John Attisha
11 June 2015

CSC 555 - Mining Big Data
Final Project

LEVERAGING HADOOP TO EXPLORE PRESCRIBER-LEVEL MEDICARE DATA

ABSTRACT

The Centers for Medicare and Medicaid Services (CMS) have recently released a dataset containing all medications prescribed under Medicare Part D throughout 2013. The dataset includes specific information regarding physicians and other health care providers who have written prescriptions covered by Medicare Part D, as well as the types of medications, quantities, costs and other relevant data. In this report, I demonstrate the advantages of using parallel processing when querying relatively large datasets by comparing run times between a three-node Hadoop cluster to a single machine. Additionally, I compare performance between Hive and Pig on both machine setups. The same queries were executed on both the single machine as well as the three-node cluster, resulting in a reduction of run times between 40% and 58% when leveraging the parallel processing capabilities of the Hadoop cluster. With regard to the comparison of Hive and Pig, results show Hive outperformed Pig on every query in terms of absolute run times, regardless of the machine setup. Nevertheless, Pig tended to have a greater percentage improvement in run time when going from a single machine to a three-node cluster, as compared to the improvement realized with Hive.

INTRODUCTION

The Centers for Medicare and Medicaid Services (CMS) have recently released a dataset containing information on all medication prescribed by physicians and other health care providers under Medicare Part D throughout 2013. The dataset consists of 19 variables, including: Physician's name; Specialty; Location; Medication prescribed; Total number of original prescriptions written for each medication; Total cost of the medication; and a number of other variables. Details of each variable are provided in a subsequent section of this report, titled "Data Description." Each row in the dataset can be uniquely identified by the physician's

ID number and the name of the medication prescribed. The dataset contains over 23 million records, representing more than one million distinct health care providers and encompassing $103 billion in prescription medications covered under the Medicare Part D program in 2013. The overall size of the data file is 2.7 gigabytes.

The purpose of my project is two-fold. First, I would like to demonstrate the advantages (and potential disadvantages) of using parallel processing via a three-node Hadoop cluster when analyzing large datasets, as compared to using a single machine. I plan on running a number of queries to compare and contrast performance and run time on both the three-node cluster and the single machine. In addition to illustrating the differences between two differently-sized clusters, I also plan to compare performance between Hive and Pig on both of the clusters.

INITIAL SETUP OF CLUSTERS

My first step was to create a number of instances in Amazon EC2. All instances were created using type 't2.small' and each contained a capacity of 8GB. Therefore, my three-node cluster would have a total capacity of 24GB, while my single machine would only have a capacity of 8GB. At this point I installed and configured Hadoop on the single machine, as well as on what would be the master node of the three-node cluster. This required setting up the Java environment as well as the Hadoop configuration files. For example, it was necessary to edit the *conf/hadoop-env.sh* file in order to set the value of *JAVA HOME* to */usr/lib/jvm/java-1,7,0-openjdk.x86_64*. The three configuration files (*conf/core-site.xml, conf/mapred-site.xml, and conf/hdfs-site.xml*) also required proper setup prior to using Hadoop. For purposes of the project, I decided to use a replication factor of 1 (specified within *conf/hdfs-site.xml*) for both the three-node cluster and the single machine. Once complete, Hadoop was formatted. At this point, a private/public key pair was generated, and the public key was added to the list of authorized keys which would allow me to login to the localhost without having to use a password.

In order to finish setting up my three-node cluster, I had to adjust the configuration files, setup the *conf/master* and *conf/slaves* files, and setup remote SSH access from the master to the

worker nodes. In addition to adding the Private IP addresses of the worker nodes to the *conf/slaves* file, I also added the IP address of the master node, which allows the master node to be used as a worker node as well. Setting up remote SSH access was done by creating public/private key pairs on the master node and adding the public key to the list of authorized keys on each of the worker nodes (as well as to the master's list of authorized keys). This procedure was rather cumbersome as the public key is quite lengthy, and ensuring that accidental line breaks were not introduced when copying it over to the worker nodes required great attention to detail! (Simple copying and pasting of the key introduced a number of line breaks which had to be removed prior to gaining access to the worker nodes.) Once I gained access to the two worker nodes, I had to pack up the Hadoop directory into a .tar file and copy it onto the two new nodes. Then I had to log into the new nodes, unpack the .tar file, and format Hadoop. Note that I also formatted Hadoop on the master node at this point as well. At this point, setting up the three-node cluster was almost complete. The last thing to do was to open the firewall by going into Amazon EC2 and adjusting the security group settings. The following ports were opened: 50070, 50075, 50060, 50030, 50010, 9001, and 9000. Of course, port 22 was already opened. I was able to go to my master node and start Hadoop (*bin/start-all.sh*). The cluster was indeed operational, as the dfshealth.jsp webpage showed three live nodes were up and running. The same was true for the single machine.

Now I had setup both my single machine and my three-node cluster, as well as installed and configured Hadoop. The only thing remaining was to download and install Hive and Pig on both the single node machine as well as on the master node of the three-node cluster.

