Programming Assignment: Joining Data

You have not submitted. You must earn 80/100 points to pass.

Deadline Pass this assignment by March 5, 11:59 PM PST

Instructions

My submission

Discussions

Introduction to Map/Reduce Module, Programming Assignment, Lesson 2

Exercise in Joining data with streaming using Python code

In Lesson 2 of the Introduction to Map/Reduce module the Join task was described. In this assignment, you are given a Python mapper and reducer to perform the Join described in the 3rd video of Lesson 2. The purpose of Part 1 of this assignment is to provide an example for Part 2. In Part 2 you will need to modify the provided Python code (or write your own from scratch) to perform a different Join and upload the output file(s).

Please read through all the instruction and the programming notes, especially if you are not a programmer. It is not a hard programming assignment, but is worth the effort to understand the nature of map/reduce framework.

PART 1

1. Follow the steps from the Wordcount assignment to set up the following files on Cloudera: join1_mapper.py, join1_reducer.py, join1_FileA.txt, and join1_FileB.txt (see the *Code and Text Files* section at the bottom)

Don't forget to enter the following at the unix prompt to make it executable

```
> chmod +x join1_mapper.py
> chmod +x join1_reducer.py
```

2. Follow the steps from the Wordcount assignment to set up the data in HDFS

3. Test the program in serial execution using the following Unix utilities and piping commands:

('cat' prints out the text files standard output; '|' pipes the standard output to the standard input of the join_mapper program, etc..)

```
> cat join1_File*.txt | ./join1_mapper.py | sort | ./join1_reducer.py
```

To debug programs in serial execution one should use small datasets and possibly extra print statements in the program. Debugging with map/reduce jobs is harder but hopefully not necessary for this assignment. There are more descriptions of how to debug your code in the reading notes for the lesson.

4. Run the Hadoop streaming command:

(Note that your file paths may differ. Note the '\' just means the command continues on next line.)

```
> hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar \
   -input /user/cloudera/input \
   -output /user/cloudera/output_join \
   -mapper /home/cloudera/join1_mapper.py \
   -reducer /home/cloudera/join1_reducer.py

```

PART 2: A new join problem

1. First generate some datasets using the scripts (see the *Code and Text Files* section at the bottom) as follows:

```
> sh make_data_join2.txt
```

(this is a script that produces 6 files:

python make_join2data.py y 1000 13 > join2_gennumA.txt

python make_join2data.py y 2000 17 > join2_gennumB.txt

...)

2. Use HDFS commands to copy the 6 files created in step 1 into one HDFS directory, just like step 2 in Part 1 and in the wordcount assignment.

Note: These datasets are pseudo-randomly generated so the output is the same for any environment. The files are not large, however they are big enough that it would be time consuming to solving the assignment by hand. One could put the data in a database but that would defeat the purpose of the assignment!

3. The datasets generated in step 1 contain the following information:

join2_gennum*.txt consist of <TV show, count> (A TV show title and the number of viewers)

Example join2_gennum*.txt:

```
Almost_News, 25
Hourly_Show,30
Hot_Cooking,7
Almost_News, 35
Postmodern_Family,8
Baked_News,15
Dumb_Games,60
…
```

join2_genchan*.txt consists of <TV show title, channel> (A TV show title and the channel it was on)

Example join2_genchan*.txt:

```
Almost_News, ABC
Hourly_Show, COM
Hot_Cooking, FNT
Postmodern_Family, NBC
Baked_News, FNT
Dumb_Games, ABC
…
```

4. Your Task: Implement the following join request in Map/Reduce:

What is the total number of viewers for shows on ABC?

The show-to-channel relationship is Many-to-Many. In other words, each show might appear on many channels, and each channel might broadcast many shows.

In pseudo-SQL it might be something like:

select sum(viewer count) from File A, File B where FileA.TV show = FileB.TV show and FileB.Channel='ABC' grouped by TV show

5. Upload the resulting output from the reducers, use numReduceTasks=1

The output will look like: <TVshow_title total_viewers>

Example Output:

```
Almost_News 60
Dumb_Games 60
…
```

Data Notes:

- TV show titles do not have spaces
- Channels have 3 letters
- TV show titles can appear multiple times, with different counts
- A TV show and channel combination might appear multiple times
- TV shows could appear on multiple channels
- The output should have no commas or punctuation, only 1 space between the TV show title and number

Programming Notes and Detailed Suggestions:

You should be able to use 1 map/reduce job. If you are not a programmer then you should review the join1 code and the wordcount code. You should consider starting with the join1_mapper and the join1_reducer code. I believe all the logic and functional examples you will need are in the wordcount and the join1 code.

