**Cheng, Loi**

9/28/2019

Harvard University Extension - Principles of Big Data Processing e88

Homework 4: Hadoop MapReduce

* **Make sure to submit all your source code (.java files , .py files or whatever language you are using) - in a separate archive, named <LastName>\_<FirstName>\_HW4.zip**
* **Make sure to add full result files into that archive as well**

Please identify which problems were completed. If any were incomplete, please identify where you encountered problems.

|  |
| --- |
| *for example:*  Problem 1: 100% complete  Problem 2: 100% complete  Problem 3: 100% complete  Problem 4: 100% complete  Problem 5: Bonus: 100% complete |

**Problem 1: [15 points] running example wordcount job with S3 data**

Include a screenshot of your S3 bucket with the 4 input log files [5 points]

|  |
| --- |
|  |

Show command you used to run the example wordcount job with your S3 data, and the job output [5 points]

|  |
| --- |
|  |

Show the number of splits, map and reduce tasks for this job - from the job console output or log files [5 points]

|  |
| --- |
|  |

**Problem 2: unique counts** [points: 40]

Paste your source code of the MR jobs into the following area [10 points]

|  |
| --- |
| #!/usr/bin/env python  '''  mapper  Query1: get count of unique URLs per hour  '''  #Loi Cheng  #9/xx/2019  import sys  # input must come from STDIN (standard input)  for line in sys.stdin:      columns = line.split(',',4)      dateHour = columns[1].split(':',1)[0].replace('T',':')      url = columns[2]      # output key and value to console, must be tab delimited, must be strings      print( '%s\t%s' % (dateHour+'::'+url, 1)  )  #!/usr/bin/env python  '''  reducer  Query1: get count of unique URLs per hour  reference  https://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/  '''  #Loi Cheng  #9/xx/2019  import sys  # input must come from STDIN (standard input)  # expected input is this: dateHour::url <tab> 1  # the keys comes presorted, so we can use the wordcount logic  # read first line  oldKey = sys.stdin.readline().split('\t', 1)[0]  count = 1  newKey = None  for line in sys.stdin:      # parse the input we got from mapper      newKey = line.split('\t', 1)[0]      # do something only if new and old key are different      if newKey != oldKey:          # check if dateHour has changed          newKeyDateHour = newKey.split('::')[0]          oldKeyDateHour = oldKey.split('::')[0]          # it's a new dateHour          if newKeyDateHour != oldKeyDateHour:              # write result to STDOUT, in form of dateHour <tab> uniqueUrlCount              print( '%s\t%s' % (oldKeyDateHour, count) )              # set up variables based on newKey              oldKey = newKey              count = 1            # it's a new unqiue url          else:              count += 1              oldKey = newKey  # output the last dateHour urlCount pair  if newKey == oldKey:      print( '%s\t%s' % (oldKey.split('::')[0], count) )  #!/usr/bin/env python  '''  mapper  Query2: get count of unique visitors per URL per hour  '''  #Loi Cheng  #9/xx/2019  import sys  # input must come from STDIN (standard input)  for line in sys.stdin:      columns = line.split(',')      dateHour = columns[1].split(':',1)[0].replace('T',':')      url = columns[2]      user = columns[3]      # output key and value to console, must be tab delimited, must be strings      print( '%s\t%s' % (dateHour+'::'+url+'::'+user, 1)  )  #!/usr/bin/env python  '''  reducer  Query2: get count of unique visitors per URL per hour  '''  #Loi Cheng  #9/xx/2019  import sys  # input must come from STDIN (standard input) and is the output from the mapper  # print( '%s\t%s' % (dateHour+'::'+url+'::'+user, 1)  )  # the keys comes presorted, so we can use the wordcount logic  # read first line  oldKey = sys.stdin.readline().split('\t', 1)[0]  count = 1  newKey = None  for line in sys.stdin:      # parse the input we got from mapper      newKey = line.split('\t', 1)[0]      # do something only if new and old key are different      if newKey != oldKey:          # check if dateHourUrl has changed          newKeyDateHourUrl = ":".join(newKey.split("::", 2)[:2])          oldKeyDateHourUrl = ":".join(oldKey.split("::", 2)[:2])          # it's a new dateHourUrl          if newKeyDateHourUrl != oldKeyDateHourUrl:              # write result to STDOUT              print( '%s\t%s' % (oldKeyDateHourUrl, count) )              # set up variables based on newKey              oldKey = newKey              count = 1          # it's a new unqiue user          else:              count += 1              oldKey = newKey  # output the last dateHour urlCount pair  if newKey == oldKey:      print( '%s\t%s' % (  ":".join(oldKey.split("::", 2)[:2])  , count) )  #!/usr/bin/env python  '''  mapper  Query3: get count of unique (by UUID) events/clicks per URL per hour (one unique click == one log line in our test log files, since we assume they all have unique UUIDs)  '''  #Loi Cheng  #9/xx/2019  import sys  # input must come from STDIN (standard input)  for line in sys.stdin:      columns = line.split(',')      dateHour = columns[1].split(':',1)[0].replace('T',':')      url = columns[2]      # output key and value to console, must be tab delimited, must be strings      print( '%s\t%s' % (dateHour+':'+url, 1)  )  #!/usr/bin/env python  '''  reducer  Query3: get count of unique (by UUID) events/clicks per URL per hour (one unique click == one log line in our test log files, since we assume they all have unique UUIDs)  '''  #Loi Cheng  #9/xx/2019  import sys  # input must come from STDIN (standard input) and is the output from the mapper  # print( '%s\t%s' % (dateHour+'::'+url, 1)  )  # the keys comes presorted, so we can use the wordcount logic  # read first line  oldKey = sys.stdin.readline().split('\t', 1)[0]  count = 1  newKey = None  for line in sys.stdin:      # parse the input we got from mapper      newKey = line.split('\t', 1)[0]      # compare newKey and oldKey and process accordingly      if newKey == oldKey:          count += 1      else:          print( '%s\t%s' % (  oldKey  , count) )          oldKey = newKey          count = 1  # output the last dateHour urlCount pair  if newKey == oldKey:      print( '%s\t%s' % (  oldKey  , count) ) |

