**Cheng, Loi**

October 13, 2019

Harvard University Extension - Principles of Big Data Processing e88

Homework 6: **Batch Views with HBase and NoSQL**

* **Make sure you submit your solution document as a separate file in Canvas**
* **submit all your source code in a separate archive, named <LastName>\_<FirstName>\_HW6.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: AWS EMR with HBase** **[points: 25]**

show version and status info from your HBase [5 points]

|  |
| --- |
| AWS EMR with Hbase    Starting Hbase    Starting Hbase (Continued)    Status    Version |

include screenshots of your 'vet\_visits\_hw6' table creation, and all commands to insert and query data. Explain the difference between 'scan' and 'get' commands [15 points]

|  |
| --- |
| Create ‘vet\_visits\_hw6’    Insert Data    Scan ‘vet\_visits\_hw6’    Scan by time range    Scan by row prefix filter    Get row    Get row and column      Explain the difference between 'scan' and 'get' commands:  It looks like the ‘scan’ command can return more than one row, and it can find rows based on the various info contained in the rows, such as rowkey or timestamp. So, the rowkey is not required to use the ‘scan’ command.  The ‘get’ command can only return one row, and it finds the row by rowkey. So, the rowkey is always required to use the ‘get’ command. |

Show results of the 'describe' table command [5 points]

|  |
| --- |
| Describe ‘vet\_visits\_hw6’ |

**Problem 2: Batch Views with HBase [points: 25]**

Show your HBase tables created for the Batch Views. Explain your design. [10 points]

|  |
| --- |
| Hbase table design:  Query1  {rowkey => ‘date:hour’}, {column => ‘count:unique\_urls’ }      The query 1 table is designed so that we can quickly lookup unique counts for each hour  Query2and3\_date  {rowkey => ‘date:hour:url’},  {column => ‘query2:count\_unique\_visitors’},  {column => ‘query3:count\_clicks’}      Query2and3\_url  {rowkey => ‘url:date:hour },  {column => ‘query2:count\_unique\_visitors’},  {column => ‘query3:count\_clicks’}      A combined table was made for query2 and query3, since they both can use the same rowkey of ‘date:hour:url’. A separate column family was made for each query because we are expecting that users of the database may not always want both sets of information when they run the queries. Since rowkeys are sorted, two tables are made, one is sorted by date and another one is sorted by url, this allows for fast retrieval of queries depending on if we are interested in a specific date/hour or a specific url  Query4  {rowkey => ‘date:hour:country’},  {column => ‘unique\_clicks:country’},  {column => ‘unique\_clicks:count’}      The query 4 table is designed so that we can find unique URLs by country by hour for a specified time range. Since it is expected that a time range would be specified, the rows are designed so that it is sorted by time. We can then query for information on a specific country in that time range. A column for country is created for easier lookup of row by specific country  Query5  {rowkey => ‘date’},  {column => ‘1st:url’}, {column => ‘1st:avg\_ttfb’},  {column => ‘2nd:url’}, {column => ‘2nd:avg\_ttfb’},  {column => ‘3rd:url’}, {column => ‘3rd:avg\_ttfb’},  {column => ‘4th:url’}, {column => ‘4th:avg\_ttfb’},  {column => ‘5th:url’}, {column => ‘5th:avg\_ttfb’}    The query 4 table is designed so that we can look up the 1st, 2nd, 3rd, 4th, and 5th top fastest TTFB url’s by day. Each of these are stored in separate column families, since it is possible that the hbase user, may not want all 5 of the top urls. For example, the user may only want information the 1st and 2nd fastest url for some specified days. |

Show the data you've inserted [5 points]

|  |
| --- |
| Query1    Query2 and Query 3 – sorted by date/hour    Query 2 and Query 3 – sorted by URL    Query 4    Query 5 |

Show execution and results of running your 5 queries in the HBase CLI - for your selected parameters [10 points]

|  |
| --- |
| Query 1  Get query by specific date/hour    Query2 and Query 3, rowkeys sorted by date/hour  Get query 2 by specific date/hour/url key    Get query 3 by specific date/hour/url key    Scan query 2 by specific date/hour    Scan query 3 by specific date/hour    Scan query 2 and query 3 by date/hour range    Query2 and Query 3, rowkeys sorted by url  Get query 2 by specific url/date/hour key  Get query 3 by specific url/date/hour key    Scan query 2 by specific url    Scan query 3 by specific url    Scan query 2 and 3 by url range    Query 4  Scan for specific country    Scan by time frame    Scan by time frame for specific country    Query 5  Get info by specific date    Get only top fastest url for a specified date |

**Problem 3: [points: 25] Spark with HBase - writes**

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

|  |
| --- |
| from pyspark import SparkContext, SparkConf  from pyspark.sql import SparkSession  import happybase  #get all the dateHourUrl and user pairs  def getDateHourUrlUser(line):      columns = line.split(",")      dateHour = columns[1].split(":")[0]      dateHour = dateHour.replace('T', ':', 1)      url = columns[2]      user = columns[3]      return ( ":".join([dateHour, url]), user )  if \_\_name\_\_ == "\_\_main\_\_":      """          Spark      """      spark = SparkSession\          .builder\          .appName("QueryTwo")\          .getOrCreate()      spark.sparkContext.setLogLevel("WARN")        #----------      #Query Two      logs = spark.sparkContext.textFile("s3://csci.e-88.principles.of.big.data.processing/input\_files/\*.csv")      #map, group, sort and collect results      #mapValues with set is used to remove duplicates      collection = logs.map(getDateHourUrlUser)      queryTwo = collection.map(lambda x: (x[0], x[1]))\          .groupByKey()\          .mapValues(lambda vals: len(set(vals)))\          .sortByKey()\      #put data in hbase      connection = happybase.Connection(host='localhost', port=9090)      connection.create\_table( 'query2', {'count': dict()} )      queryTwoTable = connection.table('query2')        for f in queryTwo.collect():          queryTwoTable.put(f[0],{b'count:count':str(f[1])})      print("DONE")      #-----------        spark.stop() |

Show execution and console output of your job - include a couple of screenshots [10 points]

|  |
| --- |
|  |

Show execution of the Query 2 from the HBase CLI and results for the specified keys [10 points]

|  |
| --- |
| 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 |

**Problem 4: Research of NoSQL DBs [points: 25]**

Fill in as much information about the Techniques, Functional and Non-functional Requirements of your chosen NoSQL DB you were able to identify. Use the same format as in the original Gessert's research: [points: 15]

|  |
| --- |
| https://lh4.googleusercontent.com/89-WkekSnPNvIeYV5z2fqEayNe-M1B7ocKcOYRGfhqMwEafPTKBa3pmhmTBL4Xe1CfyFHzjPPaCi3IU9vrZGVGK6qkT2UVsOxHl3AsJ7Kmh7ucPg41Ahb1KCy965xcqRwwOGfe5y |

Explain your DB decisions/choices and what information sources you have used [10 points]

|  |
| --- |
|  |

**Problem 5: Bonus: [points: 15] Spark with HBase - writes**

Show screenshots of the EMR cluster with both Spark and HBase installed [5 points]

|  |
| --- |
|  |

Show execution and console output of the Spark job running on the EMR cluster [5 points]

|  |
| --- |
|  |

Show results of the Query1 - retrieved by your job [5 points]

|  |
| --- |
| 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 |