Hint 1: The new join mapper is like the join1 mapper but instead of stripping dates from the key field it should be selecting rows related to 'ABC'.

In Python strings are character arrays. Strings can be declared and accessed as follows:

```
my_string = 'LMNOP'
my_string[0:3]='LMN'
```

In Python the string function to check if a string is just digits is as follows:

```
my_string.isdigit() # will return True or False.
```

Hint 2: The new join reducer is like the join1_reducer but instead of building up a list of dates & counts, it should be summing viewer counts to keep a running total. Make sure you understand what will be in the intermediate output files that become reducer input (after the mapper and after Hadoop shuffle & group). Look carefully at the groups that are present in the reducer input to see what conditions your reducer will have to check for. You can test the mapper output using the Unix piping mentioned in Part 1.

Hint 3: This new join task has some overlap with wordcounting task. Use the wordcount code to review the counting and updating logic, as well as functions that handle strings and integers.

Hint 4: Here is a possible pseudo code example of how to implement this join using tv-show as the key:

join2_mapper:

read lines, and split lines into key & value

if value is ABC or if value is a digit print it out

Note: you can test just the mapper by running something like:

```
> cat join2_gen*.txt | ./join2_mapper.py | sort
```

Also, if we did have huge files partitioned across a cluster you might have information about viewer counts in one partition and information about which show is on ABC in another partition. The mapper and the Hadoop shuffle will bring those bits of information together to the same reducer if a key is a show.

join2_reducer:

read lines and split lines into key & value

if a key has changed (and it's not the first input)

then check if ABC had been found and print out key and running total,

if value is ABC then set some variable to mark that ABC was found (like abc_found = True)

otherwise keep a running total of viewer counts

Hint 5:

The first two lines of your output should be:

```
Almost_Games 49237
Almost_News 46592
```

Code and Text Files

join1_mapper.py

```
#!/usr/bin/env python
import sys

# ------------------------------------------------------------------------
#This mapper code will input a <date word, value> input file, and move date into
#  the value field for output
#
#  Note, this program is written in a simple style and does not full advantage of Python
#     data structures,but I believe it is more readable
#
#  Note, there is NO error checking of the input, it is assumed to be correct
#     meaning no extra spaces, missing inputs or counts,etc..
#
# See #  see https://docs.python.org/2/tutorial/index.html for details  and python  tutorials
#
# ------------------------------------------------------------------------



for line in sys.stdin:
    line       = line.strip()   #strip out carriage return
    key_value  = line.split(",")   #split line, into key and value, returns a list
    key_in     = key_value[0].split(" ")   #key is first item in list
    value_in   = key_value[1]   #value is 2nd item

    #print key_in
    if len(key_in)>=2:           #if this entry has <date word> in key
        date = key_in[0]      #now get date from key field
        word = key_in[1]
        value_out = date+" "+value_in     #concatenate date, blank, and value_in
        print( '%s\t%s' % (word, value_out) )  #print a string, tab, and string
    else:   #key is only <word> so just pass it through
        print( '%s\t%s' % (key_in[0], value_in) )  #print a string tab and string

#Note that Hadoop expects a tab to separate key value
#but this program assumes the input file has a ',' separating key value