Show commands you used to run your jobs in the AWS EMR; include screenshots of the console output and the summary of each job (see example) [10 points]

|  |
| --- |
| Example excerpt of a MR job console output:  >>>  Map-Reduce Framework  Map input records=975000  Map output records=975000  Map output bytes=15600000  Map output materialized bytes=484  Input split bytes=640  Combine input records=975000  Combine output records=20  Reduce input groups=5  Reduce shuffle bytes=484  Reduce input records=20  Reduce output records=5  Spilled Records=40  Shuffled Maps =12  Failed Shuffles=0  Merged Map outputs=12  GC time elapsed (ms)=1124  CPU time spent (ms)=30280  Physical memory (bytes) snapshot=2970128384  Virtual memory (bytes) snapshot=27007504384  Total committed heap usage (bytes)=2598895616  Shuffle Errors ….  Query 1  # query 1  hadoop jar /usr/lib/hadoop/hadoop-streaming.jar -file ./hw4\_problem2q1\_mapper.py -mapper ./hw4\_problem2q1\_mapper.py -file ./hw4\_problem2q1\_reducer.py -reducer ./hw4\_problem2q1\_reducer.py -input s3://csci.e-88.principles.of.big.data.processing/input\_files/ -output s3://csci.e-88.principles.of.big.data.processing/output\_query1      Query 2  # query 2  hadoop jar /usr/lib/hadoop/hadoop-streaming.jar -file ./hw4\_problem2q2\_mapper.py -mapper ./hw4\_problem2q2\_mapper.py -file ./hw4\_problem2q2\_reducer.py -reducer ./hw4\_problem2q2\_reducer.py -input s3://csci.e-88.principles.of.big.data.processing/input\_files/ -output s3://csci.e-88.principles.of.big.data.processing/output\_query2      Query 3  # query 3  hadoop jar /usr/lib/hadoop/hadoop-streaming.jar -file ./hw4\_problem2q3\_mapper.py -mapper ./hw4\_problem2q3\_mapper.py -file ./hw4\_problem2q3\_reducer.py -reducer ./hw4\_problem2q3\_reducer.py -input s3://csci.e-88.principles.of.big.data.processing/input\_files/ -output s3://csci.e-88.principles.of.big.data.processing/output\_query3 |

