is working its magic. Fortunately, the number of map tasks can be configured to control how many processes are used. Also, a table column should be set to split ranges of rows across Sqoop's processes; otherwise the data may not be exported efficiently because several processes may try to handle the same rows. As long as the relevant column is a primary key, this issue won't happen.

The lack of built-in security in Hadoop is an obstacle that enterprises face today, but new tools have to developed that solves this issue, connecting Kerberos and MapReduce

components and ensuring compatibility between data. It is expected that these issues to be resolved so that highly regulated organizations can manage their data securely with Hadoop and sqoop.

Also the Internet of things is only possible with instant data processing and prescriptive analytics. As more things enter the data ecosystem, the burden of processing will become greater and legacy technology will not be able to keep up. Hadoop will be able to, and it will be a mission-critical foundation for many businesses tied to the Internet of things.

REFERENCES

- [1] Integrating big data and relational data with a functional sql-like query language by Carlyna Bondiombouy¹, Boyan Kolev¹(&), Oleksandra Levchenko^{1,2}, and Patrick Valduries¹.
- [2] Integrating SQL and Hadoop Jean by Pierre Dijcks and Martin Gubar.
- [3] An Oracle White Paper September 2014- Oracle: Big Data for the Enterprise.
- [4] Big Data Analysis using R and Hadoop by Anju Gahlawat Tata Consultancy Services Ltd. 4 & 5 Floor, PTI Bldg, 4, Parliament St, Connaught Place, New Delhi.