



CONGESTION IN THE SKY

INFO7250 Engineering of Big-Data Systems
Final Project Report

Kinnar Rajeshkumar Kansara

NUID: 001388213

Contents

1.	Overview	3
2.	Dataset Details	4
2.1.	Variable Descriptions	4
3.	Analysis of Flights using Hadoop MapReduce	5
3.1.	Getting top 10 source destination airport pairs	5
3.2.	Unique Carriers Names with Flights Count	5
3.3.	Year wise flight delay (> 15 minutes) and cancellation. Counts and Ratio	6
3.4.	Flight delay and cancellation by carrier including carrier names	6
3.5.	Date wise flight delay (> 15 minutes) and cancellation. Counts and Ratio	7
3.6.	Average distance covered and airtime done by each carrier	9
3.7.	Recommendation System using RMS (Root Mean Square).....	10
4.	Analysis of flights using Pig on Hadoop	12
4.1.	Top 10 cities by total traffic of flights	12
4.2.	Popular carriers.....	15
4.3.	Most busy routes (descending order).....	16
5.	Analysis on Flights using Hive on Hadoop.....	17
5.1.	Create FlightSchema and setup the parameters	17
5.2.	Create table flights and load data from hdfs path.....	17
5.3.	Create table airports and load data from hdfs path	18
5.4.	Create table carriers and load data from hdfs path	18
5.5.	Find flights which travelled more than 500 airmiles.....	18
5.6.	Find flights which arrive and depart on time	18
5.7.	Get count of flights for each carrier.....	19
5.8.	Find origin and destination pairs from DEN airport from 2008 data.....	20
6.	Graphical Representation and Analysis	22
6.1.	% of flight departures delayed > 15 minutes – daily basis	22
6.2.	% of flight cancelled – daily basis.....	24
6.3.	Flight trend on daily basis	25
6.4.	Delayed flights trend on daily basis	27
6.5.	Flight delay history by year	27
6.6.	Total flights vs delayed flights for each carrier	28
6.7.	Tabular view of top 10 most popular source destination pairs.....	28

CONGESTION IN THE SKY

7.	Lessons Learned & tips for travellers.....	29
8.	Challenges	29
9.	Future Scope	29
10.	References	29
11.	Appendix	30
11.1.	Getting top 10 source destination airport pairs	30
11.2.	Unique Carriers Names with Flights Count	35
11.3.	Year wise flight delay (> 15 minutes) and cancellation. Counts and Ratio	41
11.4.	Flight delay and cancellation by carrier including carrier names	46
11.5.	Date wise flight delay (> 15 minutes) and cancellation. Counts and Ratio	53
11.6.	Average distance covered and airtime done by each carrier	59
11.7.	Recommendation System using RMS (Root Mean Square).....	65

1. Overview

The dataset I have chosen is Flight Arrival Data from stat-computing.org (<http://stat-computing.org/dataexpo/2009/the-data.html>)

The dataset has various columns as described in the following sections. The main reason to select this dataset is the number of columns, data distribution by years and other small datasets which give information about carriers, airports and aircrafts. This will help me in performing MapReduce in efficient manner with the help of Java MapReduce, Hive and Pig.

In this project, the data I am using is of 10 years which is 1999-2008. Due to local disk space constraints, I avoided the previous 10 years' data. By going through the csv pattern, I observed that I can use 1999-2008 data for proper analysis. All csv files combined makes up 12.6 GB of disk space.

Project Code Repository URL: <https://github.com/kinnarrk/INFO7250BigData>

2. Dataset Details

2.1. Variable Descriptions

#	Name	Description
1	Year	1987-2008
2	Month	1-12
3	DayofMonth	1-31
4	DayOfWeek	1 (Monday) - 7 (Sunday)
5	DepTime	actual departure time (local, hhmm)
6	CRSDepTime	scheduled departure time (local, hhmm)
7	ArrTime	actual arrival time (local, hhmm)
8	CRSArrTime	scheduled arrival time (local, hhmm)
9	UniqueCarrier	<u>unique carrier code</u>
10	FlightNum	flight number
11	TailNum	plane tail number
12	ActualElapsedTime	in minutes
13	CRSElapsedTime	in minutes
14	AirTime	in minutes
15	ArrDelay	arrival delay, in minutes
16	DepDelay	departure delay, in minutes
17	Origin	origin <u>IATA airport code</u>
18	Dest	destination <u>IATA airport code</u>
19	Distance	in miles
20	TaxiIn	taxi in time, in minutes
21	TaxiOut	taxi out time in minutes
22	Cancelled	was the flight cancelled?
23	CancellationCode	reason for cancellation (A = carrier, B = weather, C = NAS, D = security)
24	Diverted	1 = yes, 0 = no
25	CarrierDelay	in minutes
26	WeatherDelay	in minutes
27	NASDelay	in minutes
28	SecurityDelay	in minutes
29	LateAircraftDelay	in minutes

3. Analysis of Flights using Hadoop MapReduce

3.1. Getting top 10 source destination airport pairs

This analysis gives us the basic understanding about busiest source and destination pairs. This gives us the idea about how flight booking trend is and if we want to avoid rush by choosing alternate source or destination airport. Job chaining has been carried out to get this information.

```
hadoop jar ~/Desktop/proj_jars/finalproject.jar  
com.kinnar.bigdataproyect.top10_busy_airports.TopNApp  
/project/input/years /project/Top10SourceDestinations
```

```
kinnar@ubuntu:~$ hadoop fs -cat /project/Top10SourceDestinations/part-r-00000  
2019-12-13 09:31:17,308 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrus  
ted = false, remoteHostTrusted = false  
LAX-LAS 137997  
LAS-LAX 135016  
PHX-LAX 123517  
LAX-PHX 121631  
SFO-LAX 117850  
LAX-SFO 116318  
LAS-PHX 116273  
ORD-MSP 114370  
PHX-LAS 114233  
MSP-ORD 113753  
kinnar@ubuntu:~$
```

3.2. Unique Carriers Names with Flights Count

By doing this, we can get all the carriers with their names from different file which is carriers.csv. I have performed reducer side join to achieve this. This is divided in two parts: first get the flight count of each carrier and then perform join to get the carrier names.

```
hadoop jar ~/Desktop/proj_jars/finalproject.jar  
com.kinnar.bigdataproyect.unique_carrier_names.CarriersApp  
/project/input/years /project
```

```
kinnar@ubuntu:~$ hadoop fs -cat /project/UniqueCarriersWithNames/part-r-00000  
2019-12-13 02:16:34,022 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrus  
ted = false, remoteHostTrusted = false  
Pinnacle Airlines Inc. 521059  
American Airlines Inc. 7011039  
Aloha Airlines Inc. 154381  
Alaska Airlines Inc. 1583287  
JetBlue Airways 811341  
Continental Air Lines Inc. 3325304  
Independence Air 693047  
Delta Air Lines Inc. 6826616  
Atlantic Southeast Airlines 1697172  
Frontier Airlines Inc. 336958  
AirTran Airways Corporation 1265138  
Hawaiian Airlines Inc. 274265  
America West Airlines Inc. (Merged with US Airways 9/05. Stopped reporting 10/07.) 1418057  
American Eagle Airlines Inc. 3954895  
Northwest Airlines Inc. 4817229  
Comair Inc. 1464176  
Skywest Airlines Inc. 3090853  
Trans World Airways LLC 787682  
ATA Airlines d/b/a ATA 208420  
United Air Lines Inc. 5869005  
US Airways Inc. (Merged with America West 9/05. Reporting for both starting 10/07.) 5364830  
Southwest Airlines Co. 10134222  
Expressjet Airlines Inc. 2350309  
Mesa Airlines Inc. 854056
```

3.3. Year wise flight delay (> 15 minutes) and cancellation. Counts and Ratio

This will give us the idea about the historical data. It gives us information that which years were good and worse for the travelling. By combining weather conditions, we can do better analytics

```
hadoop jar ~/Desktop/proj_jars/finalproject.jar  
com.kinnar.bigdatapproject.yearly_delay.YearlyDelayApp  
/project/input/years /project/YearlyDelayCancel
```

The fields contained in the value part are: flightsCount, delayedFlightsCount, delayPercentage, canceledFlightsCount, canceledPercentage. The labels are removed for the sake of easy extraction while doing graphical representation.

```
kinnar@ubuntu:~$ hadoop fs -cat /project/YearlyDelayCancel/part-r-00000  
2019-12-13 05:24:00,353 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrus  
ted = false, remoteHostTrusted = false  
1999      5527884,1101355,19.92,154311,2.79  
2000      5683047,1301615,22.90,187490,3.30  
2001      5967780,1053819,17.66,231198,3.87  
2002      5271359,823147,15.62,65143,1.24  
2003      6488540,1005631,15.50,101469,1.56  
2004      7129270,1355988,19.02,127757,1.79  
2005      7140596,1399557,19.60,133730,1.87  
2006      7141922,1548755,21.69,121934,1.71  
2007      7453215,1734629,23.27,160748,2.16  
2008      7009728,1466191,20.92,137434,1.96
```

3.4. Flight delay and cancellation by carrier including carrier names

This is important to analyse which carrier creates more delay and gives us important information about choosing the flights which are on time. For this analysis, showing carrier name is important so I have carried out reducer side join. Also, job chaining is needed for the analysis so after chaining, join has been carried out to show carrier information.

```
hadoop jar ~/Desktop/proj_jars/finalproject.jar  
com.kinnar.bigdatapproject.carrier_delay_cancel.CarrierDelayCancelApp  
/project/input/years /project/input/carriers.csv  
/project/CarriersDelayCancel
```

CONGESTION IN THE SKY

```

kinnar@ubuntu:~$ hadoop fs -cat /project/CarriersDelayCancel/part-r-00000
2019-12-13 05:51:50,873 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
Pinnacle Airlines Inc. 521059,94432,18.12,15039,2.89
American Airlines Inc. 7011039,1433595,20.45,180213,2.57
Aloha Airlines Inc. 154381,13195,8.55,2837,1.84
Alaska Airlines Inc. 1583287,354150,22.37,39163,2.47
JetBlue Airways 811341,191762,23.64,9281,1.14
Continental Air Lines Inc. 3325304,656452,19.74,43161,1.30
Independence Air 693047,141765,20.46,22176,3.20
Delta Air Lines Inc. 6826616,1297123,19.00,150623,2.21
Atlantic Southeast Airlines 1697172,418887,24.68,48676,2.87
Frontier Airlines Inc. 336958,62934,18.68,1778,0.53
AirTran Airways Corporation 1265138,281657,22.26,12854,1.02
Hawaiian Airlines Inc. 274265,16706,6.09,1329,0.48
America West Airlines Inc. (Merged with US Airways 9/05. Stopped reporting 10/07.) 1418057,2970
12,20.94,30564,2.16
American Eagle Airlines Inc. 3954895,842571,21.30,157478,3.98
Northwest Airlines Inc. 4817229,920197,19.10,87289,1.81
Comair Inc. 1464176,304364,20.79,47174,3.22
Skywest Airlines Inc. 3090853,517173,16.73,65390,2.12
Trans World Airways LLC 787682,138614,17.60,16010,2.03
ATA Airlines d/b/a ATA 208420,39135,18.78,2307,1.11
United Air Lines Inc. 5869005,1270813,21.65,162133,2.76
US Airways Inc. (Merged with America West 9/05. Reporting for both starting 10/07.) 5364830,1078
025,20.09,139210,2.59
Southwest Airlines Co. 10134222,1727443,17.05,104488,1.03
Expressjet Airlines Inc. 2350309,502089,21.36,51991,2.21
Mesa Airlines Inc. 854056,190593,22.32,30050,3.52

```

3.5. Date wise flight delay (> 15 minutes) and cancellation. Counts and Ratio

This MapReduce is one of the most important part of the analysis. For graphical representation, this was the key to see so many patterns in the flight delay and cancellation. This gives us clear picture about the historical data and how we can take a good decision about when to travel and what time should be avoided. Although, this MapReduce is not so difficult to carry out, but the data it can represent is very important. Graphical representation will be explained in the later part of the report.

```

hadoop jar ~/Desktop/proj_jars/finalproject.jar
com.kinnar.bigdataproyect.daily_delay_cancel.DailyDelayCancelApp
/project/input/years /project/DailyDelayCancel

```

Same as year wise details, the fields contained in the value part are: flightsCount, delayedFlightsCount, delayPercentage, canceledFlightsCount, canceledPercentage. The labels are removed for the sake of easy extraction while doing graphical representation.

