Partitioning in Apache Hive

CS34800 • Spring 2019 • Project 4

Due: Thursday, April 25, 2019 at 11:59PM on Blackboard
(10% penalty/late day. After five calendar days, the project will not be accepted.)

Overview

This project will provide an introduction to Apache Hive, a big data tool that makes it simple to query structured data. Hive is built on top of MapReduce, which is in turn built on top of HDFS. Hive accepts SQL-like queries using its own query language (HiveQL) and converts them into MapReduce jobs.

Read through this tutorial for an overview of Hive's architecture:
http://www.tutorialspoint.com/hive/hive_introduction.htm

Environment and Setup

We will use Purdue's OpenStack cluster for this assignment. The master node for this cluster is 172.18.11.30.

You can SSH into this node via data.cs.purdue.edu with username `<purdue-alias>_ostack` and password `<purdue-alias>_ostackpwd`.

```
$ ssh <purdue-alias>_ostack@172.18.11.30
```

To see a list of all the nodes in the cluster, run:

```
$ cat /etc/hosts
```

Be sure to change your password after you have logged in:

```
$ passwd
```

NOTE:

1. **Any unethical use of the cluster for purposes unrelated to your project work including unnecessarily hogging up resources will be grounds for point deduction.** This cluster is a shared resource, so be mindful of your classmates.
2. This cluster does not mount the CS department's NFS shared file system, so your CS home directory is not available.
3. This cluster is temporary. It will be wiped after the lab is graded. If you have any code or results that you wish to save, move them to permanent storage on another system.

Background

The dataset for this project is stock market data[1] obtained from Kaggle. It is distributed freely. This original data was compiled into csv format and then filtered out to an appropriate size for the purposes of this project.

The dataset is approximately 380 MB. Each row corresponds to a company's stock market data concerning a particular date. These include the respective company's opening stock price (for that date), closing stock price, highest stock price, lowest stock price, volume of company's shares traded, etc. For the purposes of this project, we are only concerned with the volume data.

We wish to query this data. An example query might be, "what is the average volume of shares traded of company *XYZ* on the first business day of every month?". When we query this data, we wish to do so efficiently. If we can split the data into partitions based on year, month, or day, then perhaps we will not have to read all of the data every time we run a query. Of course, we could accomplish these goals by distributing the data across our cluster with HDFS and then writing MapReduce jobs to partition and query our data. Instead, we will use Hive, which simplifies this process immensely.

Setup

The data was already downloaded and shared at `/home/data` (the file to be used is `stock_market_data.csv`).

Load the shared data into your personal HDFS directory:

```
$ hdfs dfs -mkdir -p /user/<purdue-alias>_ostack/project4/input
$ hdfs dfs -put /home/data/stock_market_data.csv /user/<purdue-
alias>_ostack/project4/input
```

Start the Hive CLI (command line interface), create a personal database, and use that database:

```
$ hive
hive> create database <purdue-alias>_ostack;
hive> use <purdue-alias>_ostack;
```

NOTE: When you restart the Hive CLI, you will start off in the default database. In that case, you must again switch to the database you previously created:

```
$ hive
hive> use <purdue-alias>_ostack;
```

We need to declare some structure for our data. We will use a command from Hive's data definition language:

```
hive> create table stock_market (company string, date string, open
float, high float, low float, close float, volume int, open_int int)
row format delimited fields terminated by ',';
```

Next, we need to import the data. Note that when we import an HDFS file into a Hive table, Hive does not copy the data. It simply changes the name of the file and moves it to another HDFS directory (a Hive directory).

```
hive> load data inpath '/user/<purdue-
alias>_ostack/project4/input/stock_market_data.csv' overwrite into
table stock_market;
```

We are now ready to query our data. The following queries might be interesting to you:

```
hive> show tables;
hive> describe stock_market;
hive> select * from stock_market limit 3;
hive> select count(*) from stock_market;
```

Notice that the `date` attribute is in `string` format which is not an optimal choice if we want to query our data for specific days, months, or years. Most real-world data typically needs to be preprocessed. We will create a new table and insert the same data as our original table but with day, month, and year extracted:

```
hive> create table stock_market_filtered_date (company string, year
int, month int, day int, open float, high float, low float, close
float, volume int, open_int int);
hive> insert into stock_market_filtered_date select company,
year(date), month(date), day(date), open, high, low, close, volume,
open_int from stock_market;
```

Tasks

TASK 1

Notice that when inserting data into the `stock_market_filtered_date` table, multiple mappers were spawned. **Why did this not happen when loading data into the `stock_market` table?**

TASK 2

Write and execute queries for the problem statements given below, and after each query completes, **report the queries you wrote, the returned totals, and the following performance metrics** which appear under *MapReduce Jobs Launched*:

- Cumulative CPU (for each stage)
- HDFS Read (for each stage)
- HDFS Write (for each stage)
- Time Taken (total)

1. The total volume of shares traded in the month of October.
2. The total volume of shares traded on March 31.
3. The total volume of shares traded between January 1, 2000 and January 15, 2000 (inclusive).