As mentioned in the introduction, the data file (*PartD_Prescriber_PUF_NPI_DRUG_13.txt*) is 2.7 GB. Therefore, I decided to create a storage volume in order to house my large data file prior to putting it on HDFS for both the single machine and the three-node cluster. I first made a new directory called '*project*' on HDFS, and then put the file from my drive to the HDFS of both the single machine as well as the three-node cluster via the command below.

bin/hadoop fs -put /drive1/PartD_Prescriber_PUF_NPI_DRUG_13.txt /project

DATA DESCRIPTION

As previously mentioned, there are over 23 million observations contained in the Medicare dataset. Each row consists of 19 variables. Descriptions of each variable can be found in the table below.

VARIABLE NAME	**DESCRIPTION**
NPI	National Provider Identifier (Unique for each Physician / Health Care Provider)
NPPES_PROVIDER_LAST_ORG_NAME	Last name of physician. NPPES is an acronym for the National Plan & Provider Enumeration System.
NPPES_PROVIDER_FIRST_NAME	First name of physician.
NPPES_PROVIDER_CITY	City of physician.
NPPES_PROVIDER_STATE	State of physician.
SPECIALTY_DESCRIPTION	Physician's specialty.
DESCRIPTION_FLAG	Source of physician's specialty.
DRUG_NAME	Brand name of medication prescribed.
GENERIC_NAME	Generic name of medication prescribed, if applicable.
BENE_COUNT	Number of Medicare beneficiaries (total number of unique patients prescribed a certain medication).

TOTAL_CLAIM_COUNT	Number of claims for a certain medication (different from BENE_COUNT in that it included refills).
TOTAL_DAY_SUPPLY	Number of days' supply for a medication (for all claims).
TOTAL_DRUG_COST	Total cost paid for all claims.
BENE_COUNT_GE65	Number of unique patients prescribed a certain medication, where the patients were 65 years of age or older.
BENE_COUNT_GE65_REDACT_FLAG	Explanation as to why BENE_COUNT_GE65 may have not been included.
TOTAL_CLAIM_COUNT_GE65	Number of claims for a certain medication (including refills) where the patients were 65 years of age or older.
GE65_REDACT_FLAG	Explanation as to why GE65 fields were redacted.
TOTAL_DAY_SUPPLY_GE65	Number of days' supply for a medication where the patients were 65 years of age or older.
TOTAL_DRUG_COST_GE65	Total cost paid for all claims by patients who were 65 years of age or older.

HIVE QUERIES

Prior to performing any queries, I had to create an external table. Note that the data file was located on HDFS in the */project* directory. The command below was executed on both the single machine as well as the three-node cluster. Once executed, I was ready to perform some Hive queries to compare performance and run time, as well as to obtain some interesting results from the data.

CREATE EXTERNAL TABLE Medicare123 (NPI STRING, NPPES_PROVIDER_LAST_ORG_NAME STRING, NPPES_PROVIDER_FIRST_NAME STRING, NPPES_PROVIDER_CITY STRING, NPPES_PROVIDER_STATE STRING, SPECIALTY_DESC STRING, DESCRIPTION_FLAG STRING, DRUG_NAME STRING, GENERIC_NAME STRING, BENE_COUNT FLOAT, TOTAL_CLAIM_COUNT FLOAT, TOTAL_DAY_SUPPLY FLOAT, TOTAL_DRUG_COST FLOAT, BENE_COUNT_GE65 FLOAT, BENE_COUNT_GE65_REDACT_FLAG STRING, TOTAL_CLAIM_COUNT_GE65 FLOAT, GE65_REDACT_FLAG STRING, TOTAL_DAY_SUPPLY_GE65 FLOAT, TOTAL_DRUG_COST_GE65 FLOAT) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' STORED AS TEXTFILE LOCATION '/project';

1. Let's start by performing a simple count of all the rows in the table:

   SELECT COUNT(*) FROM Medicare123;

   Single Machine:

```
MapReduce Jobs Launched:
Job 0: Map: 11  Reduce: 1   Accumulative CPU: 37.37 sec   HDFS Read: 2844595450 H
DFS Write: 9 SUCESS
Total MapReduce CPU Time Spent: 37 seconds 370 msec
OK
23650521
Time taken: 135.7 seconds
```

   Three-Node Cluster:

```
MapReduce Jobs Launched:
Job 0: Map: 12  Reduce: 1   Accumulative CPU: 40.48 sec   HDFS Read: 2844597006 H
DFS Write: 9 SUCESS
Total MapReduce CPU Time Spent: 40 seconds 480 msec
OK
23650521
Time taken: 74.576 seconds
```

The single node machine took 135.7 seconds to complete the task, while the three-node cluster took 74.576 seconds. We see a rather significant improvement in run time with the three-node cluster. The query only required one MapReduce pass. For the single machine, 11 mappers were used, and 1 reducer. The three-node cluster used 12 mappers and 1 reducer. The three node machine likely had 4 mappers on each of its nodes, running in parallel. (This number may not be exact, as it depends on how the blocks were distributed and what other data may have been stored on the nodes.) The three-node cluster's ability to perform parallel processing is at the heart of why the query was able to be completed substantially faster than with the single machine.

