Purdue University
Fall 2017
CS 348
Project 4: Partitioning in Apache Hive

Due Monday, November 20, 2017 at 11:59PM Submit via Blackboard. Every late day: 10% penalty up to 5 days, after which the assignment will not be accepted.

```
/**********
* Overview
**********/
```

This project will provide an introduction to 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 queries and converts them into MapReduce jobs.

Read through this page 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 172.18.11.17. You can SSH into this node with username:[PurdueAlias]_ostack, password:[PurdueAlias]_ostackpwd.

```
$ ssh [PurdueAlias]_ostack@172.18.11.17
```

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

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

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

```
$ passwd
```

NOTE: This cluster does not mount the CS department's NFS shared file system, so your CS home directory is not available.

CAUTION: This cluster is temporary. It will be wiped after the lab is graded. If you have any codes or results that you wish to save, move them to permanent storage on another system.

```
/**********
* Basic commands for HDFS
**********/
```

To list the contents of your HDFS directory, use:

```
$ hdfs dfs -ls /user/[PurdueAlias]_ostack
```

To move files to HDFS and back:

```
$ hdfs dfs -mkdir /user/[PurdueAlias]_ostack/new_dir
$ hdfs dfs -put file.txt /user/[PurdueAlias]_ostack/new_dir
$ hdfs dfs -get /user/[PurdueAlias]_ostack/new_dir/file.txt ./
$ hdfs dfs -getmerge
/user/[PurdueAlias]_ostack/dir_with_multiple_files
```

```
/**********
* Background
**********/
```

The Research and Innovative Technology Administration (RITA) has made available 22 years worth of flight departure and arrival data. The total dataset, when uncompressed, is approximately 10 GB.
http://stat-computing.org/dataexpo/2009/the-data.html

We wish to query this data. An example query might be, "How many flights departed on February 3, 1990?"

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 (file name is 1996_noheader.csv)
Load the shared data into your personal HDFS directory:

```
$ hdfs dfs -mkdir -p /user/[PurdueAlias]_ostack/rita/input
$ hdfs dfs -put /home/data/1996_noheader.csv
/user/[PurdueAlias]_ostack/rita/input
```

Start the Hive CLI, create a personal database, and use that database:

```
$ hive
hive> create database [PurdueAlias]_ostack;
hive> use [PurdueAlias]_ostack;
```

NOTE: If you restart the Hive CLI, you will begin in the default database. In that case you must again switch to your database with "hive> use [PurdueAlias]_ostack".

We need to declare some structure for our data. We will use a command from Hive's data definition language. Notice that the command specifies a comma as the field delimiter.

```
hive> create table flights(Year int, Month int, dayOfMonth int,
dayOfWeek int, depTime int, CRSDepTime int, arrTime int, CRSArrTime
int, uniqueCarrier string, flightNum int, tailNum int,
actualElapsedTime int, CRSElapsedTime int, airTime int, arrDelay int,
depDelay int, origin string, dest string, distance int, taxiIn int,
taxiOut int, cancelled int, cancellationCode string, diverted int,
carrierDelay int, weatherDelay int, NASDelay int, securityDelay int,
lateAircraftDelay int) row format delimited fields terminated by ',';
```

Next, we need to import the data into our table. 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/[PurdueAlias]_ostack/rita/input/1996_noheader.csv' overwrite
into table flights;
```

We are now ready to query our data. You can experiment if you like. The following queries might be interesting to you:

```
hive> show tables;
hive> describe flights;
hive> select * from flights limit 3;
hive> select count(*) from flights where month=3;
hive> select count(*) from flights where carrierdelay is null;

/**********
* Task
**********/

/* Task 1 */
```

Execute the queries below, and after each query completes, record 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)
```

```
// Query 1
hive> select count(*) from flights where month = 3;

// Query 2
hive> select count(*) from flights where month = 6 and dayofmonth =
12;

// Query 3
hive> select count(*) from flights where month = 8 and dayofmonth > 10
and dayofmonth < 20;
```

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 2 */
```

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=1000;
hive> set hive.exec.max.dynamic.partitions.pernode=1000;
```

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

```
hive> create table flights_partitioned_month(Year int, dayOfMonth int,
dayOfWeek int, depTime int, CRSDepTime int, arrTime int, CRSArrTime
int, uniqueCarrier string, flightNum int, tailNum int,
actualElapsedTime int, CRSElapsedTime int, airTime int, arrDelay int,
depDelay int, origin string, dest string, distance int, taxiIn int,
taxiOut int, cancelled int, cancellationCode string, diverted int,
carrierDelay int, weatherDelay int, NASDelay int, securityDelay int,
lateAircraftDelay 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.

After you have created the month partition table, describe it with:

```
hive> describe flights_partitioned_month;
```

Notice that the "Month" field comes last. This is how Hive chooses to order partition columns.

Next we will copy data from our "flights" table to our "flights_partitioned_month" table. Use this command:

```
hive> insert into table flights_partitioned_month partition(month)
select year, dayofmonth, dayofweek, deptime, crsdeptime, arrtime,
crsarrtime, uniquecarrier, flightnum, tailnum, actualelapsedtime,
crselapsedtime, airtime, arrdelay, depdelay, origin, dest, distance,
taxiin, taxiout, cancelled, cancellationcode, diverted, carrierdelay,
weatherdelay, nasdelay, securitydelay, lateaircraftdelay, month from
flights;
```

Here, the ordering of columns in our insert statement matches the order of columns in "flights_partitioned_month", not "flights".

Notice that when we begin our partition, Hive informs us that "Number of reduce tasks is set to 0 since there's no reduce operator." Why do we not need a reduce operator?

Run Task 1 queries #1-#3 on your partitioned table and record their performance metrics. What do you observe for cumulative CPU time compared to our queries on the unpartitioned data? What do you observe for the wall clock time ("Time taken")? Why do you think this is?

/* Task 3 */

We ask you to create two more partitioned tables: one partitioned on dayOfMonth and one partitioned on two columns: month first, and dayOfMonth second. Re-run queries #1-#3 on these partitioned tables and record their performance metrics.

/* Task 4 */

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's do a quick experiment.

Run the following command on the 1996 dataset that was downloaded to /home/data/ directory on the master node:

```
$ date +"%T"; cat /home/data/1996_noheader.csv | awk -F',' '$2 == "8"
{print $1}' | wc -l; date +"%T"
```

This command will count how many flights occurred in August of 1996. Run a query on your Hive table that accomplishes the same query. (For this comparison, don't use one of the shared Hive tables, and don't use a partitioned Hive table.) How does the runtime compare to our local job?

/**********
* Deliverables
**********/

Please turn in the following:
* A .txt file containing the commands that you used to create and populate the partitioned tables in Task 3;
* A .txt file containing the metrics which you recorded for Tasks 1-3;

* A .txt file containing the row counts obtained in Task 4 and the runtimes of the two approaches;

To turn your work in, submit the following files via Blackboard
commands.txt
metrics.txt
runtimes.txt