Show how many part-r-xxxxx result files you've got for each job - explain why the specific number? [5 points]

|  |
| --- |
| Each job returned 3 part-r-xxxxx results files, so 3 reducers were used. The EMR cluster has 3 total machines, so it appears that the default EMR settings sets the number of reducers to be the same as the number of machines. This was probably done so that the reducer jobs can be done in parallel by each machine and minimize runtime, assuming each reducer job takes about the same time to complete. |

Show results of your jobs for the specified keys - from the relevant result files [15 points]

|  |
| --- |
| Query 1:  <date\_hour>, <url\_count>  2019-09-12:13, 185 2019-09-12:14, 186 2019-09-12:15, 185 2019-09-12:16, 190 2019-09-12:17, 189  Query 2  <date:hour:url>, unique\_user\_count  2019-09-12:02:http://example.com/?url=003, 1 2019-09-12:02:http://example.com/?url=004, 3 2019-09-12:02:http://example.com/?url=005, 4 2019-09-12:02:http://example.com/?url=006, 10  Query 3  <date:hour:url>, event\_count  2019-09-12:02:http://example.com/?url=003, 1 2019-09-12:02:http://example.com/?url=004, 3 2019-09-12:02:http://example.com/?url=005, 5 2019-09-12:02:http://example.com/?url=006, 10 |

**Problem 3: time range queries** [points: 30]

Paste your source code into the following area [15 points]

|  |
| --- |
| #!/usr/bin/env python  '''  mapper  Query4  get count of unique URLs by country by hour for a  specified time range [t1-t2]  where t1 = 09/13/2019 5PM UTC and  t2 = 09/14/2019 9AM UTC  Both range ends are inclusive  '''  #Loi Cheng  #9/xx/2019  import sys  from datetime import datetime  # input must come from STDIN (standard input)  for line in sys.stdin:      columns = line.split(',')      dateHour = columns[1].split(':',1)[0].replace('T',':')      # 2019-09-14:23:55:14.880Z      country = columns[4]      url = columns[2]      t1 = datetime.strptime( '2019-09-13:17:00:00' , '%Y-%m-%d:%H:%M:%S')      t2 = datetime.strptime( '2019-09-14:09:00:00' , '%Y-%m-%d:%H:%M:%S')      dateHourDateTime = datetime.strptime( dateHour, '%Y-%m-%d:%H' )      if ( dateHourDateTime >= t1 and dateHourDateTime <= t2 ):          # output key and value to console, must be tab delimited, must be strings          print( '%s\t%s' % (dateHour+'::'+country+'::'+url, 1)  )  #!/usr/bin/env python  '''  reducer  Query4  get count of unique URLs by country by hour for a  specified time range [t1-t2]  where t1 = 09/13/2019 5PM UTC and  t2 = 09/14/2019 9AM UTC  Both range ends are inclusive  '''  #Loi Cheng  #9/xx/2019  import sys  # input must come from STDIN (standard input) and is the output from the mapper  # print( '%s\t%s' % (dateHour+'::'+country+'::'+url, 1)  )  # the keys comes presorted, so we can use the wordcount logic  # read first line  oldKey = sys.stdin.readline().split('\t', 1)[0]  count = 1  newKey = None  for line in sys.stdin:      # parse the input we got from mapper      newKey = line.split('\t', 1)[0]      # do something only if new and old key are different      if newKey != oldKey:          # check if dateHourCountry has changed          newKeyDateHourCountry = ",".join(newKey.split("::", 2)[:2])          oldKeyDateHourCountry = ",".join(oldKey.split("::", 2)[:2])          # it's a new DateHourCountry          if newKeyDateHourCountry != oldKeyDateHourCountry:              # write result to STDOUT              print( '%s\t%s' % (oldKeyDateHourCountry, count) )              # set up variables based on newKey              oldKey = newKey              count = 1          # it's a new unique url          else:              count += 1              oldKey = newKey  # output the last pair  if newKey == oldKey:      print( '%s\t%s' % (  "".join(oldKey.split("::", 2)[:2])  , count) ) |