HINT: These queries are simple one-liners whose syntax is exactly the same as it would be in pure SQL.

As a beneficial exercise, you might try to sketch out how you think Hive should configure the map and reduce jobs to accomplish each of these queries. You can also type `explain` before any query to see some details about how Hive plans to process the query.

TASK 3

Next, we would like to make our queries more efficient by making a copy of our table and dividing the copy into partitions. Our table currently lives in HDFS, and once we create the partitions, those partitions will also live in HDFS.

First, we instruct Hive to automatically determine which partitions to create when given a partition column. Next, we increase the maximum number of partitions allowed. NOTE: Some of these values do not persist, so you should set them again if you restart your Hive shell.

```
hive> set hive.exec.dynamic.partition=true;
hive> set hive.exec.dynamic.partition.mode=nonstrict;
hive> set hive.exec.max.dynamic.partitions=900;
hive> set hive.exec.max.dynamic.partitions.pernode=900;
```

Next, we declare a new table with the same columns as `stock_market_filtered_data`, but we indicate to Hive that the data should be partitioned on the `month` column:

```
hive> create table stock_market_filtered_date_partitioned_month
(company string, year int, day int, open float, high float, low float,
close float, volume int, open_int int) partitioned by (month int);
```

Notice that we have omitted `month` from the long list of fields in our table. Instead, we have included it as a partition column at the end of our statement. This `month` field comes last if we `describe` this table, which is how Hive chooses to order partition columns.

Next, we will copy data from our `stock_market_filtered_data` table to our `stock_market_filtered_date_partitioned_month` table using the following command:

```
hive> insert into table stock_market_filtered_date_partitioned_month
partition (month) select company, year, day, open, high, low, close,
volume, open_int, month from stock_market_filtered_date;
```

Note that the ordering of columns in our `insert` statement matches the order of columns in `stock_market_filtered_date_partitioned_month`.

Run Task 2 queries #1 to #3 on your partitioned table and **report the counts and the performance metrics. What do you observe for the cumulative CPU time and for the wall clock time (*Time taken*) compared to queries on the unpartitioned data? Explain why.**

TASK 4

Create two more partitioned tables:

1. Partitioned on year, called `stock_market_filtered_date_partitioned_year`.
2. Partitioned on day, called `stock_market_filtered_date_partitioned_day`.

Re-run Task 2 queries #1 to #3 on these partitioned tables. **Report the queries you wrote for partitioning and data insertion, the returned totals, and the performance metrics**. **Which column partitioning (among `year`, `month`, or `day`) performed the best in general? Explain why.**

TASK 5

Now is a good time for you to begin considering how to identify good use cases for big data tools like Hive and MapReduce. How big does our data have to be before querying becomes faster on a cluster than on a single machine? Let us do a quick experiment.

Run the following command:

```
$ date +"%T"; cat /home/data/stock_market_data.csv | awk -F',' '$2 ==
"2000-01-03" {print $1}' | wc -l; date +"%T"
```

The above command will count how many company records exist for January 3, 2001 (or more specifically, how many companies in the dataset opened for trading that day). **Report the count and the runtime.**

Run a query each on the `stock_market` table and the `stock_market_filtered_date` table that accomplishes the same goal as the above command. **Report these two queries you wrote along with the count and the runtime.**

How does the runtime compare in both the cases, that is, with and without MapReduce? Explain your observations.

Deliverables

For your submission, please upload a text file named **`<purdue-alias>_project4.txt`** with all the commands, metrics, counts, runtimes, and answers requested in tasks 1 to 5. Additionally, the content of your database on the cluster will also be evaluated.

Debugging

- Enable debug Hive CLI logging if interested to know what is happening under the hood:

  ```
  hive -hiveconf hive.log.file=debug_hive.log -hiveconf
  hive.log.dir=/home/<purdue-alias>_ostack/ -hiveconf
  hive.root.logger=DEBUG,DRFA
  ```

  For just info level logs, replace `DEBUG` with `INFO` in `hive.root.logger` parameter value.

- If you created your database but are not able to `use` it and are getting an error stating that it does not exist, re-run the `create` query for your database. Then, `use` your database and re-run the `create` queries for the tables which existed. The corresponding data (if previously inserted) will be back up and you will not need to re-run any `insert` or `load` queries.

- If you created a table but are not able to query it and are getting an error stating that it does not exist, re-run the `create` query for your table. The corresponding data (if previously inserted) will be back up and you will not need to re-run any `insert` or `load` queries.

References

[1]https://www.kaggle.com/borismarjanovic/price-volume-data-for-all-us-stocks-etfs