2. Now let's try to find those prescriptions with the highest total cost. Preliminary queries identified that there were almost 3,000 unique drugs. Therefore, I will restrict my output to those drugs with a total cost greater than $1 billion. Additionally, I will re-direct the output to a directory titled 'CostlyDrugs' :

   INSERT OVERWRITE DIRECTORY 'CostlyDrugs' SELECT DRUG_NAME, SUM(TOTAL_DRUG_COST) AS total FROM Medicare123 GROUP BY DRUG_NAME HAVING total > 1000000000 SORT BY total DESC;

Single Machine:

```
12 Rows loaded to CostlyDrugs
MapReduce Jobs Launched:
Job 0: Map: 11  Reduce: 3   Accumulative CPU: 71.4 sec   HDFS Read: 2844595450 HD
FS Write: 697 SUCESS
Job 1: Map: 1  Reduce: 1   Accumulative CPU: 1.66 sec   HDFS Read: 1778 HDFS Writ
e: 358 SUCESS
Total MapReduce CPU Time Spent: 1 minutes 13 seconds 60 msec
OK
Time taken: 192.991 seconds
```

Three-Node Cluster:

```
12 Rows loaded to CostlyDrugs
MapReduce Jobs Launched:
Job 0: Map: 12  Reduce: 3   Accumulative CPU: 73.02 sec   HDFS Read: 2844597006 H
DFS Write: 697 SUCESS
Job 1: Map: 3  Reduce: 1   Accumulative CPU: 2.65 sec   HDFS Read: 2170 HDFS Writ
e: 358 SUCESS
Total MapReduce CPU Time Spent: 1 minutes 15 seconds 670 msec
OK
Time taken: 109.252 seconds
```

File Contents:

```
hive> quit;
[ec2-user@ip-172-31-8-242 hive-0.8.1-bin]$ cd $HADOOP_HOME
[ec2-user@ip-172-31-8-242 hadoop-0.20.205.0]$ bin/hadoop fs -ls /user/ec2-user/Co
stlyDrugs
Warning: $HADOOP_HOME is deprecated.

Found 1 items
-rw-r--r--   1 ec2-user supergroup        358 2015-06-06 02:18 /user/ec2-user/Cos
tlyDrugs/000000_0
[ec2-user@ip-172-31-8-242 hadoop-0.20.205.0]$ bin/hadoop fs -cat /user/ec2-user/C
ostlyDrugs/000000_0
Warning: $HADOOP_HOME is deprecated.

NEXIUM2.3202909270216293E9
CRESTOR2.0892383506263657E9
ADVAIR DISKUS2.0692817015175l7E9
ABILIFY1.8785884395862427E9
SPIRIVA1.7787760552213745E9
CYMBALTA1.7772366455238037E9
NAMENDA1.4520098486177979E9
JANUVIA1.3089587488572998E9
LANTUS SOLOSTAR1.2102432885598145E9
REVLIMID1.1822238545839844E9
LANTUS1.1613959175170593E9
DIOVAN1.12311417123497E9
```

First, we see that the three-node cluster indeed completed the job much faster than the single machine. Two MapReduce passes were required: one to obtain the sum of the total cost for each medication, and one to sort the result in descending order by the total cost. For the first pass, the single machine utilized 11 mappers and 1 reducer, while the three-node cluster used 12 mappers and 1 reducer. For the sorting pass (i.e. the second pass), the three-node cluster used three mappers, while the single machine only used one mapper. Nevertheless, the second pass wasn't necessarily where the three-node cluster was able to save processing time. Rather, the first phase was likely where the true advantage of parallel processing was able to shine.

The output of our query shows that there were 12 distinct medications covered by Medicare Part D which cost over $1 billion each throughout 2013. Additionally, three of these medications cost over $2 billion: Nexium, Crestor, and Advair.

3. For the next query, I will try to find the medications *specifically prescribed in Chicago, IL* with the highest total cost. I will restrict my query to those medications having a total cost greater than $10 million. The output will be sent to a directory titled 'ChicagoDrugs'.

INSERT OVERWRITE DIRECTORY 'ChicagoDrugs' SELECT DRUG_NAME, SUM(TOTAL_DRUG_COST) AS total FROM Medicare123 WHERE (NPPES_PROVIDER_STATE='IL' AND NPPES_PROVIDER_CITY='CHICAGO') GROUP BY DRUG_NAME HAVING total > 10000000 SORT BY total DESC;

Single Machine:

```
8 Rows loaded to ChicagoDrugs
MapReduce Jobs Launched:
Job 0: Map: 11  Reduce: 3   Accumulative CPU: 58.88 sec   HDFS Read: 2844595450 H
DFS Write: 556 SUCESS
Job 1: Map: 1  Reduce: 1   Accumulative CPU: 1.65 sec   HDFS Read: 1637 HDFS Writ
e: 235 SUCESS
Total MapReduce CPU Time Spent: 1 minutes 0 seconds 530 msec
OK
Time taken: 174.8 seconds
```

Three-Node Cluster:

```
8 Rows loaded to ChicagoDrugs
MapReduce Jobs Launched:
Job 0: Map: 12  Reduce: 3   Accumulative CPU: 60.8 sec   HDFS Read: 2844597006 HD
FS Write: 556 SUCESS
Job 1: Map: 3  Reduce: 1   Accumulative CPU: 2.62 sec   HDFS Read: 2029 HDFS Writ
e: 235 SUCESS
Total MapReduce CPU Time Spent: 1 minutes 3 seconds 420 msec
OK
Time taken: 98.306 seconds
```

