The Cloud Computing Capstone Task 1 Report

Andrei Karpau

1.1 Data extracting and cleaning

In order to get the data new EBS volume in US East zone with data from snapshot was created. This volume had the original data from the snapshot which was packed into zip archives. In order to extract the data new Instance of size m3.xlarge with Ubuntu 14.04 preinstalled was created in EC2. For saving extracted data I created a separate EBS volume of size 80 GB. Both EBS volumes where mounted to the instance. The "On-Time Performance Data" from snapshot EBS was unpacked and copied to 80 GB EBS volume using default Ubuntu commands.

The next step was installing RSudio and RSudio Server on the m3 instance. After installing it, I could connect to AMI through port 8787 and work in RStudio in Browser. In RStudio I used "data.table" library for quickly loading ("fread" method) CSV files. Then redundant columns were removed and table was stored again into CSV file into separate folder on EBS. No improving of NA or empty values was made, since in different Hadoop tasks the way how empty values are processed can differ. This operation is done for all 240 CSV files. All files names are stored into input.txt file, which is put into the same folder. The overall data clearing process takes near 4-5 minutes on m3.xlarge instance.

1.2 Deploying the Hadoop cluster

In order to run all tasks Hadoop cluster was deployed. It consists of 4 xlarge instances. One instance is configured as a Namenode, the other as Datanodes. EBS volume was connected to the Namenode and all the data was copied into HDFS. All Hadoop tasks are run on this cluster. For persisting all the cluster settings after instances restart, all nodes are referenced by elastic IP addresses, which are used in Hadoop cluster settings.

All the jobs work with cleaned data set that consists of many CSV files stored in HDFS. The names of files are stored in input.txt, which is actual input file. Then files are added by FileInputFormat.addInputPath method. So, actually they are read simultaneously.

1.3 Tasks from group 1

In group 1 I have made tasks number 1 and 3.

Task 1.1 (top 10 popular airports) is solved by creating two map/reduce jobs. The first map job reads origin and destination of each flight and if it is non-empty it is written into context as a key with value one. Reduce tasks count the sum of 1 values and write pair key/sum further. The second Map store key/values into a tree set, sorted by "sum" and remove the pairs with smallest sums, when Tree set has more than 10 items. It writes key/values in Cleanup phase. It is made for decreasing the number of items, which come to the last Reduce task. The last reduce phase is configured to start only in one task (jobB.setNumReduceTasks(1)). It runs the similar logic as previous Map phase, but since it is started once for all pairs, it can do final filtering and sorting. The results are: ORD 12391670; ATL 11471021; DFW 10753753; LAX 7689745; PHX 6554246; DEN 6234590; DTW 5611320; IAH 5451844; MSP 5179113; SFO 5147708;

Task 1.3 (Rank the days of the week by on-time arrival performance) is solved in two map/reduce phases as well. First Map reads "day of week" and "arrival is delayed" columns and, if they are non-empty, writes them as a key and value respectively. Reduce phase sums the number of delays for each week day, counts the number of non-delayed flights and divides it by the overall number of flights. Then writes key/performance further. The last map/reduce phase works mainly the same way as in previous task. The main difference is that there is no need to remove items with worse values. The

results are: 6 0.8280189038524886; 2 0.8113310729453935; 1 0.8038023648822624; 7 0.8019268700316503; 3 0.7953223150873696; 4 0.7696807642629397; 5 0.7608263213222322;

1.4 Tasks from group 2

In group 2 I have made tasks number 1, 3 and 4. Since there was no task to directly write the results from Hadoop to Cassandra, the results of the task are stored into HDFS and then written to Cassandra that is installed on namenode (aw well as on other separate nodes without Hadoop) of Hadoop cluster. The same is done for task 3.2.

In tasks for group 2 in the input is an additional path to file, that contains X or XY values. This file is read in first map phase in setup method and is stored to the collection, which is saved in map object.