Include screenshots of the console output and the job summary executed in the AWS EMR [10 points]

|  |
| --- |
| # query 4  hadoop jar /usr/lib/hadoop/hadoop-streaming.jar -file ./hw4\_problem2q4\_mapper.py -mapper ./hw4\_problem2q4\_mapper.py -file ./hw4\_problem2q4\_reducer.py -reducer ./hw4\_problem2q4\_reducer.py -input s3://csci.e-88.principles.of.big.data.processing/input\_files/ -output s3://csci.e-88.principles.of.big.data.processing/output\_query4 |

Show results of your query for the specified keys [5 points]

|  |
| --- |
| <date,hour,country>, url\_count  2019-09-13:19,IQ, 1 2019-09-13:19,IR, 4 2019-09-13:19,IS, 9 2019-09-13:19,IT, 2 2019-09-13:19,JE, 4 |

**Problem 4: [15 points] MR Jobs configurations and UI**

show command you run to execute your jobs with different reduce job settings - and console output of each job [5 points]

|  |
| --- |
| 2 reducers  # query 1 -D mapreduce.job.reduces=2  hadoop jar /usr/lib/hadoop/hadoop-streaming.jar -D mapreduce.job.reduces=2 -file ./hw4\_problem2q1\_mapper.py -mapper ./hw4\_problem2q1\_mapper.py -file ./hw4\_problem2q1\_reducer.py -reducer ./hw4\_problem2q1\_reducer.py -input s3://csci.e-88.principles.of.big.data.processing/input\_files/ -output s3://csci.e-88.principles.of.big.data.processing/output\_query1\_r2      4 Reducers  # query 1 -D mapreduce.job.reduces=4  hadoop jar /usr/lib/hadoop/hadoop-streaming.jar -D mapreduce.job.reduces=4 -file ./hw4\_problem2q1\_mapper.py -mapper ./hw4\_problem2q1\_mapper.py -file ./hw4\_problem2q1\_reducer.py -reducer ./hw4\_problem2q1\_reducer.py -input s3://csci.e-88.principles.of.big.data.processing/input\_files/ -output s3://csci.e-88.principles.of.big.data.processing/output\_query1\_r4 |

compare results of each job - explain the difference [5 points]

|  |
| --- |
| With 2 reducers, only 2 results files are produced. On a 3 machine EMR cluster, only 2 of the machines were probably utilized during the reducer phase, and 1 of the machines was idle.  With 4 reducers, 4 result files are produced. On a 3 machine EMR cluster, 1 machine did 2 of the reducer jobs while the other 2 machines did 1 job each. |

include screenshots of the EMR ResourceManager UI with each job' statistics - explain differences if any [5 points]

|  |
| --- |
| The 2 reducers job appears to used less CPU, less memory, and completed slightly faster than the 4 reducers job. The results suggest that the number of reducers should be set no more than the number of machines in the cluster.  2 reducers    4 reducers |

**Problem 5: Bonus: Top N queries** [15 points]

Paste your source code into the following area [5 points]