File Contents:

```
hive> quit;
[ec2-user@ip-172-31-13-27 hive-0.8.1-bin]$ cd $HADOOP_HOME
[ec2-user@ip-172-31-13-27 hadoop-0.20.205.0]$ bin/hadoop fs -ls /user/ec2-user/Ch
icagoDrugs
Warning: $HADOOP_HOME is deprecated.

Found 1 items
-rw-r--r--   1 ec2-user supergroup        235 2015-06-08 15:41 /user/ec2-user/Chi
cagoDrugs/000000_0
[ec2-user@ip-172-31-13-27 hadoop-0.20.205.0]$ bin/hadoop fs -cat /user/ec2-user/C
hicagoDrugs/000000_0
Warning: $HADOOP_HOME is deprecated.

ABILIFY1.3002619017211914E7
NEXIUM1.2752062643310547E7
ADVAIR DISKUS1.2540461250488281E7
TRUVADA1.2190522922851562E7
CRESTOR1.1570525786865234E7
JANUVIA1.1057609321044922E7
ATRIPLA1.0258287380859375E7
DIOVAN1.022808633758545E7
```

Compared to the previous query, the constraints of 'Chicago' and 'IL' seemed to have reduced the total run time for both the single machine and the three-node cluster. The single machine took 174.8 seconds to run, and the three-node cluster took 98 seconds. The job required two passes. During the first pass, key:value pairs were only generated from those rows which met the city and state constraints. This first pass is likely where most of the time was saved on the three-node cluster. In one of the next sections, I will try to execute the same type of query using Pig and compare run times to Hive, on both the single machine and the three-node cluster.

As the output shows, we see that the most costly drugs in Chicago align with some of the most costly drugs across the country. Abilify accounted for over $13 million in total drug costs paid out my Medicare Part D. Nexium came in second place, with $12.75 million.

4. Next let's find the states with the highest total drug cost. The query will provide the state abbreviation as well as a figure corresponding to the total drug costs. The final result will be sorted in descending order by total costs, and the actual output will be redirected to the 'StateCosts' directory (/user/ec2-user/StateCosts).

   INSERT OVERWRITE DIRECTORY 'StateCosts' SELECT NPPES_PROVIDER_STATE, SUM(TOTAL_DRUG_COST) AS totalcost FROM Medicare123 GROUP BY NPPES_PROVIDER_STATE SORT BY totalcost DESC;

   Single Machine:

```
Moving data to: StateCosts
62 Rows loaded to StateCosts
MapReduce Jobs Launched:
Job 0: Map: 11  Reduce: 3   Accumulative CPU: 63.51 sec   HDFS Read: 2844595450 H
DFS Write: 2034 SUCESS
Job 1: Map: 1  Reduce: 1   Accumulative CPU: 1.69 sec   HDFS Read: 3112 HDFS Writ
e: 1424 SUCESS
Total MapReduce CPU Time Spent: 1 minutes 5 seconds 200 msec
OK
Time taken: 175.388 seconds
```

   Three-Node Cluster:

```
62 Rows loaded to StateCosts
MapReduce Jobs Launched:
Job 0: Map: 12  Reduce: 3   Accumulative CPU: 64.38 sec   HDFS Read: 2844597006 H
DFS Write: 2034 SUCESS
Job 1: Map: 3  Reduce: 1   Accumulative CPU: 2.63 sec   HDFS Read: 3507 HDFS Writ
e: 1427 SUCESS
Total MapReduce CPU Time Spent: 1 minutes 7 seconds 10 msec
OK
Time taken: 104.285 seconds
```

File Output:

```
[ec2-user@ip-172-31-13-27 hadoop-0.20.205.0]$ bin/hadoop fs -cat /user/ec2-user/S
tateCosts/000000_0
Warning: $HADOOP_HOME is deprecated.

CA8.205892051032583E9
NY6.459818151195004E9
FL5.949602138710251E9
TX5.511245065931952E9
PA4.2167474961736693E9
OH3.6080070460635753E9
NC2.9354821983519087E9
MI2.7889315380449595E9
NJ2.6239764818841324E9
IL2.569950419531919E9
GA2.3792620007680793E9
TN2.135176547120986E9
IN1.895110217840492E9
MO1.8857753501443582E9
MA1.6549987319763966E9
VA1.5999090904720373E9
AL1.5767040129380026E9
KY1.5714822977588723E9
LA1.401041436058755E9
WI1.3830659792482364E9
SC1.2482954629882E9
AZ1.2428770930539904E9
WA1.1630920612093592E9
MN1.10385896887937E9
CT1.0561713867967786E9
```

The query required two MapReduce passes, and the three-node cluster took 104 seconds to complete, while the single machine took 175 seconds. There were a total of 62 keys, each corresponding to either a state within the U.S., or a U.S. territory (such as Guam or Puerto Rico). For ease of viewing, I only included the top 25 states in the screenshot above, which account for those states with a total drug cost of over $1 billion (as paid out by Medicare Part D for the year 2013). California came out at the top of the list, with a total drug cost of more than $8.2 billion. New York, Florida, Texas, and Pennsylvania rounded out the top five.