Task 2.1 (rank the top-10 carriers by on-time departure performance from X) is solved in two map/reduce phases. The basic idea was the same as in Task 1 from group 1. However, the main difference is in first map phase. The X value are read from file and stored to the list. The when map method is called, it verifies if the origin airport presents in the list and if yes, then the string "origin + '_' + airlineID" is used as an output key. The output value is the "departure delayed" integer. The reduce counts the performance for each origin/arilineID and the second map/reduce phase makes sorting and finds Top 10 airlines. Since there was not clarified if airlineId or airline code should be used in output, I used airline ID (in column description is written, that airline code can be different through the years). The results are (airport, airlineId, performance):

BWI,20436,0.9280719280719281 BWI,19704,0.8912584557067353 BWI,19805,0.8887653200181571
BWI,19386,0.8771306929759791 BWI,19790,0.8538328995124097 BWI,19977,0.8535197014736576
BWI,20378,0.85003885003885 BWI,19707,0.8488599348534202 BWI,20363,0.8488488488488488
BWI,20355,0.8466528156656206 CMI,20355,0.9599618684461392 CMI,20211,0.9224615384615384
CMI,20417,0.9066171923314781 CMI,19822,0.9047051816557474 CMI,20366,0.8623853211009175
CMI,20404,0.8346613545816733 CMI,20398,0.8327250231982679 IAH,19822,0.9179838950193856
IAH,19386,0.9103532210328327 IAH,19805,0.8954984302339305 IAH,20384,0.8933333333333333
IAH,19393,0.8835589592064401 IAH,20355,0.8786087566120706 IAH,20211,0.8737080917569953
IAH,19790,0.866933375487919 IAH,20304,0.863446680163616 IAH,20374,0.8553835024888745
LAX,19391,0.9165106289238903 LAX,19690,0.9090346534653465 LAX,20398,0.8962610706904482
LAX,20312,0.886721680420105 LAX,19386,0.8816886032351313 LAX,20304,0.881110059918007
LAX,20295,0.8642160540135033 LAX,19704,0.8633010901274374 LAX,19805,0.8595725937631609
LAX,20437,0.8567339149400218 MIA,20366,0.9118942731277533 MIA,20384,0.8998587449704648
MIA,20374,0.8989473684210526 MIA,19386,0.8914127786227085 MIA,20312,0.883623275344931
MIA,19977,0.8737098800403971 MIA,20355,0.8677958233184326 MIA,20295,0.8554095045500506
MIA,19822,0.8423695808137748 MIA,19704,0.8305413155884275 SFO,20312,0.8889293517844137
SFO,19391,0.8764763779527559 SFO,20384,0.8725905030559473 SFO,19690,0.8722098214285714
SFO,19790,0.8718591568430087 SFO,19386,0.8665762148085739 SFO,20404,0.8592750533049041
SFO,19805,0.8565174146270966 SFO,19704,0.8530292151176344 SFO,20398,0.840341982304404

Task 2.3 (for each source-destination pair X-Y, rank the top-10 carriers by on-time arrival performance) is also solved in two map/reduce phases. The main difference from previous solution is in filtering by origin and destination in first map phase, as well as in key value pair which is written by this map. It writes a combination of Origin, Destination and AirlineId strings as a key and "arrival delayed" as a value. The reduce counts on time arrival performance for each origin-destination flight of concrete airline. The last map/reduce phase sorts and writes to 10 carriers for each origin-destination. The results are (origin, destination, airlineId, performance):

ATL,PHX,20437,0.7650303364589078 ATL,PHX,20355,0.7539254170755643
ATL,PHX,19991,0.740650406504065 ATL,PHX,19707,0.736734693877551
ATL,PHX,19790,0.7010516198337762 CMI,ORD,20398,0.7732948442534908
DFW,IAH,20384,0.8947368421052632 DFW,IAH,19704,0.8524019126634789
DFW,IAH,20304,0.8256029684601113 DFW,IAH,19977,0.8224852071005917
DFW,IAH,20366,0.8192513368983957 DFW,IAH,20374,0.8184926627724747
DFW,IAH,19790,0.8139865104721334 DFW,IAH,19805,0.796449530516432

