MIDS W205 Project: Data Transformations

Introduction

The architecture for the weblog analysis calls for the transformed data to be stored in a relational database. This allows the flexibility for data scientist to use standard ODBC type tools to interface with the database to extract the new data and perform data analysis. In this document we will present the steps that we took to implement the proposed architecture using a Postgresql database.

Configuring Hive to Utilize Postgresql to Store Metadata

By default, Hive uses derby to store the metadata about the schemas, databases and tables it creates. It also stores the data in flat files. In order to easily insert the analyzed weblog data back into Postgres, we reconfigured Hive “metastore” to use a Postgresql database. The Postgresql software is installed in the AMI image that was provided as part of the class. Here are the steps required to achieve this:

1. We first ensure that the property “listen_address” was set to *, and “standard_conforming_strings” property is set to off in the postgresql.conf file. This can be see in the code snippet, executed as “root”, shown below:

```
$ cat /var/lib/pgsql/data/postgresql.conf  | grep -e listen -e standard_conformin
g_strings
listen_addresses = '*'
standard_conforming_strings = off
```

2. Next, we install “postgresql-jdbc” package and create symbolic link to the /usr/lib/hive/lib/ directory as shown below:

```
$ sudo yum install postgresql-jdbc
$ ln -s /usr/share/java/postgresql-jdbc.jar /usr/lib/hive/lib/postgresql-jdbc.jar
```

3. Creating the metastore database and user account is the next step. This is shown in the code. We used a password different than the one included in the code. Note that for the version of Hive that is provided in the AMI image (version 1.1.0-cdh5.4.5), the correct schema version is 1.1.0.

```
$ sudo -u postgres psql
postgres=# CREATE USER hiveuser WITH PASSWORD 'mypassword';
postgres=# CREATE DATABASE metastore;
postgres=# \c metastore;
You are now connected to database 'metastore'.
postgres=# \i /usr/lib/hive/scripts/metastore/upgrade/postgres/hive-schema-1.1.0.
postgres.sql
```

4. Once we have the database created and the correct schema has been applied, next we need to provide the appropriate permissions for the user, “hiveuser”, to access the database. This is shown in the code:

```
bash# sudo -u postgres psql
metastore=# \c metastore
metastore=# \pset tuples_only on
metastore=# \o /tmp/grant-privs
metastore=#   SELECT 'GRANT SELECT,INSERT,UPDATE,DELETE ON "'  || schemaname || '
". "' ||tablename ||'" TO hiveuser ;'
metastore-#   FROM pg_tables
metastore-#   WHERE tableowner = CURRENT_USER and schemaname = 'public';
metastore=# \o
metastore=# \pset tuples_only off
metastore=# \i /tmp/grant-privs
```

5. We need to configure Hive to use the postgresql database that we have created. This is achieved by modifying the file, /usr/lib/hive/conf/hive-site.xml. Specifically, the changes shown below have to be made for Hive to start using the postgres database.

```
<property>
<name>javax.jdo.option.ConnectionURL</name>
<value>jdbc:postgresql://localhost/metastore</value>
</property>
<property>
<name>javax.jdo.option.ConnectionDriverName</name>
<value>org.postgresql.Driver</value>
</property>
<property>
<name>javax.jdo.option.ConnectionUserName</name>
<value>hiveuser</value>
</property>
<property>
<name>javax.jdo.option.ConnectionPassword</name>
<value>mypassword</value>
</property>
<property>
<name>datanucleus.autoCreateSchema</name>
<value>false</value>
</property>
<property>
<name>hive.metastore.schema.verification</name>
<value>true</value>
</property>
```

Once these steps were performed, the configuration can be tested by executing the command `hive -e "show tables;"`. A succesful completion of the test is the precursor to running our weblog processing pipeline to utilize the Postgres database.

Using Postgresql Database to Store Processed Weblog Data

As mentioned earlier, to facilitate ongoing analysis performed by data scientists, the architecture calls for the data generated from Hive jobs to be inserted back into a postgresql database. For this study, we used Apache Sqoop to move the data between platforms.

Creating a Postgresql Database to Store Hive Data

There are a number of ways that one can utilize to create a database and the appropriate tables to store the data. One way to do so is by examining the details of the tables created in Hive and stored in the database, “metastore”, referenced in the previous section. The other option– which we used in this project for the sake of expediency– is to examine the Hive job to build the database and the tables. The SQL code to do so is available as a SQL script in the GIT repository (`to_postgres/create_tables.sql`). Let’s examine a few

key elements of the SQL script. The first section shown below,

```
--
-- First create the database that will store all the processed weblog data
--
-- It is presumed the user is starting the script logged on as user "postgres"
-- We will create the database and transfer the ownership to user "hiveuser"
--
CREATE DATABASE ebags_weblog
       WITH OWNER = hiveuser;
--
-- Create tables in preparation for importing data from HDFS into postgres
--
-- The tables are created in the database we just created
--
\c ebags_weblog
```

creates the database and changes the owner to the user “hiveuser” that we used earlier on. In addition, notice the profuse use of comments in the SQL script to explain the logic behind the commands.