5. Finally, for our last Hive query, I will find the total cost of medication grouped by medical specialty. My query will be as follows:

INSERT OVERWRITE DIRECTORY 'SpecialtyCost' SELECT SPECIALTY_DESC, SUM(TOTAL_DRUG_COST) AS total FROM Medicare123 GROUP BY SPECIALTY_DESC SORT BY total DESC;

Single Machine:

```
Moving data to: SpecialtyCost
203 Rows loaded to SpecialtyCost
MapReduce Jobs Launched:
Job 0: Map: 11  Reduce: 3   Accumulative CPU: 65.03 sec   HDFS Read: 2844595450 H
DFS Write: 10147 SUCESS
Job 1: Map: 1  Reduce: 1   Accumulative CPU: 1.74 sec   HDFS Read: 11228 HDFS Wri
te: 8582 SUCESS
Total MapReduce CPU Time Spent: 1 minutes 6 seconds 770 msec
OK
Time taken: 183.861 seconds
```

Three-Node Cluster:

```
Moving data to: SpecialtyCost
203 Rows loaded to SpecialtyCost
MapReduce Jobs Launched:
Job 0: Map: 12  Reduce: 3   Accumulative CPU: 65.11 sec   HDFS Read: 2844597006 H
DFS Write: 10147 SUCESS
Job 1: Map: 3  Reduce: 1   Accumulative CPU: 2.68 sec   HDFS Read: 11620 HDFS Wri
te: 8582 SUCESS
Total MapReduce CPU Time Spent: 1 minutes 7 seconds 790 msec
OK
Time taken: 100.34 seconds
```

Final Output:

```
[ec2-user@ip-172-31-8-242 hadoop-0.20.205.0]$ bin/hadoop fs -cat /user/ec2-user/S
pecialtyCost/000000_0
Warning: $HADOOP_HOME is deprecated.

Internal Medicine2.130705464266242E10
Family Practice1.722132295763989E10
Nurse Practitioner4.658594072977973E9
Neurology4.139061142531748E9
Psychiatry4.055759537107511E9
Cardiology3.6757179487285357E9
Hematology/Oncology2.5516201029451404E9
Physician Assistant2.1661029724498925E9
Pulmonary Disease2.0821184383153584E9
Rheumatology1.9359357405629818E9
Ophthalmology1.8057438396531715E9
Endocrinology1.7993611238203425E9
Nephrology1.621546560307877E9
Infectious Disease1.3984337170171113E9
General Practice1.3894606183056877E9
Gastroenterology1.1728627078418393E9
```

The three-node cluster required 100 seconds to complete, while the single machine took 183 seconds. There were a total of 203 key:value pairs written to the output file, with each key corresponding to a unique medical specialty. Sorting the data in descending order by total cost, we see that the first 16 specialties each prescribed over $1 billion in total medications covered under Medicare Part D for 2013. Internal Medicine topped the list, with over $2.1 billion in total drug costs. Family Practice came in second place with $1.7 billion.

PIG QUERIES

At this point I'd like to use Pig to perform three of the same queries we've seen above. As with Hive, one of the first steps is to develop a command which will 'load' the data:

```
Medicare456 = LOAD '/project/' USING PigStorage ('\t') AS (NPI:CHARARRAY,
NPPES_PROVIDER_LAST_ORG_NAME: CHARARRAY, NPPES_PROVIDER_FIRST_NAME:
CHARARRAY, NPPES_PROVIDER_CITY: CHARARRAY, NPPES_PROVIDER_STATE: CHARARRAY,
SPECIALTY_DESC: CHARARRAY, DESCRIPTION_FLAG: CHARARRAY, DRUG_NAME: CHARARRAY,
GENERIC_NAME: CHARARRAY, BENE_COUNT:FLOAT, TOTAL_CLAIM_COUNT:FLOAT,
TOTAL_DAY_SUPPLY:FLOAT, TOTAL_DRUG_COST:FLOAT, BENE_COUNT_GE65:FLOAT,
BENE_COUNT_GE65_REDACT_FLAG: CHARARRAY, TOTAL_CLAIM_COUNT_GE65:FLOAT,
GE65_REDACT_FLAG: CHARARRAY, TOTAL_DAY_SUPPLY_GE65:FLOAT,
TOTAL_DRUG_COST_GE65:FLOAT);
```

1. Let's start by finding a count of all the rows in the file. After the initial 'load' command, three additional commands will need to be executed:

```
MedicareRowCount = GROUP Medicare456 ALL;
Count = FOREACH MedicareRowCount GENERATE COUNT(Medicare456);
DUMP Count;
```

Single Machine:

```
HadoopVersion   PigVersion      UserId  StartedAt       FinishedAt      Features
0.20.205.1      0.9.2   ec2-user        2015-06-09 15:03:59     2015-06-09 15:12:
57      GROUP_BY

Success!
```

Three-Node Cluster:

```
HadoopVersion   PigVersion      UserId  StartedAt       FinishedAt      Features
0.20.205.1      0.9.2   ec2-user        2015-06-09 15:02:43     2015-06-09 15:06:
05      GROUP_BY

Success!
```

The single machine took a total of 538s to complete the row count, while the three-node cluster took 202s. Both run times were considerably longer than the same query when run on Hive. The query consisted of one MapReduce pass. For both the single machine and the three-node cluster, Pig used 43 mappers and 1 reducer to complete the job, which was much higher than any query completed with Hive. Whereas Hive used no more than 12 mappers for any of its MapReduce jobs, Pig was using 1 mapper for each 64MB block of the data (since the data file was 2.67GB, this amounted to 43 blocks, and thus 43 mappers). With 43 mappers, one can see why the run time was taking so long to perform the row count query. For example, each mapper likely takes time to start up and get running, which may have added a considerable amount of time to complete the query.