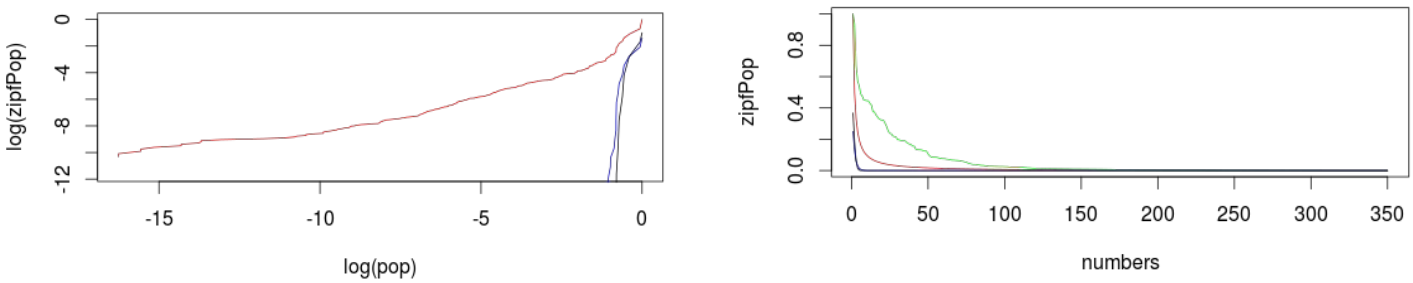
DFW,IAH,20398,0.7413990825688074 IND,CMH,19805,1.0 IND,CMH,19704,0.9300602130616026
IND,CMH,19991,0.8580392156862745 IND,CMH,20355,0.8534087846941525
IND,CMH,19386,0.8276923076923077 IND,CMH,19790,0.7932692307692307
IND,CMH,19707,0.7710280373831776 JFK,LAX,19977,0.7862573380810317
JFK,LAX,19805,0.738291966035271 JFK,LAX,19991,0.7318611987381703
JFK,LAX,19790,0.7042831495255194 JFK,LAX,20211,0.6675928264713311
JFK,LAX,20384,0.6483931947069943 LAX,SFO,20312,0.9047619047619048
LAX,SFO,20436,0.8725817211474316 LAX,SFO,19391,0.8718094157685763
LAX,SFO,20366,0.797427652733119 LAX,SFO,19805,0.7830426008348769
LAX,SFO,20355,0.7824853837541983 LAX,SFO,20398,0.7781885397412199
LAX,SFO,19704,0.7704347826086957 LAX,SFO,19977,0.7566439518032959
LAX,SFO,19393,0.7537890504175688

Task 2.4 (For each source-destination pair X-Y, determine the mean arrival delay (in minutes) for a flight from X to Y). Since sorting is not needed here, this task was implemented using one map/reduce phase. Map task reads the input values of origin and destination in setup method and saves them into collection. In map method it filters the flights that are from X to Y. Then wrights origin destination combination as a key and arrival delay in minutes as a value. In reduce phase it simply counts the mean for each origin destination combination. The results are (origin, destination, mean):

ATL,PHX,13.635306900298364; CMI,ORD,15.881176154672396; DFW,IAH,11.064503269490507; IND,CMH,6.613487475915222; JFK,LAX,14.663494193115593; LAX,SFO,13.708278287160274

1.4 Tasks from group 3

Task 3.1 (Zipf distribution). For investigating Zipf distribution, it is needed to find the popularity of each airport and then investigate the results. For counting the popularity of airports, the similar map/reduce algorithm as in task 1.1 was used. However, in this case, in second map reduce phase all the airports with their popularities were sorted and written into the output file. And then investigated in R Studio. On the left log log plot in red is Zipf distribution/airports popularity. It fits much better than other similar distributions (poisson, geometric – in black and blue). The same is on the right plot, where airports distribution is in green and Zipf in red. So, to my mind, it can be considered, that airports population fits Zipf distribution.



Task 3.2 (Tom's travel). This task is completed in one map/reduce phase. Mapper reads the input XYZ values in setup method and saves them into collection. For each XYZ it makes two collection instances, which contain sample number (e.g. 1 for first XY and ZY, 2 for second XY and YZ etc.), origin, destination, date and flag that shows if it is the first or second part of the trip. In map task each income flight is combined with all the collection items and if it has right origin, destination, date and time values, then it is written further (key is sample number, value – all needed information about the flight). Since, the key value is unique for flights for the same XYZ sample, the reduce task gets all the needed information to find out the best flights (by arrival time) for first and second trip part. Then reduce writes date, origin, destination, arrival time and flight id for both trip parts. The results are: 2008-03-04,CMI,ORD,659,4374,2008-03-06,ORD,LAX,1428,1407

2008-09-09,JAX,DFW,856,845,2008-09-11,DFW,CRP,1428,3701

2008-04-01,SLC,BFL,1205,3755,2008-04-03,BFL,LAX,1551,5429

2008-07-12,LAX,SFO,752,3534,2008-07-14,SFO,PHX,1426,1550

2008-06-10,DFW,ORD,824,2320,2008-06-12,ORD,DFW,114,2355

2008-01-01,LAX,ORD,836,1740,2008-01-03,ORD,JFK,1515,908

1.5 Writing to Cassandra

For storing values in NoSQL database, separate Cassandra cluster was deployed. This cluster consists of 2 t2.micro instances and one xlarge seed instance, that is namenode used in Hadoop. After running the Hadoop jobs, the result files were copied from HDFS to local storage and then were stored into Cassandra using COPY FROM command.