CONGESTION IN THE SKY

```
kinnar@ubuntu:~$ hadoop fs -cat /project/DailyDelayCancel/part-r-00000
2019-12-12 23:39:47,068 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
19990101      13038,3326,25.51,358,2.75
19990102      13308,4455,33.48,3059,22.99
19990103      14520,6803,46.85,2199,15.14
19990104      15267,6804,44.57,961,6.29
19990105      15205,5273,34.68,617,4.06
19990106      15213,5448,35.81,1036,6.81
19990107      15223,4318,28.36,746,4.90
19990108      15245,6050,39.69,2281,14.96
19990109      12928,3589,27.76,825,6.38
19990110      14244,3029,21.27,259,1.82
19990111      15194,3340,21.98,1036,6.82
19990112      15168,2699,17.79,612,4.03
19990113      15179,4048,26.67,1058,6.97
19990114      15210,4677,30.75,1921,12.63
19990115      15240,5216,34.23,1508,9.90
19990116      12901,1718,13.32,320,2.48
19990117      14229,3008,21.14,374,2.63
19990118      15220,4158,27.32,685,4.50
19990119      15186,1648,10.85,468,3.08
19990120      15194,3443,22.66,591,3.89
19990121      15225,4604,30.24,570,3.74
19990122      15257,4927,32.29,1172,7.68
19990123      12904,3180,24.64,1078,8.35
19990124      14227,2554,17.95,353,2.48
19990125      15224,1901,12.49,378,2.48
19990126      15182,1889,12.44,454,2.99
19990127      15201,1895,12.47,310,2.04
19990128      15235,2952,19.38,425,2.79
19990129      15261,2916,19.11,289,1.89
19990130      12900,1548,12.00,274,2.12
19990131      14286,2465,17.25,326,2.28
19990201      15285,3280,21.46,616,4.03
```

CONGESTION IN THE SKY

```

kinnar@ubuntu: ~
File Edit View Search Terminal Help
20081123 17344,1389,11.72,83,0.48
20081124 18427,2138,11.60,162,0.88
20081125 18638,2311,12.40,106,0.57
20081126 18855,2013,10.68,45,0.24
20081127 12008,892,7.43,28,0.23
20081128 14328,967,6.75,37,0.26
20081129 16654,3322,19.95,87,0.52
20081130 18619,9273,49.80,463,2.49
20081201 18877,8960,47.47,464,2.46
20081202 18166,3171,17.46,141,0.78
20081203 18019,2947,16.35,263,1.46
20081204 18350,3516,19.16,173,0.94
20081205 18405,2527,13.73,161,0.87
20081206 14440,2074,14.36,68,0.47
20081207 17116,3209,18.75,132,0.77
20081208 18301,2642,14.44,276,1.51
20081209 17600,3976,22.59,502,2.85
20081210 17917,4979,27.79,675,3.77
20081211 18377,4766,25.93,682,3.71
20081212 18433,3832,20.79,297,1.61
20081213 14459,2197,15.19,126,0.87
20081214 17137,4168,24.32,206,1.20
20081215 18317,6459,35.26,578,3.16
20081216 17631,6940,39.36,1020,5.79
20081217 18294,7721,42.21,883,4.83
20081218 18746,7412,39.54,975,5.20
20081219 18828,8482,45.05,2089,11.10
20081220 16216,7737,47.71,741,4.57
20081221 17725,8654,48.82,1774,10.01
20081222 18665,8306,44.50,910,4.88
20081223 18633,8696,46.67,868,4.66
20081224 15263,6745,44.19,298,1.95
20081225 15288,3374,22.07,214,1.40
20081226 18640,7155,38.39,1220,6.55
20081227 16593,7944,47.88,788,4.75
20081228 17696,4730,26.73,278,1.57
20081229 18540,2749,14.83,138,0.74
20081230 18538,3689,19.90,285,1.54
20081231 15748,3634,23.08,554,3.52
kinnar@ubuntu:~$ hadoop fs -cat /project/Top10SourceDestinations/part-r-00000

```

3.6. Average distance covered and airtime done by each carrier

This analysis is useful for viewing carrier reach and capacity. We can segregate the carriers by their ranks using this analysis. This information combined with delay and cancellation details gives us the overall reliability and quality of the airline service.

```

hadoop jar ~/Desktop/proj_jars/finalproject.jar
com.kinnar.bigdataproyect.avg_dist_carrier.AverageMain
/project/input/years /project/AvgDistAirtime

```

CONGESTION IN THE SKY

```

kinnar@ubuntu: /usr/local/bin/hadoop-3.2.1/sbin$
File Edit View Search Terminal Help
Peak Reduce Physical memory (bytes)=227459072
Peak Reduce Virtual memory (bytes)=5322620928
Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
Bytes Read=6286172299
File Output Format Counters
Bytes Written=3551
kinnar@ubuntu: /usr/local/bin/hadoop-3.2.1/sbin$ hadoop fs -cat /project/AvgDistAirTime/part-r-00000
2019-12-14 10:05:37,961 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localhostTrusted = false, remoteHostTrusted = false
9E AverageCountTuple{Total Flights=584652, Total Distance=22703482, Total Air Time=754313474, Average Distance=451.21, Average Air Time=1494.72}
AA AverageCountTuple{Total Flights=6812019, Total Distance=1466992226, Total Air Time=1700810330, Average Distance=215.35, Average Air Time=249.68}
AQ AverageCountTuple{Total Flights=151507, Total Distance=51227324, Total Air Time=207101837, Average Distance=338.12, Average Air Time=1366.95}
AS AverageCountTuple{Total Flights=1539089, Total Distance=1290250071, Total Air Time=2018484637, Average Distance=838.32, Average Air Time=1311.48}
B6 AverageCountTuple{Total Flights=799333, Total Distance=95055615, Total Air Time=1151338297, Average Distance=1196.07, Average Air Time=1440.37}
CO AverageCountTuple{Total Flights=3273935, Total Distance=793014514, Total Air Time=657647574, Average Distance=242.22, Average Air Time=200.87}
DH AverageCountTuple{Total Flights=669687, Total Distance=251483885, Total Air Time=985613333, Average Distance=375.52, Average Air Time=1471.75}
DL AverageCountTuple{Total Flights=6662775, Total Distance=1136390051, Total Air Time=1446709282, Average Distance=170.56, Average Air Time=217.13}
EV AverageCountTuple{Total Flights=1645211, Total Distance=742277300, Total Air Time=1825235102, Average Distance=451.17, Average Air Time=1109.42}
F9 AverageCountTuple{Total Flights=334857, Total Distance=297757070, Total Air Time=509777308, Average Distance=889.21, Average Air Time=1522.37}
FL AverageCountTuple{Total Flights=1249409, Total Distance=833241928, Total Air Time=1856768383, Average Distance=666.91, Average Air Time=1486.12}
HA AverageCountTuple{Total Flights=272880, Total Distance=161404032, Total Air Time=381096796, Average Distance=591.48, Average Air Time=1396.57}
HP AverageCountTuple{Total Flights=1385572, Total Distance=1307302755, Total Air Time=2020988458, Average Distance=943.51, Average Air Time=1458.60}
MQ AverageCountTuple{Total Flights=3790050, Total Distance=1388700927, Total Air Time=1204517607, Average Distance=366.41, Average Air Time=317.81}
NW AverageCountTuple{Total Flights=4719161, Total Distance=755915091, Total Air Time=1520335962, Average Distance=160.18, Average Air Time=322.16}
OH AverageCountTuple{Total Flights=1414180, Total Distance=664044565, Total Air Time=2083770904, Average Distance=469.56, Average Air Time=1473.48}
OO AverageCountTuple{Total Flights=3020777, Total Distance=117518881, Total Air Time=143697020, Average Distance=389.14, Average Air Time=47.57}
TW AverageCountTuple{Total Flights=770098, Total Distance=603216587, Total Air Time=1142595909, Average Distance=783.30, Average Air Time=1483.70}
TZ AverageCountTuple{Total Flights=206007, Total Distance=237096582, Total Air Time=304677183, Average Distance=1150.92, Average Air Time=1478.97}
UA AverageCountTuple{Total Flights=5695307, Total Distance=1489642256, Total Air Time=107909057, Average Distance=261.56, Average Air Time=18.95}
US AverageCountTuple{Total Flights=5214046, Total Distance=717359727, Total Air Time=779580447, Average Distance=137.58, Average Air Time=149.52}
WN AverageCountTuple{Total Flights=10012641, Total Distance=1407230291, Total Air Time=2109730442, Average Distance=140.55, Average Air Time=210.71}
XE AverageCountTuple{Total Flights=2290684, Total Distance=1232713411, Total Air Time=900268480, Average Distance=538.14, Average Air Time=393.01}
YV AverageCountTuple{Total Flights=822403, Total Distance=327982123, Total Air Time=1201040057, Average Distance=398.81, Average Air Time=1460.40}
kinnar@ubuntu: /usr/local/bin/hadoop-3.2.1/sbin$

```

3.7. Recommendation System using RMS (Root Mean Square)

This process is divided in two parts: First we need to calculate average of arrival and departure delay for each source destination pair for each carrier. After that, second part is to calculate RMS value for average of arrival and departure delay.

Once this is done, we have to sort the result in ascending order by the RMS value. This will give us the recommendation for choosing carrier for source and destination pair.

```

hadoop jar ~/Desktop/proj_jars/finalproject.jar
com.kinnar.bigdataportect.rms_carrier.RMSMain /project/input/years
/project/FlightRMS

```

```

kinnar@ubuntu: ~/Documents/project/INFO7200BigData
File Edit View Search Terminal Help
YUM-SLC OO 5.7659
kinnar@ubuntu:~/Documents/project/INFO7200BigData$ hadoop fs -head /project/FlightRMS/part-r-00000
2019-12-13 15:06:50,127 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localhostTrus
ted = false, remoteHostTrusted = false
ABE-ATL DL 10.8032
ABE-ATL EV 19.9292
ABE-ATL OH 15.3449
ABE-AVP EV 0.0000
ABE-AVP OH 23.7539
ABE-AZO OO 0.0000
ABE-BHM OO 11.4018
ABE-BWI OH 100.6032
ABE-CLE XE 4.2448
ABE-CLT US 1.3987
ABE-CLT YV 8.6536
ABE-CVG EV 8.5775
ABE-CVG OH 2.8560
ABE-DTW 9E 9.6700

```

Getting RMS values

```

hadoop jar ~/Desktop/proj_jars/finalproject.jar
com.kinnar.bigdataportect.recommendation_sys.SecondarySortDriver
/project/FlightRMS /project/FlightRecommendation

```

CONGESTION IN THE SKY

```
kinnar@ubuntu: ~/Documents/project/INFO7200BigData
File Edit View Search Terminal Help
ABQ-DEN OO      7.7549
kinnar@ubuntu:~/Documents/project/INFO7200BigData$ hadoop fs -head /project/FlightRecommendation/part-r-00000
2019-12-13 15:07:44,442 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localhostTrusted = false, remoteHostTrusted = false
ABE-ATL : DL      10.8032
ABE-ATL : OH      15.3449
ABE-ATL : EV      19.9292
ABE-AVP : EV       0.0000
ABE-AVP : OH      23.7539
ABE-AZO : OO       0.0000
ABE-BHM : OO      11.4018
ABE-BWI : OH     100.6032
ABE-CLE : XE       4.2448
ABE-CLT : US       1.3987
ABE-CLT : YV       8.6536
ABE-CVG : OH       2.8560
ABE-CVG : EV       8.5775
ABE-DTW : NW       2.5224
```

Get the recommended carrier for source-destination pair

4. Analysis of flights using Pig on Hadoop

4.1. Top 10 cities by total traffic of flights

Here, this analysis would be very difficult to achieve by Java MapReduce. Pig makes it really easy and less time consuming. It also executes really fast and it can leverage the facility to run on the local mode using `-x local` switch.

This analysis gives us the information about the busies cities by traffic. For each airport, we are finding number of inbound, outbound and all flights.

CONGESTION IN THE SKY

```

Documents ▾ Open [icon] 1top10airportsbytraffic.pig Save [icon] [icon] [icon]
~/Documents/project/INFO7200BigData/pig_scripts

1top10airportsby... x
-- Top 10 airports by traffic - only year 2008 data
-- First, load the data
RAW_DATA = LOAD '/home/kinnar/Desktop/proj_dataset/2008.csv' USING
PigStorage(',') AS
    (year: int, month: int, day: int, dow: int,
    dtime: int, sdttime: int, arftime: int, satime: int,
    carrier: chararray, fn: int, tn: chararray,
    etime: int, setime: int, airtime: int,
    adelay: int, ddelay: int,
    scode: chararray, dcode: chararray, dist: int,
    tintime: int, touttime: int,
    cancel: chararray, cancelcode: chararray, diverted: int,
    cdelay: int, wdelay: int, ndelay: int, sdelay: int, latedelay: int);

-- INBOUND TRAFFIC, PER MONTH, TOP 10
-- projection for only getting useful fields: only month and destination ID
INBOUND = FOREACH RAW_DATA GENERATE month AS m, dcode AS d;
-- group by month, then sort ID
GROUP_INBOUND = GROUP INBOUND BY (m,d);
-- aggregate and flatten group so that output relation has 3 fields
COUNT_INBOUND = FOREACH GROUP_INBOUND GENERATE FLATTEN(group),
COUNT(INBOUND) AS count;
-- aggregate over months
GROUP_COUNT_INBOUND = GROUP COUNT_INBOUND BY m;
-- apply UDF to compute top k (k=10)
topMonthlyInbound = FOREACH GROUP_COUNT_INBOUND {
    result = TOP(10, 2, COUNT_INBOUND);
    GENERATE FLATTEN(result);
}

-- dump topMonthlyInbound
STORE topMonthlyInbound INTO '/home/kinnar/Documents/project/INFO7200BigData/
-- output (date, inbound, month, destination ID)