Therefore, in order to optimize my query, I decided to change the number of mappers used by Pig for this query. The commands below changed the number of mappers from 43 to 14:

SET pig.noSplitCombination false;
SET pig.maxCombinedSplitSize 250000000;

The first command allows Pig to combine input splits, which results in a lower total number of map tasks. The default for this setting is 'true'. The second command specifies the maximum amount of data, as measured in bytes, to be processed by each map. I decided to set this value to 250,000,000 bytes. Once the changes were made, the following command was re-run on the three-node cluster:

DUMP Count;

```
HadoopVersion   PigVersion      UserId  StartedAt       FinishedAt      Features
0.20.205.1      0.9.2   ec2-user        2015-06-09 19:53:59     2015-06-09 19:56:27     GROUP_BY

Success!
```

As you can see, the time required to complete the job on the three-node cluster went down noticeably. For this run, Pig only used 14 mappers, and the three-node cluster needed 148s to complete the job, which was about 1 minute faster than when it used 43 mappers. After trying it out on the three-node cluster, I also changed these settings on the single machine and re-ran the query (screenshot not included). The run time for the single machine also went down, from 538s to 353s.

Final Output:

```
2015-06-09 15:06:05,718 [main] INFO  org.apache.pig.backend.hadoop.executionengin
e.mapReduceLayer.MapReduceLauncher - Success!
2015-06-09 15:06:05,727 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileIn
putFormat - Total input paths to process : 1
2015-06-09 15:06:05,727 [main] INFO  org.apache.pig.backend.hadoop.executionengin
e.util.MapRedUtil - Total input paths to process : 1
(23650521)
```

The final output coincides with the output of our Hive query. There are indeed a total of 23,650,521 rows in the dataset.

Note that at this point all of the following queries were analyzed using both the default settings (where the first pass of each query used 43 mappers), as well as with the newly added constraints (where the first pass of each query used 14 mappers). Reducing the number of mappers to 14 improved the run-time of every query, for both the single machine and the three-node cluster. Therefore, the next two queries below have screenshots corresponding to the use of only 14 mappers, rather than 43.

2. For our second query let's once again try to find the most prescribed medications in Chicago, IL. Here we will be using a filter to tell Pig to only analyze those rows whose city=='Chicago' and whose state=='IL'.

```
ChicagoRows = FILTER Medicare456 BY NPPES_PROVIDER_STATE == 'IL' AND
NPPES_PROVIDER_CITY == 'CHICAGO';
Chi2Row = FOREACH ChicagoRows GENERATE DRUG_NAME, TOTAL_DRUG_COST;
GroupedRxs = GROUP Chi2Row BY (DRUG_NAME);
ChiOutput = FOREACH GroupedRxs GENERATE group as grp,
SUM(Chi2Row.TOTAL_DRUG_COST) as ChiTotal;
ChiOrdered = ORDER ChiOutput BY ChiTotal DESC;
STORE ChiOrdered INTO 'ChicagoCostsPig2' USING PigStorage ('\t');
```

Single Machine:

```
HadoopVersion   PigVersion      UserId  StartedAt       FinishedAt      Features
0.20.205.1      0.9.2   ec2-user        2015-06-09 20:34:45     2015-06-09 20:39:
07      GROUP_BY,ORDER_BY,FILTER

Success!
```

Three-Node Cluster:

```
HadoopVersion   PigVersion      UserId  StartedAt       FinishedAt      Features
0.20.205.1      0.9.2   ec2-user        2015-06-09 20:34:37     2015-06-09 20:37:
29      GROUP_BY,ORDER_BY,FILTER

Success!
```

Summary Info for the Three-Node Cluster:

```
Job Stats (time in seconds):
JobId   Maps    Reduces MaxMapTime      MinMapTIme      AvgMapTime      MaxReduce
Time    MinReduceTime   AvgReduceTime   Alias   Feature Outputs
job_201506092019_0002   14      3       27      18      21      138     15      9
6       Chi2Row,ChiOutput,ChicagoRows,GroupedRxs,Medicare456    GROUP_BY,COMBINER
job_201506092019_0003   1       1       6       6       6       12      12      1
2       ChiOrdered      SAMPLER
job_201506092019_0004   1       1       6       6       6       12      12      1
2       ChiOrdered      ORDER_BY        hdfs://localhost:9000/user/ec2-user/Chica
goCostsPig2,
```

The single machine needed 262s to complete the task, while the three-node cluster took 172s. (The corresponding query on Hive took 174s for the single machine and 98s for the three-node cluster.) The screen shot above (taken from the three-node cluster) shows that the query required three passes. The first pass had 14 mappers and 3 reducers. The next two passes each used 1 mapper and 1 reducer. This was also true for the single machine. The final output can be seen below. While the file contains over 3,000 rows of unique medications, only those with over $10 million in total cost are shown in the screenshot. (This query was also done with Pig using 43 mappers, and the run times were as follows: the single machine took 442s, and the three-node cluster took 218s.)