```

join1_reducer.py

```
#!/usr/bin/env python
import sys

# ---------------------------------------------------------------------------
#This reducer code will input a <word, value> input file, and join words together
# Note the input will come as a group of lines with same word (ie the key)
# As it reads words it will hold on to the value field
#
# It will keep track of current word and previous word, if word changes
#   then it will perform the 'join' on the set of held values by merely printing out
#   the word and values.  In other words, there is no need to explicitly match keys b/c
#   Hadoop has already put them sequentially in the input
#
# At the end it will perform the last join
#
#
#  Note, there is NO error checking of the input, it is assumed to be correct, meaning
#   it has word with correct and matching entries, no extra spaces, etc.
#
#  see https://docs.python.org/2/tutorial/index.html for python tutorials
#
#  San Diego Supercomputer Center copyright
# ---------------------------------------------------------------------------

prev_word          = "  "                #initialize previous word  to blank string
months             = ['Jan','Feb','Mar','Apr','Jun','Jul','Aug','Sep','Nov','Dec']

dates_to_output    = [] #an empty list to hold dates for a given word
day_cnts_to_output = [] #an empty list of day counts for a given word
# see https://docs.python.org/2/tutorial/datastructures.html for list details

line_cnt           = 0  #count input lines

for line in sys.stdin:
    line       = line.strip()       #strip out carriage return
    key_value  = line.split('\t')   #split line, into key and value, returns a list
    line_cnt   = line_cnt+1

    #note: for simple debugging use print statements, ie:
    curr_word  = key_value[0]         #key is first item in list, indexed by 0
    value_in   = key_value[1]         #value is 2nd item

    #----------------------------------------------------
    # Check if its a new word and not the first line
    #   (b/c for the first line the previous word is not applicable)
    #   if so then print out list of dates and counts
    #----------------------------------------------------
    if curr_word != prev_word:

        # -----------------------
  #now write out the join result, but not for the first line input
        # -----------------------
        if line_cnt>1:
      for i in range(len(dates_to_output)):  #loop thru dates, indexes start at 0
           print('{0} {1} {2} {3}'.format(dates_to_output[i],prev_word,day_cnts_to_output[i],curr_word_total_cnt))
            #now reset lists
      dates_to_output   =[]
            day_cnts_to_output=[]
        prev_word         =curr_word  #set up previous word for the next set of input lines


    # ---------------------------------------------------------------
    #whether or not the join result was written out,
    #   now process the curr word

    #determine if its from file <word, total-count> or < word, date day-count>
    # and build up list of dates, day counts, and the 1 total count
    # ---------------------------------------------------------------
    if (value_in[0:3] in months):

        date_day =value_in.split() #split the value field into a date and day-cnt

        #add date to lists of the value fields we are building
        dates_to_output.append(date_day[0])
        day_cnts_to_output.append(date_day[1])
    else:
        curr_word_total_cnt = value_in  #if the value field was just the total count then its
                                           #the first (and only) item in this list

# ---------------------------------------------------------------
#now write out the LAST join result
# ---------------------------------------------------------------
for i in range(len(dates_to_output)):  #loop thru dates, indexes start at 0
         print('{0} {1} {2} {3}'.format(dates_to_output[i],prev_word,day_cnts_to_output[i],curr_word_total_cnt))

```

join1_FileA.txt

```
able,991
about,11
burger,15
actor,22
```

join1_FileB.txt

```
Jan-01 able,5
Feb-02 about,3
Mar-03 about,8
Apr-04 able,13
Feb-22 actor,3
Feb-23 burger,5
Mar-08 burger,2
Dec-15 able,100

```

make_join2data.py

```
#!/usr/bin/env python
import sys

# ----------------------------------------------------------------------------
#  (make_join2data.py) Generate a random combination of titles and viewer
    counts, or channels
# this is a simple version of a congruential generator,
#   not a great random generator but enough
# ----------------------------------------------------------------------------

chans   = ['ABC','DEF','CNO','NOX','YES','CAB','BAT','MAN','ZOO','XYZ'
    ,'BOB']
sh1 =['Hot','Almost','Hourly','PostModern','Baked','Dumb','Cold','Surreal'
    ,'Loud']
sh2 =['News','Show','Cooking','Sports','Games','Talking','Talking']
vwr =range(17,1053)

chvnm=sys.argv[1]  #get number argument, if its n, do numbers not channels,

lch=len(chans)
lsh1=len(sh1)
lsh2=len(sh2)
lvwr=len(vwr)
ci=1
s1=2
s2=3
vwi=4
ri=int(sys.argv[3])
for i in range(0,int(sys.argv[2])):  #arg 2 is the number of lines to output

    if chvnm=='n':  #no numuber
        print('{0}_{1},{2}'.format(sh1[s1],sh2[s2],chans[ci]))
    else:
        print('{0}_{1},{2}'.format(sh1[s1],sh2[s2],vwr[vwi]))
    ci=(5*ci+ri) % lch
    s1=(4*s1+ri) % lsh1
    s2=(3*s1+ri+i) % lsh2
    vwi=(2*vwi+ri+i) % lvwr

    if (vwi==4): vwi=5

```

make_data_join2.txt (a short command line script)

```
python make_join2data.py y 1000 13 > join2_gennumA.txt
python make_join2data.py y 2000 17 > join2_gennumB.txt
python make_join2data.py y 3000 19 > join2_gennumC.txt
python make_join2data.py n 100  23 > join2_genchanA.txt
python make_join2data.py n 200  19 > join2_genchanB.txt
python make_join2data.py n 300  37 > join2_genchanC.txt


```

How to submit

When you're ready to submit, you can upload files for each part of the assignment on the "My submission" tab.