```

CONGESTION IN THE SKY

```

1top10airportsby... x STORE topMonthlyInbound INTO '/home/kinnar/Documents/project/INF07200BigData/
pig_output/1top_inbound' USING PigStorage(',');

-- dump topMonthlyInbound

-- OUTBOUND TRAFFIC, PER MONTH, TOP 10 - same as above
OUTBOUND = FOREACH RAW_DATA GENERATE month AS m, scode AS s;
GROUP_OUTBOUND = GROUP OUTBOUND BY (m,s);
COUNT_OUTBOUND = FOREACH GROUP_OUTBOUND GENERATE FLATTEN(group),
COUNT(OUTBOUND) AS count;
GROUP_COUNT_OUTBOUND = GROUP COUNT_OUTBOUND BY m;
topMonthlyOutbound = FOREACH GROUP_COUNT_OUTBOUND {
    result = TOP(10, 2, COUNT_OUTBOUND);
    GENERATE FLATTEN(result);
}

-- dump topMonthlyOutbound
STORE topMonthlyOutbound INTO '/home/kinnar/Documents/project/
INF07200BigData/pig_output/1top_outbound' USING PigStorage(',');

-- TOTAL TRAFFIC, PER MONTH, TOP 10
UNION_TRAFFIC = UNION COUNT_INBOUND, COUNT_OUTBOUND;
GROUP_UNION_TRAFFIC = GROUP UNION_TRAFFIC BY (m,d);
TOTAL_TRAFFIC = FOREACH GROUP_UNION_TRAFFIC GENERATE FLATTEN(group) AS
(m,code), SUM(UNION_TRAFFIC.count) AS total;
TOTAL_MONTHLY = GROUP TOTAL_TRAFFIC BY m;

topMonthlyTraffic = FOREACH TOTAL_MONTHLY {
    result = TOP(10, 2, TOTAL_TRAFFIC);
    GENERATE FLATTEN(result) AS (month, iata, traffic);
}

STORE topMonthlyTraffic INTO '/home/kinnar/Documents/project/INF07200BigData/
pig_output/1monthly-top-traffic/' USING PigStorage(',');

explain -brief -dot -out ./ topMonthlyTraffic
Pig Tab Width: 8 Ln 60, Col 90 INS

```

Output:

Top 10 inbound

```

part-r-00000 x 1, SAN, 2989
1, MCO, 3461
1, DAL, 3861
1, LAX, 3553
1, BWI, 4988
1, HOU, 4188
1, OAK, 4159
1, PHX, 5820
1, LAS, 7295
1, MDW, 6590

```

Top 10 outbound

CONGESTION IN THE SKY

```

Documents  Open  [icon]  part-r-00000
~/Documents/project/INFO7200BigData/pig_output/1top_inbound  Save  [icon] [icon] [icon] [icon]

part-r-00000  x  1,SAN,1502
                  1,MCO,1735
                  1,DAL,1931
                  1,LAX,1774
                  1,BWI,2504
                  1,HOU,2101
                  1,OAK,2086
                  1,PHX,2920
                  1,LAS,3530
                  1,MDW,3306

```

Top 10 monthly

```

Documents  Open  [icon]  part-r-00000
~/Documents/project/INFO7200BigData/pig_output/1top_outbound  Save  [icon] [icon] [icon] [icon]

part-r-00000  x  1,SAN,1487
                  1,MCO,1726
                  1,DAL,1930
                  1,LAX,1779
                  1,BWI,2484
                  1,HOU,2087
                  1,OAK,2073
                  1,PHX,2900
                  1,LAS,3765
                  1,MDW,3284

```

4.2. Popular carriers

Let's calculate the volume of each carrier by total flights of a year. Carrier ranking is carried out by their median volume value over 10 years' span.

```

Documents  Open  [icon]  2popularcarriers.pig
~/Documents/project/INFO7200BigData/pig_scripts  Save  [icon] [icon] [icon] [icon]

2popularcarriers....  x  -- Carrier popularity
                        -- First, we load the raw data from the dataset - year 2008
RAW_DATA = LOAD '/home/kinnar/Desktop/proj_dataset/2008.csv' USING
PigStorage(',') AS
  (year: int, month: int, day: int, dow: int,
   dtime: int, sdttime: int, arrtime: int, satime: int,
   carrier: chararray, fn: int, tn: chararray,
   etime: int, setime: int, airtime: int,
   adelay: int, ddelay: int,
   scode: chararray, dcode: chararray, dist: int,
   tintime: int, touttime: int,
   cancel: chararray, cancelcode: chararray, diverted: int,
   cdelay: int, wdelay: int, ndelay: int, sdelay: int, latedelay: int);

CARRIER_DATA = FOREACH RAW_DATA GENERATE month AS m, carrier AS cname;
GROUP_CARRIERS = GROUP CARRIER_DATA BY (m,cname);
COUNT_CARRIERS = FOREACH GROUP_CARRIERS GENERATE FLATTEN(group),
(COUNT(CARRIER_DATA)) AS popularity;

STORE COUNT_CARRIERS INTO '/home/kinnar/Documents/project/INFO7200BigData/
pig_output/2top_carriers' USING PigStorage(',');
--dump top_carriers

```

Output: The data output is very less because I took 2008 year's data. It would give better result if I use 10 years data.

```

Documents  Open  [icon]  part-r-00000
~/Documents/project/INFO7200BigData/pig_output/2top_carriers  Save  [icon] [icon] [icon] [icon]

part-r-00000  x  1,NK,2
                  1,NW,2
                  1,WN,49995

```


4.3. Most busy routes (descending order)

In this part of analysis, the idea is to create a frequency table for unordered pair (m, n) in which they are the unique pairs of airport codes and it will give us one of the most important information that which airport pairs are busiest.



```
-- Most busy routes
-- First, we load the raw data from the dataset - year 2008
RAW_DATA = LOAD '/home/kinnar/Desktop/proj_dataset/2008.csv' USING
PigStorage(',') AS
(year: int, month: int, day: int, dow: int,
 dtime: int, sdttime: int, arftime: int, satime: int,
 carrier: chararray, fn: int, tn: chararray,
 etime: int, setime: int, airtime: int,
 adelay: int, ddelay: int,
 scode: chararray, dcode: chararray, dist: int,
 tintime: int, touttime: int,
 cancel: chararray, cancelcode: chararray, diverted: int,
 cdelay: int, wdelay: int, ndelay: int, sdelay: int, latedelay: int);

-----

-- APPROACH 1:
-- The idea is to build a frequency table for the unordered pair (i,j) where i
and j are distinct airport codes
-- This means we are not interested in any relative counts. In APPROACH 2 we
will see how to do this

-- QUESTION: what about the shuffle key space? Is it balanced? How can it be
made balanced?
-----

-- project to get rid of unused fields
A = FOREACH RAW_DATA GENERATE scode AS s, dcode AS d;

-- group by (s,d) pair
B = GROUP A BY (s,d);

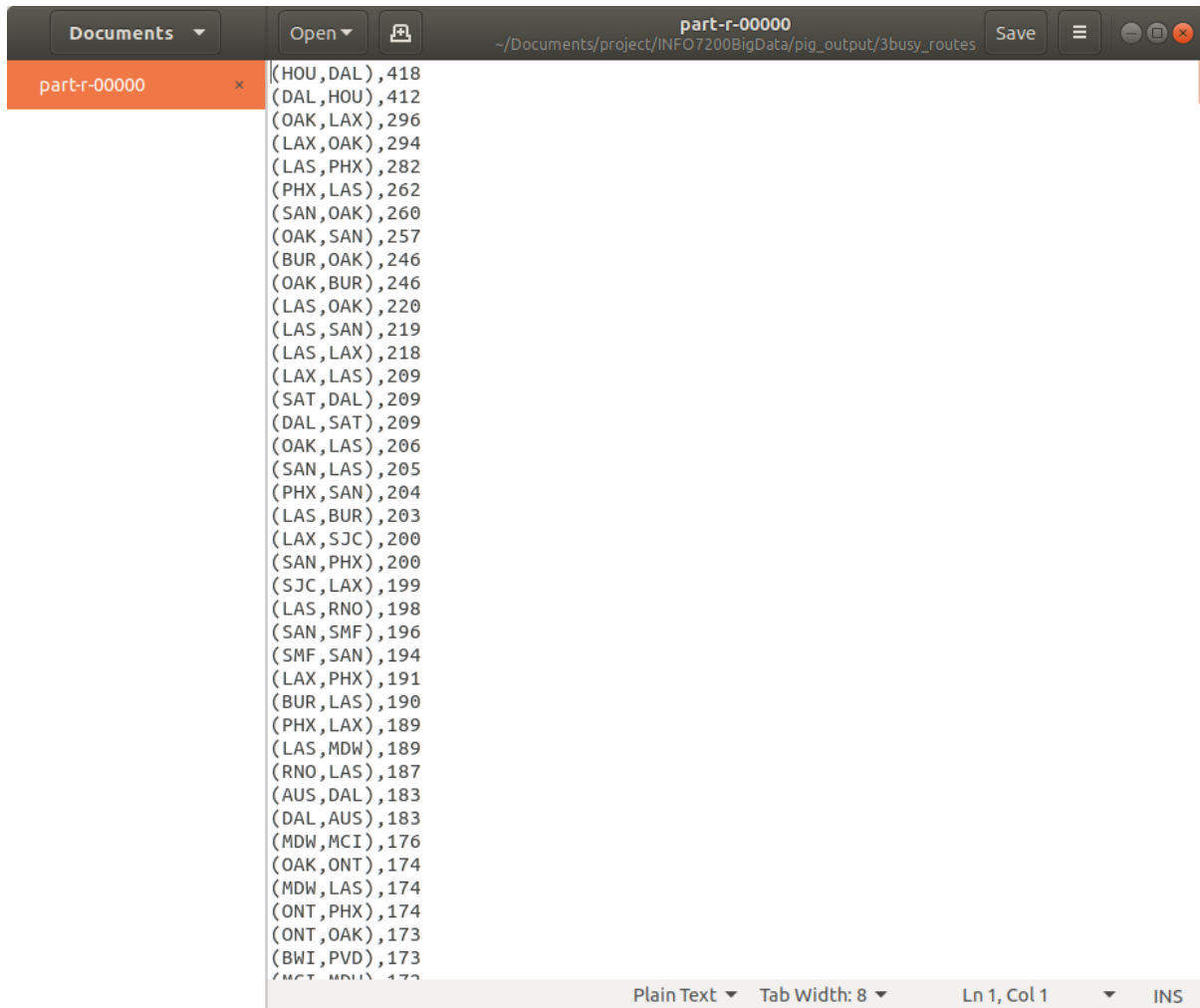
COUNT = FOREACH B GENERATE group, COUNT(A) AS CNT;

INVORDER = ORDER COUNT BY CNT DESC;

--dump COUNT;
STORE INVORDER INTO '/home/kinnar/Documents/project/INF072008BigData/pig_output/
3busy_routes' USING PigStorage(',');
```

Output: We are sorting them in descending order so we can check the busiest and popular pair of airports which in turn are source and destinations

CONGESTION IN THE SKY



5. Analysis on Flights using Hive on Hadoop

Hive makes it easy to do analysis since it supports SQL which is one of the most familiar languages for data extraction and manipulation.

Following are the commands used for executing some tasks on Hive. They are according to the below steps:

5.1. Create FlightSchema and setup the parameters

```
create schema FlightSchema;  
  
use FlightSchema;  
  
SET hive.exec.dynamic.partition = true;  
  
SET hive.exec.dynamic.partition.mode = nonstrict;
```

5.2. Create table flights and load data from hdfs path

```
create external table flights(Year INT, Month INT, DayofMonth INT,  
DayOfWeek INT, DepTime INT, CRSDepTime INT, ArrTime INT, CRSArrTime
```

CONGESTION IN THE SKY

```
INT, UniqueCarrier String, FlightNum INT, TailNum String,  
ActualElapsedTime INT, CRSElapsedTime INT, AirTime INT, ArrDelay  
INT, DepDelay INT, Origin String, Dest String, Distance INT, TaxiIn  
INT, TaxiOut INT, Cancelled INT, CancellationCode String, Diverted  
String, CarrierDelay INT, WeatherDelay INT, NASDelay INT,  
SecurityDelay INT, LateAircraftDelay INT ) ROW FORMAT DELIMITED  
FIELDS TERMINATED BY ',';
```

```
LOAD DATA INPATH '/project/input/flights.csv' OVERWRITE INTO TABLE  
flights;
```

5.3. Create table airports and load data from hdfs path

```
create external table airports (Iata String, airport String, city  
String, state String, country String, lat String, longi String) ROW  
FORMAT DELIMITED FIELDS TERMINATED BY ',';
```

```
LOAD DATA INPATH '/project/input/airports.csv' OVERWRITE INTO TABLE  
airports;
```

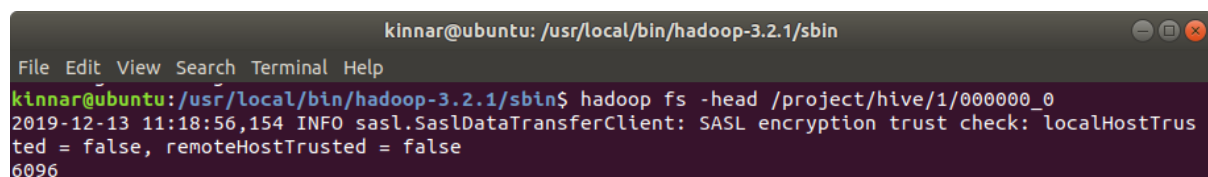
5.4. Create table carriers and load data from hdfs path

```
create external table carriers (Code String, Description String) ROW  
FORMAT DELIMITED FIELDS TERMINATED BY ',';
```

```
LOAD DATA INPATH '/project/input/carriers.csv' OVERWRITE INTO TABLE  
carriers;
```

5.5. Find flights which travelled more than 500 airmiles

```
INSERT OVERWRITE DIRECTORY '/project/hive/1' select count(*) from  
flights where AirTime > 500;
```



The screenshot shows a terminal window titled 'kinnar@ubuntu: /usr/local/bin/hadoop-3.2.1/sbin'. The command executed is 'hadoop fs -head /project/hive/1/000000_0'. The output shows a log entry from 'sasL.SaslDataTransferClient' indicating a SASL encryption trust check where 'localHostTrusted = false' and 'remoteHostTrusted = false'. The output is truncated with '6096' at the bottom.