|  |
| --- |
| #!/usr/bin/env python  '''  mapper  Query5  get top 5 URLs by avg daily TTFB, per each day in the input data set; consider "top" URLs to be those that have the smallest avg TTFB  '''  #Loi Cheng  #9/xx/2019  import sys  from datetime import datetime  # input must come from STDIN (standard input)  for line in sys.stdin:      columns = line.split(',')      date = columns[1].split('T',1)[0]      url = columns[2]      ttfb = str(columns[8]).rstrip()  #floats leave a newline and must be removed      # output key and value to console, must be tab delimited, must be strings      print( '%s\t%s' % (date, url+"::"+ttfb)  )  #!/usr/bin/env python  '''  reducer  Query5  get top 5 URLs by avg daily TTFB, per each day in the input data set; consider "top" URLs to be those that have the smallest avg TTFB  '''  #Loi Cheng  #9/xx/2019  import sys  def getTopFiveTtfb(date, urlDict):      # calculate average      avgTtfbDict = {}      for url in urlDict:          avgTtfbDict[url] = urlDict[url][0] / float( urlDict[url][1] )      for key, value in sorted(avgTtfbDict.items(), key=lambda item: item[1])[:5]:          print("%s: %s" % (date+'::'+key, value))  # input must come from STDIN (standard input) and is the output from the mapper  # print( '%s\t%s' % (date, url+"::"+ttfb)  )  # process first line  firstLine = sys.stdin.readline().rstrip().split('\t', 1)  urlTtfb = firstLine[1].split('::')  url = urlTtfb[0]  ttfb = float( urlTtfb[1] )  #make dict in format >> { url : [ttfb, ttbcount] }  urlDict = { url : [ttfb, 1] }  oldDate = firstLine[0]  newDate = None  for line in sys.stdin:      # parse the input we got from mapper      line = line.rstrip().split('\t', 1)      newDate = line[0]      urlTtfb = line[1].split('::')      url = urlTtfb[0]      ttfb = float( urlTtfb[1] )      # still on same date      if newDate == oldDate:          # if newUrl already exists          if url in urlDict.keys():              storedTtfb = urlDict[url][0]              storedCount = urlDict[url][1]              urlDict[url] = [ storedTtfb + ttfb , storedCount + 1]          # if newUrl is brand new          else:              urlDict[url] = [ttfb, 1]        # received new date      else:          #write out results from oldDate          getTopFiveTtfb(oldDate, urlDict)          #reset everything          urlDict = {}          urlDict[url] = [ttfb, 1]          oldDate = newDate  #write out last date  if newDate == oldDate:      getTopFiveTtfb(oldDate, urlDict) |

What are the main differences with the Problem 3 and 4 implementation? [5 points]

|  |
| --- |
| With the problem 3 and problem 4 implementation, the data can be split into many more different keys, such as by the hour, or by the hour and url. With more keys, the data can be distributed to more reducers and enable more parallel processing. With query 5, the data was only split by dates as keys, in order to rank the top average TTFB’s for each date. Since there are only 4 dates in the data set, there are only 4 keys and so the data can only be distributed to at most 4 reducers. |

Show results of your query [5 points]

|  |
| --- |
| # query 5  hadoop jar /usr/lib/hadoop/hadoop-streaming.jar -file ./hw4\_problem2q5\_mapper.py -mapper ./hw4\_problem2q5\_mapper.py -file ./hw4\_problem2q5\_reducer.py -reducer ./hw4\_problem2q5\_reducer.py -input s3://csci.e-88.principles.of.big.data.processing/input\_files/ -output s3://csci.e-88.principles.of.big.data.processing/output\_query5    2019-09-12::http://example.com/?url=114: 0.39310140845070424  2019-09-12::http://example.com/?url=101: 0.40254500000000004  2019-09-12::http://example.com/?url=133: 0.4133171875000001  2019-09-12::http://example.com/?url=033: 0.4188678571428572  2019-09-12::http://example.com/?url=157: 0.4192893939393939  2019-09-13::http://example.com/?url=039: 0.4028142857142858  2019-09-13::http://example.com/?url=110: 0.42946935483870974  2019-09-13::http://example.com/?url=005: 0.4306358490566037  2019-09-13::http://example.com/?url=006: 0.43138115942028993  2019-09-13::http://example.com/?url=028: 0.43190645161290325  2019-09-14::http://example.com/?url=123: 0.40559807692307703  2019-09-14::http://example.com/?url=070: 0.4185893333333335  2019-09-14::http://example.com/?url=124: 0.4233158730158729  2019-09-14::http://example.com/?url=107: 0.42657058823529403  2019-09-14::http://example.com/?url=101: 0.42943692307692305  2019-09-15::http://example.com/?url=185: 0.2863  2019-09-15::http://example.com/?url=175: 0.6621  2019-09-15::http://example.com/?url=167: 0.7849  2019-09-15::http://example.com/?url=008: 0.8516 |