Final Output:

```
grunt> quit;
[ec2-user@ip-172-31-13-27 pig-0.9.2]$ cd $HADOOP_HOME
[ec2-user@ip-172-31-13-27 hadoop-0.20.205.0]$ bin/hadoop fs -ls /user/ec2-user/Ch
icagoCostsPig2
Warning: $HADOOP_HOME is deprecated.

Found 3 items
-rw-r--r--   1 ec2-user supergroup          0 2015-06-09 20:37 /user/ec2-user/Chi
cagoCostsPig2/_SUCCESS
drwxr-xr-x   - ec2-user supergroup          0 2015-06-09 20:36 /user/ec2-user/Chi
cagoCostsPig2/_logs
-rw-r--r--   1 ec2-user supergroup      42921 2015-06-09 20:37 /user/ec2-user/Chi
cagoCostsPig2/part-r-00000
[ec2-user@ip-172-31-13-27 hadoop-0.20.205.0]$ bin/hadoop fs -cat /user/ec2-user/C
hicagoCostsPig2/part-r-00000
Warning: $HADOOP_HOME is deprecated.

ABILIFY 1.3002619017211914E7
NEXIUM  1.2752062643310547E7
ADVAIR DISKUS   1.2540461250488281E7
TRUVADA 1.2190522922851562E7
CRESTOR 1.1570525786865234E7
JANUVIA 1.1057609321044922E7
ATRIPLA 1.0258287380859375E7
DIOVAN  1.022808633758545E7
```

3. Our final query finds the total cost of medications, grouped by state. I started out by selecting the two columns I wanted (state and total cost). From there I told Pig to group the total cost by state, and then to order by total cost. Finally, I used the STORE command to save the output on HDFS in the 'StateCountPig2' directory.

```
TwoColStateCost = FOREACH Medicare456 GENERATE NPPES_PROVIDER_STATE,
TOTAL_DRUG_COST;
GroupedStates = GROUP TwoColStateCost BY (NPPES_PROVIDER_STATE);
TotalCostByState = FOREACH GroupedStates GENERATE group as grp,
SUM(TwoColStateCost.TOTAL_DRUG_COST) as totalcost;
StatesOrdered = ORDER TotalCostByState BY totalcost DESC;
STORE StatesOrdered INTO 'StateCountPig2' USING PigStorage ('\t');
```

Single Machine:

```
HadoopVersion   PigVersion      UserId  StartedAt       FinishedAt      Features
0.20.205.1      0.9.2   ec2-user        2015-06-09 21:05:40     2015-06-09 21:11:
27      GROUP_BY,ORDER_BY

Success!
```

Three-Node Cluster:

```
HadoopVersion   PigVersion      UserId  StartedAt       FinishedAt      Features
0.20.205.1      0.9.2   ec2-user        2015-06-09 21:05:24     2015-06-09 21:08:
42      GROUP_BY,ORDER_BY

Success!
```

Summary Info for the Three-Node Cluster:

```
Job Stats (time in seconds):
JobId   Maps    Reduces MaxMapTime      MinMapTIme      AvgMapTime      MaxReduce
Time    MinReduceTime   AvgReduceTime   Alias   Feature Outputs
job_201506091923_0008   14      3       39      18      32      69      63      6
6       GroupedStates,Medicare456,TotalCostByState,TwoColStateCost      GROUP_BY,
COMBINER
job_201506091923_0009   1       1       6       6       6       12      12      1
2       StatesOrdered   SAMPLER
job_201506091923_0010   1       1       6       6       6       12      12      1
2       StatesOrdered   ORDER_BY        hdfs://ip-172-31-13-27.us-west-2.compute.
internal:9000/user/ec2-user/StateCountPig2,
```

Final Output:

```
grunt> quit;
[ec2-user@ip-172-31-13-27 pig-0.9.2]$ cd $HADOOP_HOME
[ec2-user@ip-172-31-13-27 hadoop-0.20.205.0]$ bin/hadoop fs -ls /user/ec2-user/St
ateCountPig2
Warning: $HADOOP_HOME is deprecated.

Found 3 items
-rw-r--r--   1 ec2-user supergroup          0 2015-06-09 21:08 /user/ec2-user/Sta
teCountPig2/_SUCCESS
drwxr-xr-x   - ec2-user supergroup          0 2015-06-09 21:08 /user/ec2-user/Sta
teCountPig2/_logs
-rw-r--r--   1 ec2-user supergroup       1423 2015-06-09 21:08 /user/ec2-user/Sta
teCountPig2/part-r-00000
[ec2-user@ip-172-31-13-27 hadoop-0.20.205.0]$ bin/hadoop fs -cat /user/ec2-user/S
tateCountPig2/part-r-00000
Warning: $HADOOP_HOME is deprecated.

CA      8.205892051032583E9
NY      6.4598181511950035E9
FL      5.949602138710251E9
TX      5.5112450659319 53E9
PA      4.21674749617367E9
OH      3.60800704606357 53E9
NC      2.9354821983519087E9
MI      2.788931538044 96E9
NJ      2.6239764818841324E9
IL      2.569950419531919E9
GA      2.379262000768079E9
TN      2.135176547120986E9
IN      1.895110217840492E9
MO      1.8857753501443582E9
MA      1.6549987319763966E9
VA      1.5999090904720373E9
AL      1.5767040129380026E9
KY      1.571482297758872E9
LA      1.4010414360587547E9
WI      1.3830659792482367E9
SC      1.2482954629882E9
AZ      1.2428770930539904E9
WA      1.1630920612093594E9
MN      1.1038589688793702E9
CT      1.0561713867967786E9
```

The single machine needed 347s to complete, while the three-node cluster took 198s. As seen above, this query also utilized 14 mappers and 3 reducers in the first pass, and 1 mapper and 1 reducer in the 2nd and 3rd passes. This was true for both the three-node cluster and the single machine. The corresponding query using Hive took 175s for the single machine and 104s for the three-node cluster. (The query was also completed using the default 43 mappers, and the run times were as follows: the single machine took 573s, and the three-node cluster took 267s.)