5.6. Find flights which arrive and depart on time

```
INSERT OVERWRITE DIRECTORY '/project/hive/2' select  
Year,Month,DayofMonth,Origin,Dest,AirTime,Distance,TaxiIn,TaxiOut  
from flights where DepTime<=CRSDepTime and ArrTime<=CRSArrTime;
```


CONGESTION IN THE SKY

```

kinnar@ubuntu: /usr/local/bin/apache-hive-3.1.2-bin
File Edit View Search Terminal Help
hive> INSERT OVERWRITE DIRECTORY '/project/hive/3' Select carriers.description, uniqCount.countCance
lled, uniqCount.countCarrier from (Select UniqueCarrier, sum(cancelled) as countCancelled, count(*)
as countCarrier from flights group by UniqueCarrier) AS uniqCount, carriers where carriers.code = un
iqCount.UniqueCarrier;
Query ID = kinnar_20191213111003_6563ff30-c79a-45c0-abb0-40b4dc09db4e
Total jobs = 2
Launching Job 1 out of 2
Number of reduce tasks not specified. Estimated from input data size: 6
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1576263822604_0002, Tracking URL = http://ubuntu:8088/proxy/application_157626382
2604_0002/
Kill Command = /usr/local/bin/hadoop-3.2.1/bin/mapred job -kill job_1576263822604_0002
Hadoop job information for Stage-1: number of mappers: 6; number of reducers: 6
2019-12-13 11:10:17,750 Stage-1 map = 0%, reduce = 0%
2019-12-13 11:10:29,263 Stage-1 map = 17%, reduce = 0%, Cumulative CPU 6.81 sec
2019-12-13 11:10:38,666 Stage-1 map = 33%, reduce = 0%, Cumulative CPU 13.63 sec
2019-12-13 11:10:49,217 Stage-1 map = 50%, reduce = 0%, Cumulative CPU 20.66 sec
2019-12-13 11:11:01,806 Stage-1 map = 67%, reduce = 0%, Cumulative CPU 27.95 sec
2019-12-13 11:11:17,592 Stage-1 map = 83%, reduce = 0%, Cumulative CPU 35.0 sec
2019-12-13 11:11:30,224 Stage-1 map = 100%, reduce = 0%, Cumulative CPU 39.23 sec
2019-12-13 11:11:40,718 Stage-1 map = 100%, reduce = 17%, Cumulative CPU 41.99 sec
2019-12-13 11:11:51,217 Stage-1 map = 100%, reduce = 33%, Cumulative CPU 44.58 sec
2019-12-13 11:12:02,787 Stage-1 map = 100%, reduce = 50%, Cumulative CPU 48.07 sec
2019-12-13 11:12:10,088 Stage-1 map = 100%, reduce = 67%, Cumulative CPU 50.4 sec
2019-12-13 11:12:16,255 Stage-1 map = 100%, reduce = 83%, Cumulative CPU 53.25 sec
2019-12-13 11:12:21,432 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 55.49 sec
MapReduce Total cumulative CPU time: 55 seconds 490 msec
Ended Job = job_1576263822604_0002
Execution completed successfully
MapredLocal task succeeded
Launching Job 2 out of 2
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1576263822604_0003, Tracking URL = http://ubuntu:8088/proxy/application_157626382
2604_0003/
Kill Command = /usr/local/bin/hadoop-3.2.1/bin/mapred job -kill job_1576263822604_0003

```

5.8. Find origin and destination pairs from DEN airport from 2008 data

```

INSERT OVERWRITE DIRECTORY '/project/hive/4'
select f.Origin, f.Dest, count(*) cnt
FROM airports a
JOIN flights f ON (a.Iata = f.Origin)
JOIN airports b ON (b.Iata = f.Dest)
WHERE f.Origin = 'DEN' AND f.Year = 2018
GROUP BY f.Origin, f.Dest;

```

CONGESTION IN THE SKY

```

kinnar@ubuntu: /usr/local/bin/apache-hive-3.1.2-bin
File Edit View Search Terminal Help
hive> INSERT OVERWRITE DIRECTORY '/project/hive/4'
> select f.Origin, f.Dest, count(*) cnt
> FROM airports a
> JOIN flights f ON (a.Iata = f.Origin)
> WHERE f.Origin = 'DEN' AND f.Year = 2018
> GROUP BY f.Origin, f.Dest;
Warning: Map Join MAPJOIN[19][bigTable=?] in task 'Stage-2:MAPRED' is a cross product
Query ID = kinnar_20191213112041_e4969725-45ac-44c2-89b7-30c20d085d77
Total jobs = 1

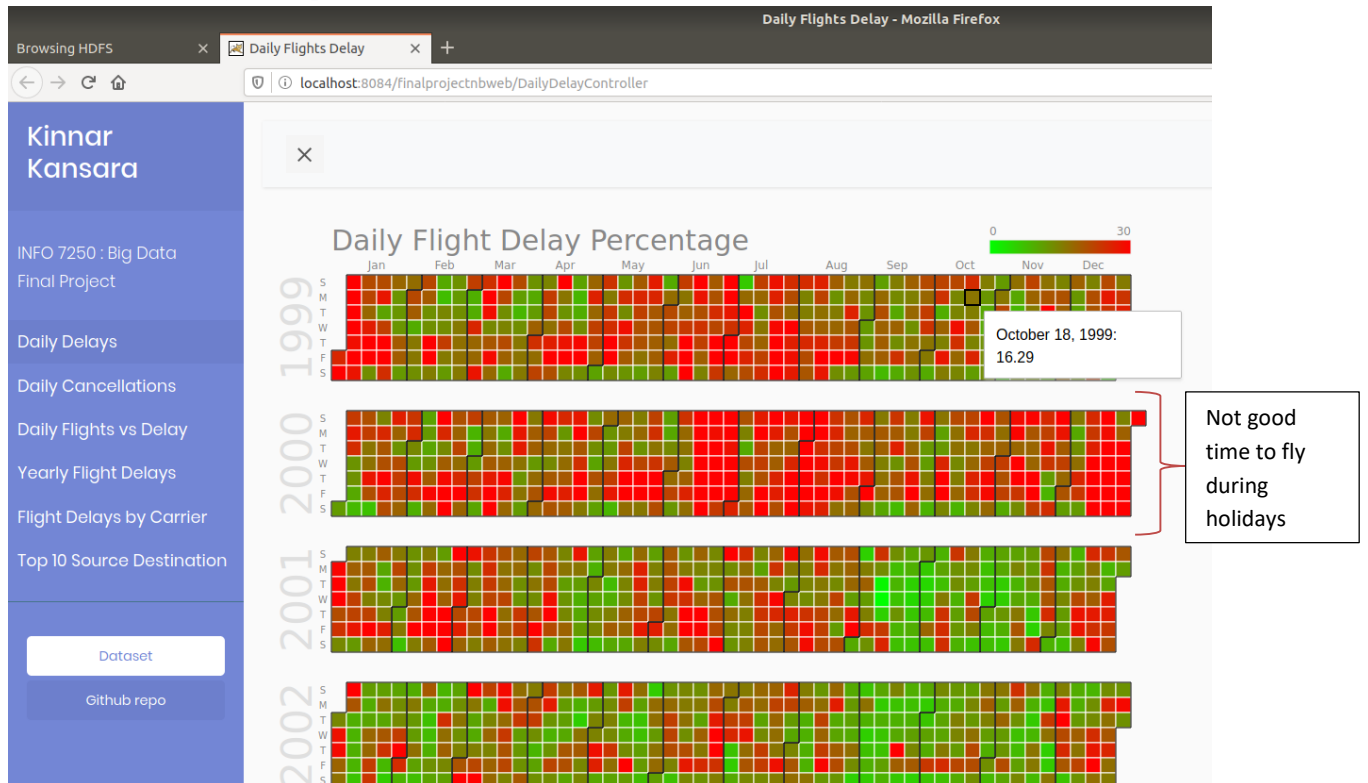
2019-12-13 11:20:51      Uploaded 1 File to: file:/tmp/kinnar/e1e164ea-0b5d-4372-9ebe-e05c742e0ad8/hive_2019-12-13_11-20-41_444_2150743585144938746-1/-local-10003/HashTable-Stage-2/MapJoin-mapfile10--.hashtable (260 bytes)
Execution completed successfully
MapredLocal task succeeded
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 6
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1576263822604_0004, Tracking URL = http://ubuntu:8088/proxy/application_1576263822604_0004/
Kill Command = /usr/local/bin/hadoop-3.2.1/bin/mapred job -kill job_1576263822604_0004
Hadoop job information for Stage-2: number of mappers: 6; number of reducers: 6
2019-12-13 11:21:02,044 Stage-2 map = 0%,   reduce = 0%
2019-12-13 11:21:12,565 Stage-2 map = 17%,  reduce = 0%, Cumulative CPU 6.33 sec
2019-12-13 11:21:21,054 Stage-2 map = 33%,  reduce = 0%, Cumulative CPU 12.65 sec
2019-12-13 11:21:31,505 Stage-2 map = 50%,  reduce = 0%, Cumulative CPU 19.47 sec
2019-12-13 11:21:43,130 Stage-2 map = 67%,  reduce = 0%, Cumulative CPU 26.59 sec
2019-12-13 11:21:52,534 Stage-2 map = 83%,  reduce = 0%, Cumulative CPU 32.68 sec
2019-12-13 11:22:00,852 Stage-2 map = 100%, reduce = 0%, Cumulative CPU 37.42 sec
2019-12-13 11:22:09,928 Stage-2 map = 100%, reduce = 17%, Cumulative CPU 40.71 sec
2019-12-13 11:22:20,019 Stage-2 map = 100%, reduce = 33%, Cumulative CPU 44.16 sec
2019-12-13 11:22:29,057 Stage-2 map = 100%, reduce = 50%, Cumulative CPU 48.58 sec
2019-12-13 11:22:35,294 Stage-2 map = 100%, reduce = 67%, Cumulative CPU 51.4 sec
2019-12-13 11:22:40,505 Stage-2 map = 100%, reduce = 83%, Cumulative CPU 54.88 sec
2019-12-13 11:22:46,714 Stage-2 map = 100%, reduce = 100%, Cumulative CPU 58.22 sec

```

6. Graphical Representation and Analysis

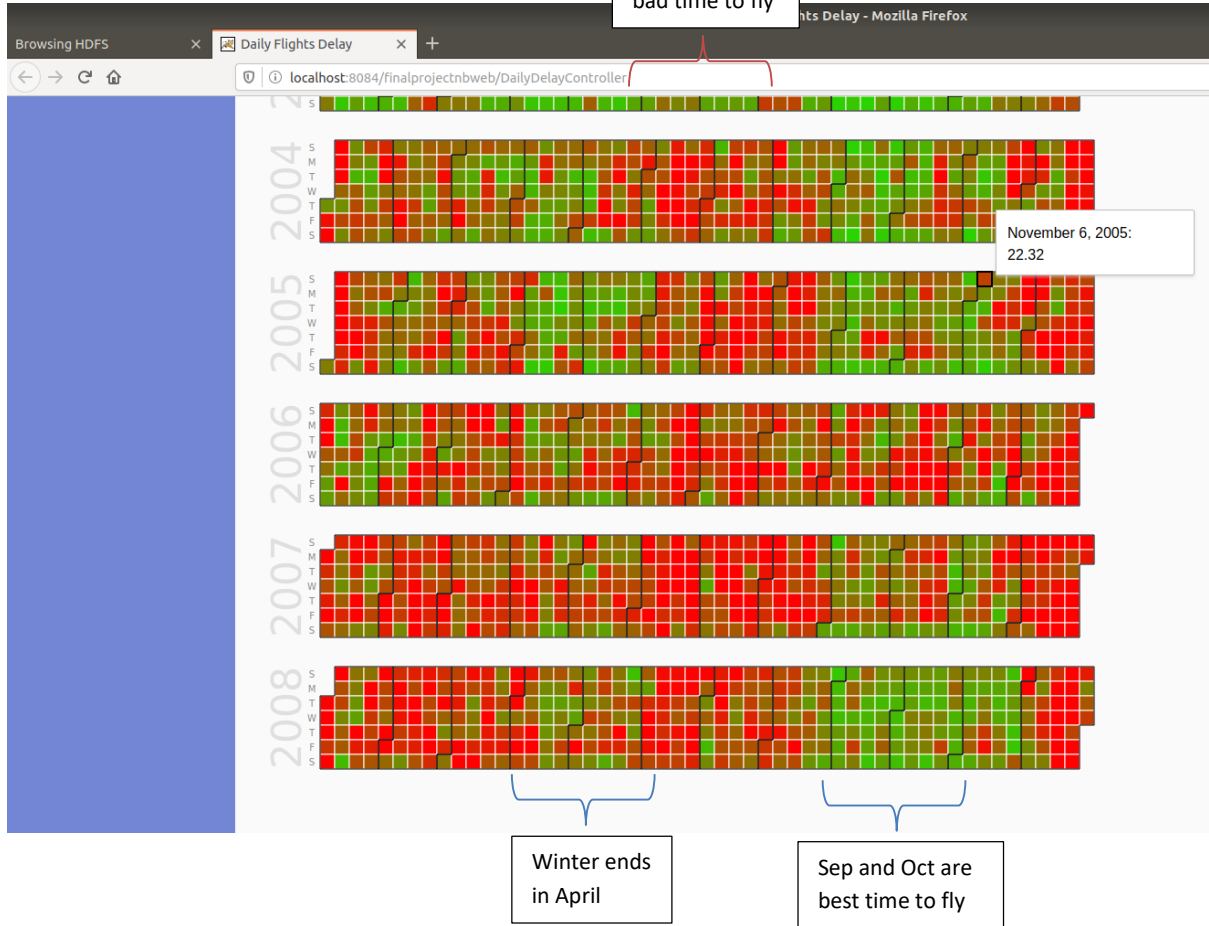
We can do many meaningful analyses from the graphical representation rather than the textual data. I will show you how impactful the graphical representation can be.

