CMU MSP 36602 Apache Pig, Part 2

H. Seltman, Mar 5 2019

1) One good **workflow** is as follows:

   a) Use UNIX tools to obtain a reasonable subset of the data. For smaller files, something like `cat myInFile | sort -R | head -n 500 > myOutFile` is OK (using the -R flag to sort into random order), but that is very slow on large files. For large files, a Bash script that reads lines and writes them out if `[[ $RANDOM -LT $n ]]` works well by taking advantage of the fact that $RANDOM is a different number between 0 and 32767 each time it is referenced.

   b) Work in local mode as long as possible.

   c) Use `emacs myPigScript.pig&` to create your commands in a separate editor window.

   d) Use `pig –x local myPigScript.pig` to run your script.

   e) Write the code one line (relation) at a time. Examine the results for the last relation with `DUMP myRelation;` and/or `DESCRIBE myRelation;`.

   f) Finally, change `myRelation1 = LOAD 'myFile.ext' …;` to `myRelation1 = LOAD '$in' …;` and `DUMP myRelation;` to `STORE myRelation INTO '$out';` (and possibly add other parameters). Test with `pig -x local -param in='myInfile.ext' -param out=myOutDirectory myPigScript.pig`.

   g) Upload the full data to Hadoop and run `pig -param in=myInfile.ext` **`-param=myOutDirectory myPigScript.pig`**.

2) MultiStore Example

We are working for a large retailer with many stores, each of which has its sales information on the Hadoop file system. Each store makes purchases of various products at a particular price, which varies over time and by store. This information is stored in P#.dat where "#" is the store number. They also record each sale, including the purchase number (to link to the store's purchase price), the number of units bought, and the price paid.

Here we define profit margin as (price paid minus cost) / cost.

Input: purchases.zip, sales.zip
Goal: Compare total sales per store to average profit margin

Details of purchases.zip (one record per PO number)
Files named P#.dat where "#" is a store number.
Tab separated fields: store #
purchase order number (for one product and one lot/date)
product #
unit cost (cost of one item)

Details of sales.zip (one record per customer and product)

Files named S#.dat where "#" is a store number.

Tab separated fields: store # (link to purchase file)
purchase order number (link to purchase file)
price paid (per item)
count (# of items customer purchased)

Data setup:

In a Linux folder run:

```
hget data purchases.zip
hget data sales.zip
unzip purchases.zip
unzip sales.zip
```

Hints:

a) Once you have loaded some data, using
```
my10 = LIMIT myData 10;
DUMP myData;
```
can be a good way to get a quick look at the data (note that DESCRIBE shows the schema without data, and ILLUSTRATE only shows one line of data).

b) To join on two columns use:
```
myRelation3 = JOIN myRelation1 BY (myCol1, myCol2), myRelation1 BY
                  (myCol1, myCol2);
```

c) Note that the elements in a schema have the form "varName: varType". When there is ambiguity in a variable name, e.g., after a join, this changes to "originalRelation::varName: varType".

d) Remember that `GROUP myRelation BY myColumn` has two elements named "group" and "myRelation". Then myRelation.myOtherColumn references the several values of "myOtherColumn".

e) Use DESCRIBE a lot! Use DUMP (perhaps with LIMIT) as needed.

Tasks:

a) Load in data from all files as two relations and examine the first 15 records of each
b) Join the two relations, examine the first 5 records, and think about what the relation means.
c) Simplify the relation to remove duplicate elements ending up with storeNum, productNum, cost, count, price, and "margin" (price-cost)/cost.
d) (Using Linux utility programs, find out how many different products there are.)
e) Make a rank ordering of total items purchased across all stores for each item
f) Make a rank ordering of revenue per store and store into storeRevenue.csv.
g) Make a rank ordering of the number of purchase orders per store
h) Prepare data for an R regression of revenue ~ number of POs (per store)