The second element,

```
--
--  The weblog table
--
CREATE TABLE  weblog
 (
  utc_datetime TIMESTAMP,
  mt_datetime TIMESTAMP,
  utc_time_ms INT,
  s_ip VARCHAR,
  cs_method VARCHAR,
  s_sitename VARCHAR,
  cs_uri_stem VARCHAR,
  s_computername VARCHAR,
  cs_uri_query VARCHAR,
  cs_user_agent VARCHAR,
  s_port VARCHAR,
  cs_referer VARCHAR,
  cs_host VARCHAR,
  sc_status VARCHAR,
  sc_win32_status VARCHAR,
  sc_bytes BIGINT,
  cs_bytes BIGINT,
  time_taken_ms BIGINT,
  visitor_id_hash VARCHAR
 );

ALTER TABLE weblog OWNER TO hiveuser;
```

creates a table (`weblog`) with a prescribed schema (obtained from the Hive job). The ownership of the table is also changed to “hiveuser” to enable Sqoop to insert the data into the tables at a later time.

Installing Sqoop and Appropriate Tools

The AMI image provided in the class does not have Apache Sqoop. We utilize this tool to move the processed weblog data from the Hive job insto the “ebags_weblog” database created before. Apache Sqoop can be downloaded by cloning the Apache Github repository as shown below:

```
git clone https://git-wip-us.apache.org/repos/asf/sqoop.git
```

The command downloads the latest version of Sqoop, which in our case was version 1.5.0-SNAPSHOT. To built Sqoop binaries and documentation, one needs the Apache ANT, asciidoc and xmlto. The following procedure, if followed, will build a working Sqoop product on the UC Berkeley provided AMI image.

1. A prebuilt version of Apache ANT for JRE version 1.7.0 can be downloaded and installed in the home directory for user w205. For our project, the ANT version that was appropriate was 1.9.9.
2. The package asciidoc is readily available as a RPM package and can be installed using the command `yum install -y asciidoc`.
3. Similarly the package xmlto can be installed using the command `yum install -y xmlto`.
4. The PATH environment variable should be set to include the ANT binaries.
5. Follow the instructions provided in the COMPILING.txt file in the Sqoop directory to build Sqoop. The simplest way to do so is to execute `ant package`. This will build everything including the documentation.
6. Update the PATH variable to include the directory in which the sqoop binary resides. This was in our implementation `/home/w205/sqoop/bin`.

Executing Sqoop to Populate the Postgresql Database

Once all the tools and the data is in place as described above, `sqoop`, can be used to populate the processed weblog data from HDFS into Postgres. The code to do so is included in the aforementioned GIT repository for our project (`to_postgres/sqoop.sh`). The first sqoop command in that file is shown below:

```
#!/bin/bash
#
#  This script executes a number of sqoop commands to import data from HDFS into Post
gres database.
#
#  It assumes
#
#   1. The database, ebags_weblog, has been created a priori
#   2. ANT is installed on the UCB provided AMI
#   3. Packages, asciidoc and xmlto, has been installed on the image
#   4. Sqoop version 1.5.0 has been downloaded from Apache github repository and comp
iled.
#   5. Hive has been set up to use Postgres and the the weblog pipelining job has bee
n executed.
#
#   Import the weblog data from HDFS
#
sqoop export --input-null-string '\\N' --input-null-non-string '\\N' --connect jdbc:p
ostgresql://localhost/ebags_weblog \
       --username hiveuser --input-fields-terminated-by "\t" --export-dir=hdfs://loca
lhost:8020/user/hive/warehouse/weblog/ --table=weblog
```

It can be seen from the code that we are exporting information stored in files located in the HDFS directory, `/user/hive/warehouse/weblog` to the table `weblog` in the database `ebags_weblog`. Sqoop

performs additional mapreduce jobs to transform the data generated from Hive and populates the database. It should be noted that the password for accessing the database is not shown in the command line. It is provided to Sqoop through the use of PGPASSFILE environment variable. Users should refer to Postgres version 8.4 documentation to understand how to use this variable.

In summary, our architecture involves multiple tools in our data pipelining and presents data in different formats for various use cases. In our vision, as new weblogs become available, the user would automatically process the new log using the Hive procedure (discussed in other documents) and store it in HDFS database for anyone to utilize. They would also run the Sqoop job cited above to transform the data into a relational database and store it in that format for rapid access to structured information.