6.1. % of flight departures delayed > 15 minutes – daily basis



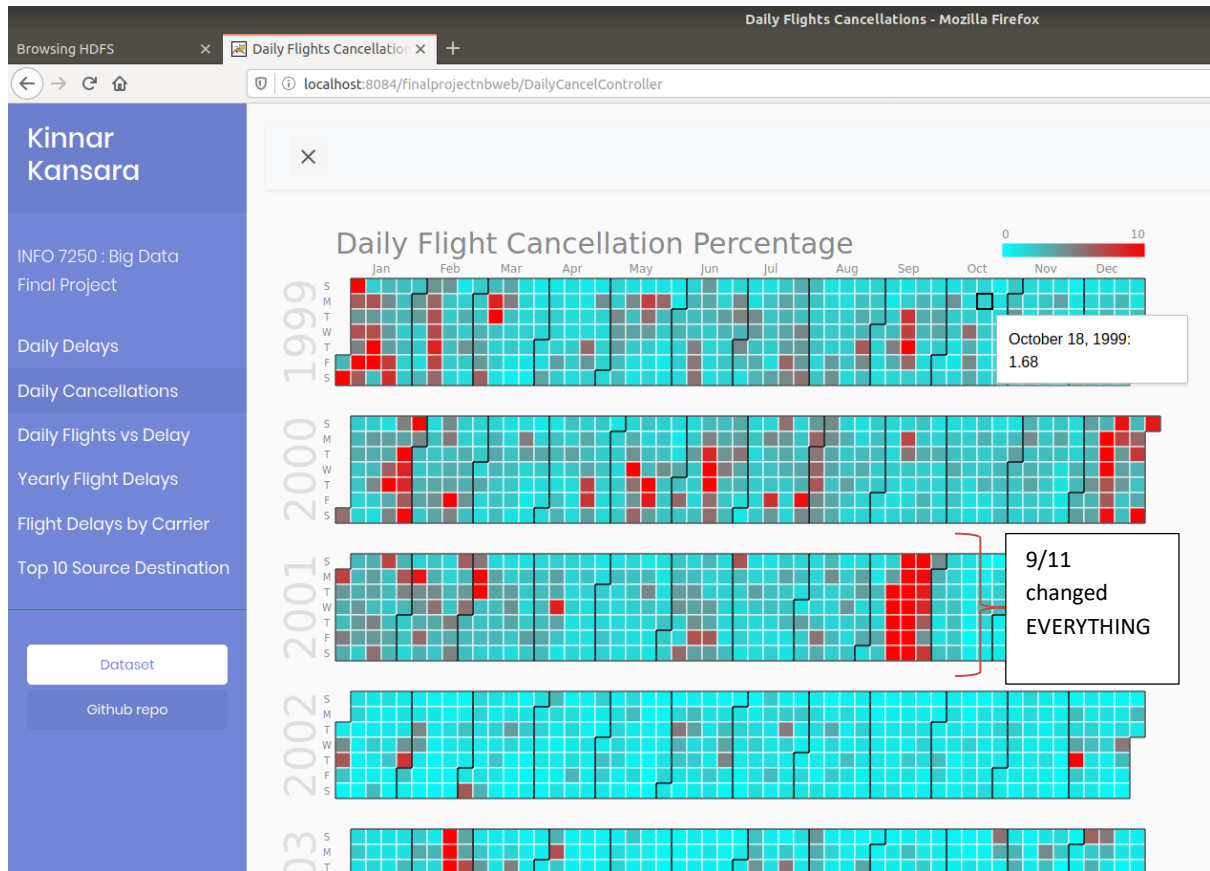
CONGESTION

Summers are
bad time to fly

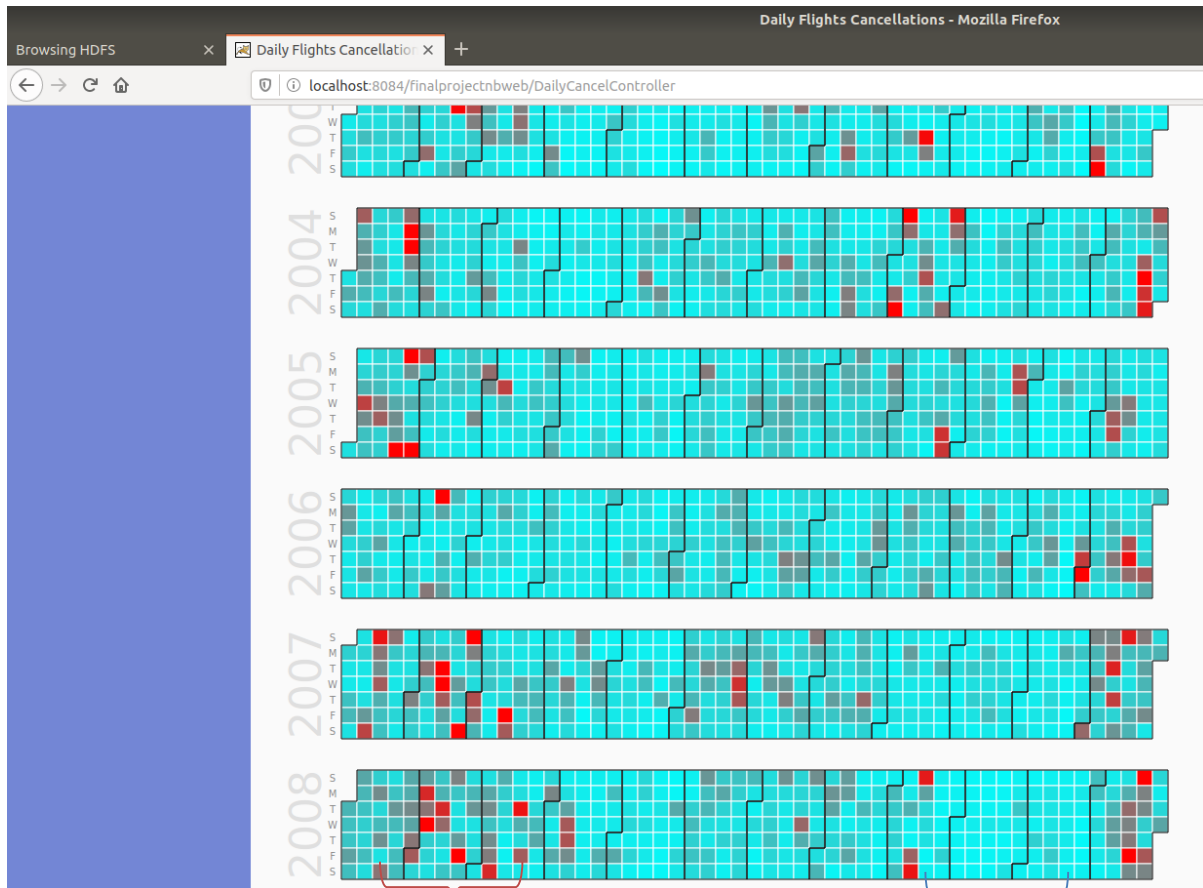


CONGESTION IN THE SKY

6.2. % of flight cancelled – daily basis



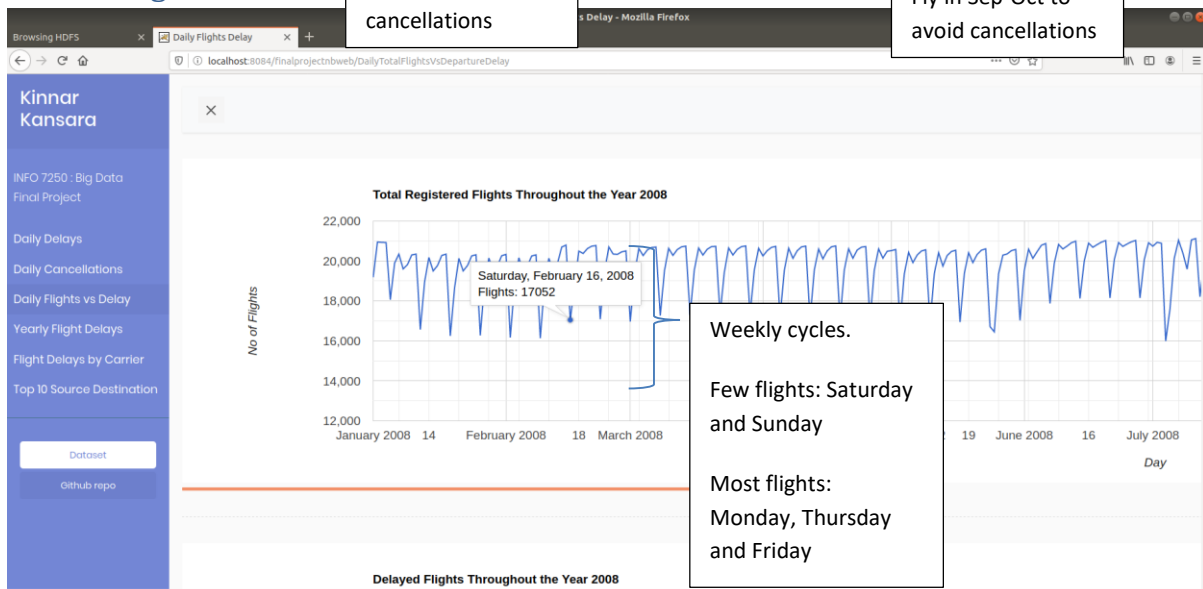
CONGESTION IN THE SKY



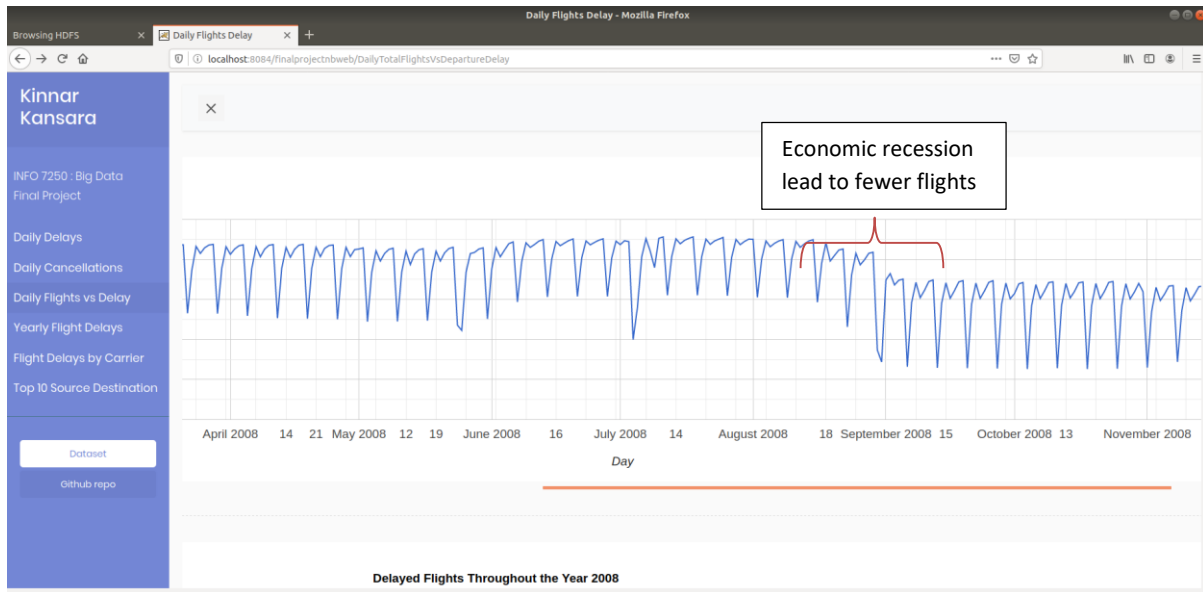
6.3. Flight trend on

Winter leads to
cancellations

Fly in Sep-Oct to
avoid cancellations

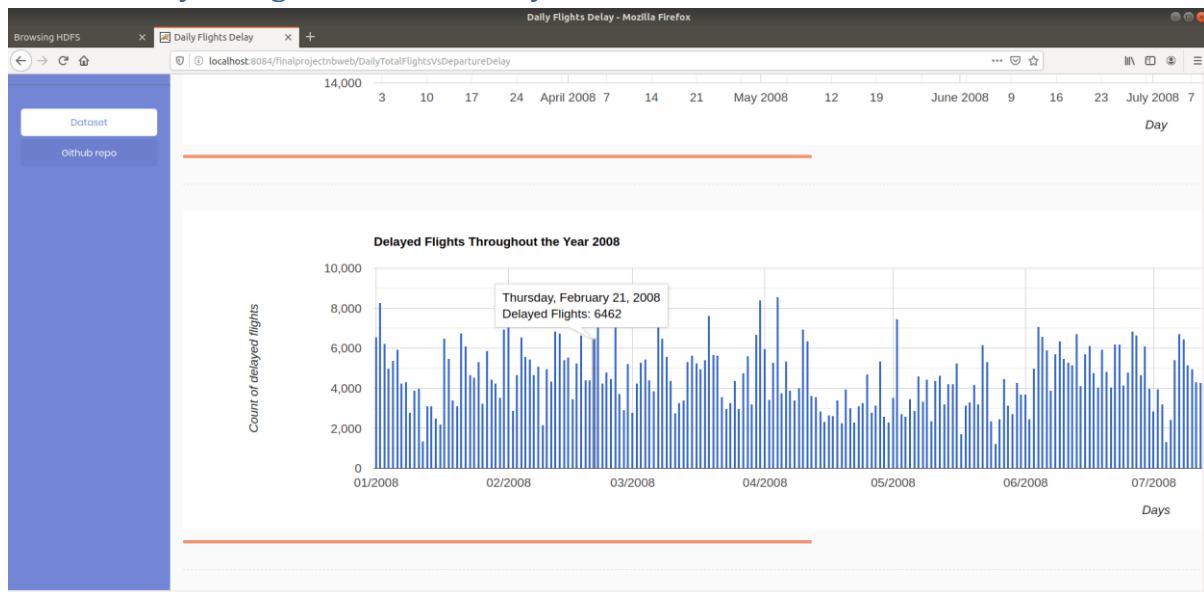


CONGESTION IN THE SKY

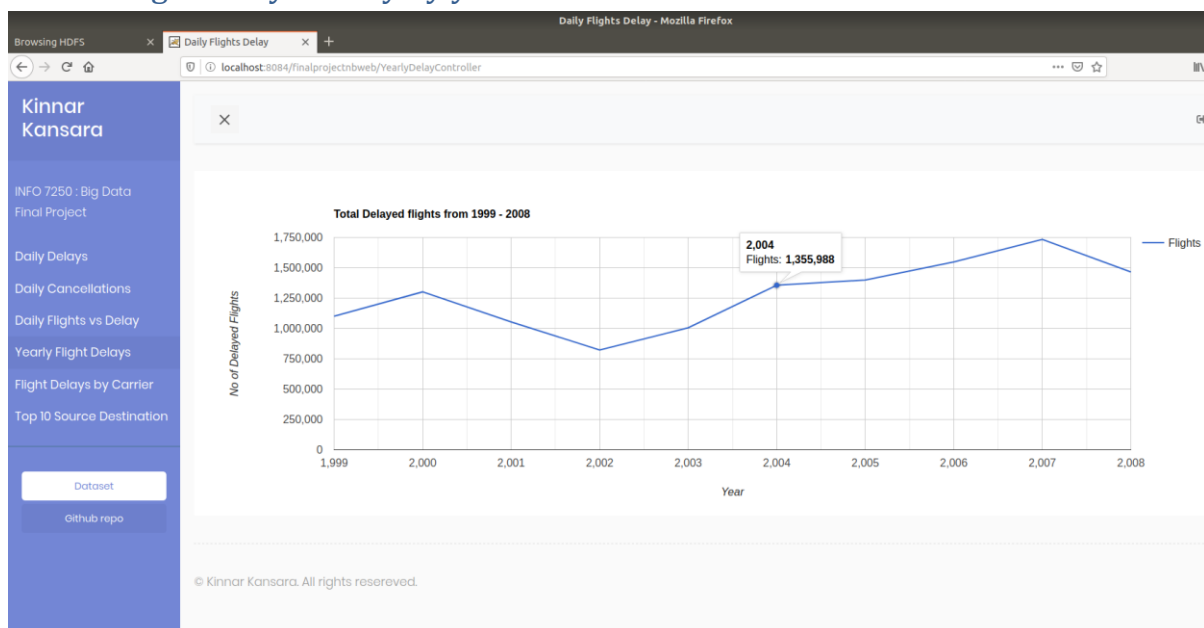


CONGESTION IN THE SKY

6.4. Delayed flights trend on daily basis

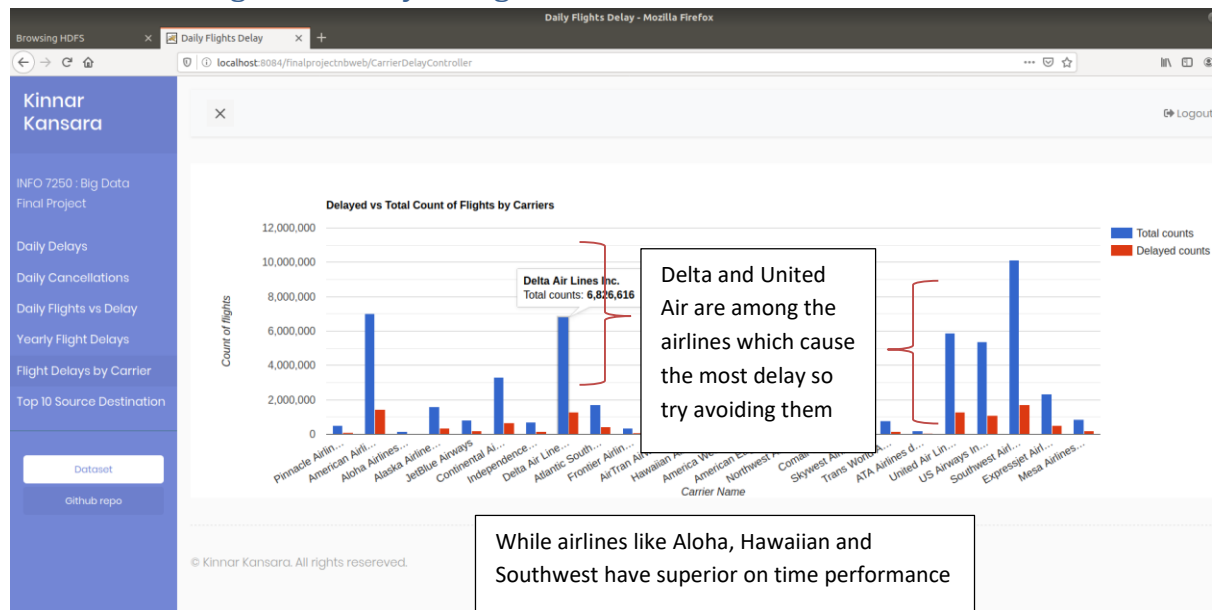


6.5. Flight delay history by year



CONGESTION IN THE SKY

6.6. Total flights vs delayed flights for each carrier



6.7. Tabular view of top 10 most popular source destination pairs

Top 10 Source Destination Pair Counts			
#	Source	Destination	Count
1	LAX	LAS	137997
2	LAS	LAX	135016
3	PHX	LAX	123517
4	LAX	PHX	121631
5	SFO	LAX	117850
6	LAX	SFO	116318
7	LAS	PHX	116273
8	ORD	MSP	114370
9	PHX	LAS	114233
10	MSP	ORD	113753

7. Lessons Learned & tips for travellers

- Avoid flying during holidays and summer to avoid delays and cancellations due to huge rush
- Fly in April, May, September and October
- Watch the weather!
- Avoid busy airports like JFK, Newark, Chicago which are causing consistent delays
- Use carriers like Southwest, Aloha and Hawaiian with better on-time performance
- Avoid flights which depart at peak hours like 5 to 7 pm.
- Some factors like 9/11 cause tremendous impact on the aviation industry

8. Challenges

- I could incorporate the weather details with the analysis but due to inconsistent weather and flight data could not lead to a valuable analysis

9. Future Scope

- More analysis with graphical representations can lead to better timing performances
- I could do better analysis with aircrafts like how old are they, what models generally create delays, etc.

10. References

<https://www.oreilly.com/library/view/data-algorithms/9781491906170/?ar>
<https://learning.oreilly.com/library/view/mapreduce-design-patterns/9781449341954/>
<http://stat-computing.org/dataexpo/2009/posters/wicklin-allison.pdf>
<https://developers.google.com/chart/interactive/docs>
<http://timepasstechies.com/category/programming/data-analytics/hive/>
http://hadoopilluminated.com/hadoop_illuminated/Public_Bigdata_Sets.html

11. Appendix

11.1. Getting top 10 source destination airport pairs

CONGESTION IN THE SKY

```
package com.kinnar.bigdataproject.top10_busy_airports;

import org.apache.hadoop.mapreduce.Mapper;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;

public class SrcDestMapper extends Mapper<LongWritable, Text, Text,
IntWritable> {
    Text word = new Text();
    IntWritable one = new IntWritable(1);

    @Override
    protected void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException {

        String line = value.toString();
        String[] data = line.split(",");
        if (data[0].equals("Year"))
            return;
        String orig_dest_pair = data[16] + "-" + data[17];
        word.set(orig_dest_pair);
        context.write(word, one);
    }
}
--

package com.kinnar.bigdataproject.top10_busy_airports;

import org.apache.hadoop.mapreduce.Reducer;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;

public class SrcDestReducer extends Reducer<Text, IntWritable, Text,
IntWritable> {
    @Override
    protected void reduce(Text key, Iterable<IntWritable> values, Context
context)
throws IOException, InterruptedException {

        int sum = 0;
        for (IntWritable v : values) {
            sum += v.get();
        }
        context.write(key, new IntWritable(sum));
    }
}
--

package com.kinnar.bigdataproject.top10_busy_airports;

import java.io.IOException;
import java.util.TreeMap;
import org.apache.hadoop.io.NullWritable;
```


CONGESTION IN THE SKY

```

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class TopNAirportsMapper extends Mapper<Object, Text, NullWritable,
Text> {
    private TreeMap<Integer, Text> counter = new TreeMap<>();

    @Override
    public void map(Object key, Text value, Context context) throws
IOException, InterruptedException {
        String[] val = value.toString().split("\\t");

        if (val.length == 2) {
            int count = Integer.parseInt(val[1]);
            counter.put(count, new Text(value));
        }

        if (counter.size() > 10)