SUMMARY

Tables summarizing the performance run times of all the queries can be seen below. The times reported for the Pig queries correspond to those where 14 mappers were used, rather than 43.

HIVE:

Query Number	Query Description	Single Machine Run Time	Three-Node Cluster Run Time	% Reduction Using Three-Node Cluster
1	Row count	135.7s	74.576s	-45.04%
2	Total cost grouped by medication	192.991s	109.252s	-43.39%
3	Total cost grouped by medication (only prescribed in Chicago, IL)	174.8s	98.306s	-43.76%
4	Total cost grouped by state	175.388s	104.285s	-40.54%
5	Total cost grouped by medical specialty	183.861s	100.34s	-45.43%

PIG:

Query Number	Query Description	Single Machine Run Time	Three-Node Cluster Run Time	% Reduction Using Three-Node Cluster
1	Row count	353s	148s	-58.07%
2	Total cost grouped by medication (only prescribed in Chicago, IL)	262s	172s	-34.35%
3	Total cost grouped by state	347s	198s	-42.94%

Regarding both the Hive and Pig queries, the performance of the three-node cluster was always noticeably faster than the single machine. Of course the recorded run times cannot indicate a statistically significant difference without further analysis, but a simple look at the run times above positively illustrates the three-node cluster was indeed more computationally efficient than the single machine with regard to processing Hive and Pig queries.

Unfortunately, the improved performance on the three-node cluster never amounted to a run time which was in actuality one-third that of the single machine. In fact, the best improvement using Hive was Query #5 (Total prescription costs grouped by medical specialty), where we saw that the three-node cluster had a 45.43% reduction in run time compared to the same query being run on a single machine. With regard to Pig, the biggest improvement was seen in Query #1 (Row count), where the three-node cluster had a 58.07% reduction in query run time compared to using a single machine. While a 3x improvement in performance isn't necessarily expected for each query, it would have been nice to see faster run times on the three-node cluster. The slower than expected run times can be due to any number of reasons, such as the types of instances being used, the order in which Hive and Pig decided to run the queries, etc. As far as comparing performance between Hive and Pig, it can be seen that Hive outperformed Pig on every query with regard to actual run time, regardless of using a single machine or a three-node cluster. This may be due to, among other reasons, the method by which Hive and Pig decided to internally optimize each query. Nevertheless, additional optimization techniques (such as continuing to adjust the number of mappers, reducers, combiners, etc.) could potentially improve run times for both Pig and Hive. It should also be noted, however, that by using a three-node cluster Pig actually had a greater reduction in run time for two out of three queries, compared with Hive.

Lastly, findings from the various queries were rather interesting. Medications treating heart burn, cholesterol and asthma (Nexium, Crestor and Advair, respectively) are some of the most widely prescribed drugs throughout the United States covered by Medicare Part D. Additionally, medical professionals in Internal Medicine and Family Practice prescribe substantially more drugs than any other specialty covered in this dataset. Needless to say, this dataset provides a wealth of information which can be used to improve medical treatment plans in the future and have a wide ranging effect on the overall state of public health in our nation. As an example, one such proposition may be that specialties which are known to prescribe the most medications should be advised to consider other holistic avenues or proactive approaches when treating illnesses.

REFERENCES

1. Medicare Provider Utilization and Payment Data: Part D Prescriber. April 30, 2015. http://www.cms.gov/Research-Statistics-Data-and-Systems/Statistics-Trends-and-Reports/Medicare-Provider-Charge-Data/Part-D-Prescriber.html

2. Centers for Medicare & Medicaid Services. Medicare Fee-For-Service Provider Utilization & Payment Data Part D Prescriber Public Use File: A Methodological Overview. April 7, 2015. http://www.cms.gov/Research-Statistics-Data-and-Systems/Statistics-Trends-and-Reports/Medicare-Provider-Charge-Data/Downloads/Prescriber_Methods.pdf

3. Confluence. Apache Pig User Documentation FAQ. November 05, 2013. https://cwiki.apache.org/confluence/display/PIG/FAQ

4. Grokbase. Apache Pig Forum. June 22, 2012. http://grokbase.com/t/pig/user/126qkv4s9f/how-can-i-set-the-mapper-number-for-pig-script

5. Pig Cookbook. The Apache Software Foundation. April 29, 2011. http://pig.apache.org/docs/r0.8.1/cookbook.html

6. A. Gates. *Programming Pig*. Ch. 7 – Developing and Testing Pig Latin Scripts. O'Reilly Media. 2011. http://chimera.labs.oreilly.com/books/1234000001811/ch07.html

7. Chuck Lam. *Hadoop in Action.* Manning Publications. 2010.

8. A. Rajaraman, J.D. Ullman. *Mining of Massive Datasets.* Cambridge University Press. 2011.

9. CSC 555 – In-Class Lecture Notes. DePaul University. Spring Quarter 2015.