            counter.remove(counter.firstKey());
    }

    @Override
    protected void cleanup(Context context) throws IOException,
InterruptedException {
        for (Text t : counter.values())
            context.write(NullWritable.get(), t);
    }
}
--

package com.kinnar.bigdataproject.top10_busy_airports;

import org.apache.hadoop.mapreduce.Reducer;
import java.io.IOException;
import java.util.TreeMap;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;

public class TopNAirportsReducer extends Reducer<NullWritable, Text,
NullWritable, Text> {
    private TreeMap<Integer, Text> counter = new TreeMap<>();

    @Override
    protected void reduce(NullWritable key, Iterable<Text> values,
Context context)
        throws IOException, InterruptedException {
        for (Text value : values) {
            String[] val = value.toString().split("\\t");

            if (val.length == 2) {
                int count = Integer.parseInt(val[1]);
                counter.put(count, new Text(value));
            }

            if (counter.size() > 10)
                counter.remove(counter.firstKey());
        }

        for (Text t : counter.descendingMap().values())
            context.write(NullWritable.get(), t);
    }
}

```

CONGESTION IN THE SKY

```

}
--

package com.kinnar.bigdataproject.top10_busy_airports;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class TopNApp {
    public static void main(String[] args) throws Exception {

        Configuration conf1 = new Configuration();

        Job job1 = Job.getInstance(conf1);
        job1.setJarByClass(TopNApp.class);
        job1.setJobName("Get src-dest airport combination count");

        FileInputFormat.setInputPaths(job1, new Path(args[0]));
        FileOutputFormat.setOutputPath(job1, new Path(args[1] +
"/temp/topnintermediate"));

        job1.setMapperClass(SrcDestMapper.class);
        job1.setReducerClass(SrcDestReducer.class);

        job1.setOutputKeyClass(Text.class);
        job1.setOutputValueClass(IntWritable.class);

        job1.setInputFormatClass(TextInputFormat.class);
        job1.setOutputFormatClass(TextOutputFormat.class);

        job1.setMapOutputKeyClass(Text.class);
        job1.setMapOutputValueClass(IntWritable.class);

        if (!job1.waitForCompletion(true)) {
            System.exit(1);
        }

        Configuration conf2 = new Configuration();

        Job job2 = Job.getInstance(conf2);
        job2.setJarByClass(TopNApp.class);
        job2.setJobName("Top 10 Source Destination airport combination
from previous MR job");

        FileInputFormat.setInputPaths(job2, new Path(args[1] +
"/temp/topnintermediate"));
        FileOutputFormat.setOutputPath(job2, new Path(args[1] +
"/Top10SourceDestinations"));

        job2.setMapperClass(TopNAirportsMapper.class);
        job2.setReducerClass(TopNAirportsReducer.class);

        job2.setMapOutputKeyClass(NullWritable.class);

```

CONGESTION IN THE SKY

```
job2.setMapOutputValueClass(Text.class);

job2.setInputFormatClass(TextInputFormat.class);
job2.setOutputFormatClass(TextOutputFormat.class);

job2.setMapOutputKeyClass(NullWritable.class);
job2.setMapOutputValueClass(Text.class);

if (!job2.waitForCompletion(true)) {
    System.exit(1);
}
}
```

11.2. Unique Carriers Names with Flights Count

```
package com.kinnar.bigdataproject.unique_carrier_names;

import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class UniqueCarrierMapper extends Mapper<LongWritable, Text, Text,
IntWritable> {
    Text word = new Text();
    IntWritable one = new IntWritable(1);

    @Override
    public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {
        String line = value.toString();
        String[] data = line.split(",");

        if (data[0].equals("Year"))
            return;

        String carrier = data[8];
        word.set(carrier);
        context.write(word, one);
    }
}

package com.kinnar.bigdataproject.unique_carrier_names;

import org.apache.hadoop.mapreduce.Reducer;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;

public class UniqueCarrierReducer extends Reducer<Text, IntWritable, Text,
IntWritable> {

    @Override
    protected void reduce(Text key, Iterable<IntWritable> values, Context
context)
        throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable value : values) {
            sum += value.get();
        }
        context.write(key, new IntWritable(sum));
    }
}
```

CONGESTION IN THE SKY

```
package com.kinnar.bigdataproject.unique_carrier_names;

import org.apache.hadoop.conf.Configuration;
//import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
//import org.apache.hadoop.mapreduce.lib.jobcontrol.ControlledJob;
//import org.apache.hadoop.mapreduce.lib.jobcontrol.JobControl;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class CarriersApp {
    public static void main(String[] args) throws Exception {

        Configuration conf1 = new Configuration();

        Job job1 = Job.getInstance(conf1);
        job1.setJarByClass(CarriersApp.class);
        job1.setJobName("Unique carrier count");

        FileInputFormat.setInputPaths(job1, new Path(args[0]));
        FileOutputFormat.setOutputPath(job1, new Path(args[1] +
"/temp/carriersintermediate"));

        job1.setMapperClass(UniqueCarrierMapper.class);
        job1.setReducerClass(UniqueCarrierReducer.class);

        job1.setInputFormatClass(TextInputFormat.class);
        job1.setOutputFormatClass(TextOutputFormat.class);

        job1.setMapOutputKeyClass(Text.class);
        job1.setMapOutputValueClass(IntWritable.class);

        job1.setOutputKeyClass(Text.class);
        job1.setOutputValueClass(IntWritable.class);

        if (!job1.waitForCompletion(true)) {
            System.exit(1);
        }
    }
}

package com.kinnar.bigdataproject.unique_carrier_names;

import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
```

CONGESTION IN THE SKY

CONGESTION IN THE SKY

```

public class FlightDetailsMapper extends Mapper<LongWritable, Text, Text,
Text> {
    Text word = new Text();
    IntWritable one = new IntWritable(1);

    @Override
    public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {
        String line = value.toString();
        String[] data = line.split("\t");

        String newKey = data[0];
        word.set(newKey);
        System.out.println("Bkey:" + newKey + ":");
        String outValue = "B" + data[1]; // right table
        context.write(word, new Text(outValue));
    }
}
--

package com.kinnar.bigdataproject.unique_carrier_names;

import org.apache.hadoop.mapreduce.Mapper;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;

public class CarrierInfoMapper extends Mapper<LongWritable, Text, Text,
Text> {
    Text word = new Text();
    IntWritable one = new IntWritable(1);

    @Override
    protected void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException {

        String line = value.toString();
        String[] data = line.split(",");
        if (data[0].equals("Code"))
            return;
        String nkey = data[0].replace("\\"", "");
        System.out.println("Akey:" + nkey + ":");
        word.set(nkey);
        String out = "A" + data[1].replace("\\"", ""); // left table
        context.write(word, new Text(out));
    }
}
--

package com.kinnar.bigdataproject.unique_carrier_names;

import org.apache.hadoop.mapreduce.Reducer;
import java.io.IOException;
import java.util.ArrayList;
import org.apache.hadoop.io.Text;

public class FlightDetailsReducer extends Reducer<Text, Text, Text, Text> {
    private ArrayList<Text> listA = new ArrayList<>();
    private ArrayList<Text> listB = new ArrayList<>();
    private String jointype = "inner";

```

CONGESTION IN THE SKY

```

@Override
    protected void setup(Context context) throws IOException,
        InterruptedException {
        // joinType = context.getConfiguration().get("join.type");
        joinType = "inner";
    }

    @Override
    protected void reduce(Text key, Iterable<Text> values, Context
context) throws IOException, InterruptedException {
        listA.clear();
        listB.clear();
        for (Text text : values) {
            if (text.charAt(0) == 'A') {
                listA.add(new Text(text.toString().substring(1)));
            } else if (text.charAt(0) == 'B') {
                listB.add(new Text(text.toString().substring(1)));
            }
        }
        executeInnerJoin(context);
    }

    private void executeInnerJoin(Context context) throws IOException,
        InterruptedException {
        // System.out.println("A size:" + listA.size() + " B size:" +
listB.size());
        if (joinType.equals("inner")) {
            if (!listA.isEmpty() && !listB.isEmpty()) {
                for (Text textA : listA) {
                    for (Text textB : listB) {
                        context.write(textA, textB);
                    }
                }
            }
        }
    }
}
--

package com.kinnar.bigdataproject.unique_carrier_names;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.MultipleInputs;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class CarriersApp2 {
    public static void main(String[] args) throws Exception {

        Configuration conf2 = new Configuration();

        Job job2 = Job.getInstance(conf2);
        job2.setJarByClass(CarriersApp2.class);
        job2.setJobName("Reducer Side Inner Join");

        MultipleInputs.addInputPath(job2, new Path(args[1]),
        TextInputFormat.class, CarrierInfoMapper.class);
    }
}

```


CONGESTION IN THE SKY

```
MultipleInputs.addInputPath(job2, new Path(args[0] +  
"/temp/carriersintermediate"), TextInputFormat.class,  
    FlightDetailsMapper.class);  
  
    job2.setReducerClass(FlightDetailsReducer.class);  
    job2.setNumReduceTasks(1);  
  
    job2.setOutputKeyClass(Text.class);  
    job2.setOutputValueClass(Text.class);  
  
    TextOutputFormat.setOutputPath(job2, new Path(args[2]));  
    System.exit(job2.waitForCompletion(true) ? 0 : 1);  
}  
}
```

CONGESTION IN THE SKY

11.3. Year wise flight delay (> 15 minutes) and cancellation. Counts and Ratio

CONGESTION IN THE SKY

```
package com.kinnar.bigdataproject.yearly_delay;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import org.apache.hadoop.io.Writable;

public class DelayRatioTuple implements Writable {

    private int flightsCount = 0;
    private int delayedFlightsCount = 0;
    private double delayPercentage = 0.0;
    private int canceledFlightsCount = 0;
    private double canceledPercentage = 0.0;

    public int getFlightsCount() {
        return flightsCount;
    }

    public void setFlightsCount(int flightsCount) {
        this.flightsCount = flightsCount;
    }

    public int getDelayedFlightsCount() {
        return delayedFlightsCount;
    }

    public void setDelayedFlightsCount(int delayedFlightsCount) {
        this.delayedFlightsCount = delayedFlightsCount;
    }

    public double getDelayPercentage() {
        return delayPercentage;
    }

    public void setDelayPercentage(double delayPercentage) {
        this.delayPercentage = delayPercentage;
    }

    public int getCanceledFlightsCount() {
        return canceledFlightsCount;
    }

    public void setCanceledFlightsCount(int canceledFlightsCount) {
        this.canceledFlightsCount = canceledFlightsCount;
    }

    public double getCanceledPercentage() {
        return canceledPercentage;
    }

    public void setCanceledPercentage(double canceledPercentage) {
        this.canceledPercentage = canceledPercentage;
    }

    @Override
    public String toString() {
```

CONGESTION IN THE SKY

```
//          Commented below for using values easily for graphical
representation
//          return  "flightsCount=" + flightsCount +
//                  ", delayedFlightsCount=" + delayedFlightsCount +
//                  ", delayPercentage=" + String.format("%.2f",
delayPercentage) +
//                  ", canceledFlightsCount=" + canceledFlightsCount +
//                  ", canceledPercentage=" + String.format("%.2f",
canceledPercentage);
        return "" + flightsCount + "," + delayedFlightsCount + "," +
String.format("%.2f", delayPercentage) + "," +
        + canceledFlightsCount + "," +
String.format("%.2f", canceledPercentage);
    }

    @Override
    public void write(DataOutput dataOutput) throws IOException {
        dataOutput.writeInt(flightsCount);
        dataOutput.writeInt(delayedFlightsCount);
        dataOutput.writeDouble(delayPercentage);
        dataOutput.writeInt(canceledFlightsCount);
        dataOutput.writeDouble(canceledPercentage);
    }

    @Override
    public void readFields(DataInput dataInput) throws IOException {
        flightsCount = dataInput.readInt();
        delayedFlightsCount = dataInput.readInt();
        delayPercentage = dataInput.readDouble();
        canceledFlightsCount = dataInput.readInt();
        canceledPercentage = dataInput.readDouble();
    }
}

--

package com.kinnar.bigdataproject.yearly_delay;

import java.io.IOException;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class YearlyDelayMapper extends Mapper<Object, Text, Text,
DelayRatioTuple> {
    private DelayRatioTuple tuple = new DelayRatioTuple();
    boolean flag = true;

    @Override
    public void map(Object key, Text value, Context context) throws
IOException, InterruptedException {
        String line = value.toString();
        String[] data = line.split(",");

        if (data[0].equals("Year"))
            return;

        String year = data[0];
        try {
            int delay = Integer.parseInt(data[14]);

            if (delay > 15) {
                tuple.setDelayedFlightsCount(1);
            }
        }
    }
}
```

CONGESTION IN THE SKY

```

        } else {
            tuple.setDelayedFlightsCount(0);
        }
    } catch (Exception e) {
        tuple.setDelayedFlightsCount(0);
    }
    try {
        if (data[21].equals("1")) {
            tuple.setCanceledFlightsCount(1);
        } else {
            tuple.setCanceledFlightsCount(0);
        }
    } catch (Exception e) {
        tuple.setCanceledFlightsCount(0);
    }
    tuple.setFlightsCount(1);
    context.write(new Text(year), tuple);
}
}
--

package com.kinnar.bigdataproject.yearly_delay;

import org.apache.hadoop.mapreduce.Reducer;
import java.io.IOException;
import org.apache.hadoop.io.Text;

public class YearlyDelayReducer extends Reducer<Text, DelayRatioTuple,
Text, DelayRatioTuple> {

    private DelayRatioTuple tuple = new DelayRatioTuple();

    @Override
    protected void reduce(Text key, Iterable<DelayRatioTuple> values,
Context context)
        throws IOException, InterruptedException {
        int total = 0;
        int delayedTotal = 0;
        int cancelledTotal = 0;

        for (DelayRatioTuple dt : values) {
            total += dt.getFlightsCount();
            delayedTotal += dt.getDelayedFlightsCount();
            cancelledTotal += dt.getCanceledFlightsCount();
        }

        double delayPercentage = ((double) delayedTotal / total) * 100;
        double cancelledPercentage = ((double) cancelledTotal / total)
* 100;

        tuple.setFlightsCount(total);
        tuple.setDelayedFlightsCount(delayedTotal);
        tuple.setDelayPercentage(delayPercentage);
        tuple.setCanceledFlightsCount(cancelledTotal);
        tuple.setCanceledPercentage(cancelledPercentage);

        context.write(key, tuple);
    }
}
--

```

CONGESTION IN THE SKY

```
package com.kinnar.bigdataproject.yearly_delay;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class YearlyDelayApp {
    public static void main(String[] args) throws Exception {

        Configuration conf1 = new Configuration();

        Job job1 = Job.getInstance(conf1);
        job1.setJarByClass(YearlyDelayApp.class);
        job1.setJobName("Yearly delay ratio with delay > 15 minutes and  
cancelled flights ratio");

        FileInputFormat.setInputPaths(job1, new Path(args[0]));
        FileOutputFormat.setOutputPath(job1, new Path(args[1]));

        job1.setMapperClass(YearlyDelayMapper.class);
        job1.setCombinerClass(YearlyDelayReducer.class);
        job1.setReducerClass(YearlyDelayReducer.class);

        job1.setInputFormatClass(TextInputFormat.class);
        job1.setOutputFormatClass(TextOutputFormat.class);

        job1.setMapOutputKeyClass(Text.class);
        job1.setMapOutputValueClass(DelayRatioTuple.class);

        job1.setOutputKeyClass(Text.class);
        job1.setOutputValueClass(DelayRatioTuple.class);

        if (!job1.waitForCompletion(true)) {
            System.exit(1);
        }
    }
}
```

11.4. Flight delay and cancellation by carrier including carrier names

```
package com.kinnar.bigdataproject.carrier_delay_cancel;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import org.apache.hadoop.io.Writable;

public class DelayRatioTuple implements Writable {

    private int flightsCount = 0;
    private int delayedFlightsCount = 0;
    private double delayPercentage = 0.0;
    private int canceledFlightsCount = 0;
    private double canceledPercentage = 0.0;

    public int getFlightsCount() {
        return flightsCount;
    }

    public void setFlightsCount(int flightsCount) {
        this.flightsCount = flightsCount;
    }

    public int getDelayedFlightsCount() {
        return delayedFlightsCount;
    }

    public void setDelayedFlightsCount(int delayedFlightsCount) {
        this.delayedFlightsCount = delayedFlightsCount;
    }

    public double getDelayPercentage() {
        return delayPercentage;
    }

    public void setDelayPercentage(double delayPercentage) {
        this.delayPercentage = delayPercentage;
    }

    public int getCanceledFlightsCount() {
        return canceledFlightsCount;
    }

    public void setCanceledFlightsCount(int canceledFlightsCount) {
        this.canceledFlightsCount = canceledFlightsCount;
    }

    public double getCanceledPercentage() {
        return canceledPercentage;
    }

    public void setCanceledPercentage(double canceledPercentage) {
        this.canceledPercentage = canceledPercentage;
    }

    @Override
    public String toString() {
```

CONGESTION IN THE SKY

```

//          Commented below lines for recommendation system and written
return just with values and not labels
//          return  "flightsCount=" + flightsCount +
//                  ", delayedFlightsCount=" + delayedFlightsCount +
//                  ", delayPercentage=" + String.format("%.2f",
delayPercentage) +
//                  ", canceledFlightsCount=" + canceledFlightsCount +
//                  ", canceledPercentage=" + String.format("%.2f",
canceledPercentage);
        return "" + flightsCount + "," + delayedFlightsCount + "," +
String.format("%.2f", delayPercentage) + "," +
        + canceledFlightsCount + "," +
String.format("%.2f", canceledPercentage);
    }

    @Override
    public void write(DataOutput dataOutput) throws IOException {
        dataOutput.writeInt(flightsCount);
        dataOutput.writeInt(delayedFlightsCount);
        dataOutput.writeDouble(delayPercentage);
        dataOutput.writeInt(canceledFlightsCount);
        dataOutput.writeDouble(canceledPercentage);
    }

    @Override
    public void readFields(DataInput dataInput) throws IOException {

        flightsCount = dataInput.readInt();
        delayedFlightsCount = dataInput.readInt();
        delayPercentage = dataInput.readDouble();
        canceledFlightsCount = dataInput.readInt();
        canceledPercentage = dataInput.readDouble();
    }
}
--

package com.kinnar.bigdataproject.carrier_delay_cancel;

import java.io.IOException;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class CarrierDelayCancelMapper extends Mapper<Object, Text, Text,
DelayRatioTuple> {
    private DelayRatioTuple tuple = new DelayRatioTuple();
    boolean flag = true;

    @Override
    public void map(Object key, Text value, Context context) throws
IOException, InterruptedException {
        String line = value.toString();
        String[] data = line.split(",");

        if (data[0].equals("Year"))
            return;

        String carrier = data[8];
        try {
            int delay = Integer.parseInt(data[14]);

```


CONGESTION IN THE SKY

```

        if (delay > 15) {
            tuple.setDelayedFlightsCount(1);
        } else {
            tuple.setDelayedFlightsCount(0);
        }
    } catch (Exception e) {
        tuple.setDelayedFlightsCount(0);
    }
    try {
        if (data[21].equals("1")) {
            tuple.setCanceledFlightsCount(1);
        } else {
            tuple.setCanceledFlightsCount(0);
        }
    } catch (Exception e) {
        tuple.setCanceledFlightsCount(0);
    }
    tuple.setFlightsCount(1);

    context.write(new Text(carrier), tuple);
}

}
--

package com.kinnar.bigdataproject.carrier_delay_cancel;

import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;
import org.apache.hadoop.io.Text;

public class CarrierDelayCancelReducer extends Reducer<Text,
DelayRatioTuple, Text, DelayRatioTuple> {

    private DelayRatioTuple tuple = new DelayRatioTuple();
    @Override
    protected void reduce(Text key, Iterable<DelayRatioTuple> values,
Context context) throws IOException, InterruptedException {
        int total=0;
        int delayedTotal=0;
        int cancelledTotal=0;

        for(DelayRatioTuple dt: values){
            total += dt.getFlightsCount();
            delayedTotal +=dt.getDelayedFlightsCount();
            cancelledTotal += dt.getCanceledFlightsCount();
        }

        double delayPercentage = ((double)delayedTotal/total)*100;
        double cancelledPercentage = ((double)cancelledTotal/total)*100;

        tuple.setFlightsCount(total);
        tuple.setDelayedFlightsCount(delayedTotal);
        tuple.setDelayPercentage(delayPercentage);
        tuple.setCanceledFlightsCount(cancelledTotal);
        tuple.setCanceledPercentage(cancelledPercentage);

        context.write(key, tuple);
    }
}

```

CONGESTION IN THE SKY

```

}
--

package com.kinnar.bigdataproject.carrier_delay_cancel;

import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class FlightDetailsMapper extends Mapper<LongWritable, Text, Text,
Text> {
    Text word = new Text();
    IntWritable one = new IntWritable(1);

    @Override
    public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {
        String line = value.toString();
        String[] data = line.split("\t");

        String newKey = data[0];
        word.set(newKey);
        System.out.println("Bkey:"+newKey+":");
        String outValue = "B"+data[1]; //right table
        context.write(word, new Text(outValue));
    }
}

}
--

package com.kinnar.bigdataproject.carrier_delay_cancel;

import org.apache.hadoop.mapreduce.Reducer;
import java.io.IOException;
import java.util.ArrayList;
import org.apache.hadoop.io.Text;

public class FlightDetailsReducer extends Reducer<Text, Text, Text, Text> {
//    private Text tmp = new Text();
    private ArrayList<Text> listA = new ArrayList<>();
    private ArrayList<Text> listB = new ArrayList<>();
    private String joinType = "inner";

    @Override
    protected void setup(Context context) throws IOException,
InterruptedException {
        // joinType = context.getConfiguration().get("join.type");
        joinType = "inner";
    }

    @Override
    protected void reduce(Text key, Iterable<Text> values, Context
context) throws IOException, InterruptedException {
        listA.clear();
        listB.clear();
        for (Text text : values) {
            if (text.charAt(0) == 'A') {
                listA.add(new Text(text.toString().substring(1)));
            } else if (text.charAt(0) == 'B') {

```

CONGESTION IN THE SKY

```

        listB.add(new Text(text.toString().substring(1)));
    }

    executeInnerJoin(context);
}

private void executeInnerJoin(Context context) throws IOException,
InterruptedException {
    // System.out.println("A size:" + listA.size() + " B size:" +
listB.size());
    if (joinType.equals("inner")) {
        if (!listA.isEmpty() && !listB.isEmpty()) {
            for (Text textA : listA) {
                for (Text textB : listB) {
                    context.write(textA, textB);
                }
            }
        }
    }
}

--

package com.kinnar.bigdataproject.carrier_delay_cancel;

import org.apache.hadoop.mapreduce.Mapper;
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;

public class CarrierInfoMapper extends Mapper<LongWritable, Text, Text,
Text> {
    Text word = new Text();
    IntWritable one = new IntWritable(1);

    @Override
    protected void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException {

        String line = value.toString();
        String[] data = line.split(",");
        if (data[0].equals("Code"))
            return;
        String nkey = data[0].replace("\\\"", "");
        System.out.println("Akey:" + nkey + ":");
        word.set(nkey);
        String out = "A" + data[1].replace("\\\"", ""); // left table
        context.write(word, new Text(out));
    }
}

--

package com.kinnar.bigdataproject.carrier_delay_cancel;

import java.net.URI;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;

```

CONGESTION IN THE SKY

```

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.MultipleInputs;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import com.kinnar.bigdataproyect.unique_carrier_names.CarrierInfoMapper;
import com.kinnar.bigdataproyect.unique_carrier_names.CarriersApp2;
import com.kinnar.bigdataproyect.unique_carrier_names.FlightDetailsMapper;
import com.kinnar.bigdataproyect.unique_carrier_names.FlightDetailsReducer;

public class CarrierDelayCancelApp {
    public static void main(String[] args) throws Exception {

        Configuration conf1 = new Configuration();

        FileSystem hdfs =
        FileSystem.get(URI.create("hdfs://localhost:9000"), conf1);

        Path temp = new
        Path("/project/temp/carriersdelaycancelintermediate");
        // delete existing directory
        if (hdfs.exists(temp)) {
            hdfs.delete(temp, true);
        }
        Path output = new Path(args[2]);
        // delete existing directory
        if (hdfs.exists(output)) {
            hdfs.delete(output, true);
        }

        Job job1 = Job.getInstance(conf1);
        job1.setJarByClass(CarrierDelayCancelApp.class);
        job1.setJobName("Carrier delay ratio with delay > 15 minutes
and cancelled flights ratio");

        FileInputFormat.setInputPaths(job1, new Path(args[0]));
        FileOutputFormat.setOutputPath(job1, temp);

        job1.setMapperClass(CarrierDelayCancelMapper.class);
        job1.setCombinerClass(CarrierDelayCancelReducer.class);
        job1.setReducerClass(CarrierDelayCancelReducer.class);

        job1.setInputFormatClass(TextInputFormat.class);
        job1.setOutputFormatClass(TextOutputFormat.class);

        job1.setMapOutputKeyClass(Text.class);
        job1.setMapOutputValueClass(DelayRatioTuple.class);

        job1.setOutputKeyClass(Text.class);
        job1.setOutputValueClass(DelayRatioTuple.class);

        if (!job1.waitForCompletion(true)) {
            System.exit(1);
        }

        Configuration conf2 = new Configuration();

        Job job2 = Job.getInstance(conf2);
        job2.setJarByClass(CarriersApp2.class);

```

CONGESTION IN THE SKY

```
job2.setJobName("Reducer Side Inner Join: Get carrier info");

    MultipleInputs.addInputPath(job2, new Path(args[1]),
TextInputFormat.class, CarrierInfoMapper.class);
    MultipleInputs.addInputPath(job2, temp, TextInputFormat.class,
FlightDetailsMapper.class);

    job2.setReducerClass(FlightDetailsReducer.class);
    job2.setNumReduceTasks(1);

    job2.setOutputKeyClass(Text.class);
    job2.setOutputValueClass(Text.class);

    TextOutputFormat.setOutputPath(job2, output);
    System.exit(job2.waitForCompletion(true) ? 0 : 1);
}

}
```

CONGESTION IN THE SKY

11.5. Date wise flight delay (> 15 minutes) and cancellation. Counts and Ratio

CONGESTION IN THE SKY

```
package com.kinnar.bigdataproject.daily_delay_cancel;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import org.apache.hadoop.io.Writable;

public class DelayRatioTuple implements Writable {

    private int flightsCount = 0;
    private int delayedFlightsCount = 0;
    private double delayPercentage = 0.0;
    private int canceledFlightsCount = 0;
    private double canceledPercentage = 0.0;

    public int getFlightsCount() {
        return flightsCount;
    }

    public void setFlightsCount(int flightsCount) {
        this.flightsCount = flightsCount;
    }

    public int getDelayedFlightsCount() {
        return delayedFlightsCount;
    }

    public void setDelayedFlightsCount(int delayedFlightsCount) {
        this.delayedFlightsCount = delayedFlightsCount;
    }

    public double getDelayPercentage() {
        return delayPercentage;
    }

    public void setDelayPercentage(double delayPercentage) {
        this.delayPercentage = delayPercentage;
    }

    public int getCanceledFlightsCount() {
        return canceledFlightsCount;
    }

    public void setCanceledFlightsCount(int canceledFlightsCount) {
        this.canceledFlightsCount = canceledFlightsCount;
    }

    public double getCanceledPercentage() {
        return canceledPercentage;
    }

    public void setCanceledPercentage(double canceledPercentage) {
        this.canceledPercentage = canceledPercentage;
    }

    @Override
    public String toString() {
```

CONGESTION IN THE SKY

```
//          Commented below for using values easily for graphical
representation
//          return  "flightsCount=" + flightsCount +
//                  ", delayedFlightsCount=" + delayedFlightsCount +
//                  ", delayPercentage=" + String.format("%.2f",
delayPercentage) +
//                  ", canceledFlightsCount=" + canceledFlightsCount +
//                  ", canceledPercentage=" + String.format("%.2f",
canceledPercentage);
        return "" + flightsCount + "," + delayedFlightsCount + "," +
String.format("%.2f", delayPercentage) + "," +
        + canceledFlightsCount + "," + String.format("%.2f",
canceledPercentage);
    }

    @Override
    public void write(DataOutput dataOutput) throws IOException {
        dataOutput.writeInt(flightsCount);
        dataOutput.writeInt(delayedFlightsCount);
        dataOutput.writeDouble(delayPercentage);
        dataOutput.writeInt(canceledFlightsCount);
        dataOutput.writeDouble(canceledPercentage);
    }

    @Override
    public void readFields(DataInput dataInput) throws IOException {

        flightsCount = dataInput.readInt();
        delayedFlightsCount = dataInput.readInt();
        delayPercentage = dataInput.readDouble();
        canceledFlightsCount = dataInput.readInt();
        canceledPercentage = dataInput.readDouble();
    }
}

--

package com.kinnar.bigdataproject.daily_delay_cancel;

import java.io.IOException;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class DailyDelayCancelMapper extends Mapper<Object, Text, Text,
DelayRatioTuple> {
    private DelayRatioTuple tuple = new DelayRatioTuple();
    boolean flag = true;

    @Override
    public void map(Object key, Text value, Context context) throws
IOException, InterruptedException {
        String line = value.toString();
        String[] data = line.split(",");

        if (data[0].equals("Year"))
```


CONGESTION IN THE SKY

```

        return;

String year = data[0];
String month = data[1];
month = String.format("%02d", Integer.parseInt(month));
String date = data[2];
date = String.format("%02d", Integer.parseInt(date));
try {
    int delay = Integer.parseInt(data[14]);

    if (delay > 15) {
        tuple.setDelayedFlightsCount(1);
    } else {
        tuple.setDelayedFlightsCount(0);
    }
} catch (Exception e) {
    tuple.setDelayedFlightsCount(0);
}
try {
    //         int cancelled = Integer.parseInt(data[21]);
    //         if(flag) {
    //             System.out.println("cancelled:"+cancelled+":");
    //             flag = !flag;
    //         }
    //         if (cancelled == 1) {
    //             if (data[21].equals("1")) {
    //                 tuple.setCanceledFlightsCount(1);
    //             } else {
    //                 tuple.setCanceledFlightsCount(0);
    //             }
    //         }
    //         } catch (Exception e) {
    //             tuple.setCanceledFlightsCount(0);
    //         }
    tuple.setFlightsCount(1);

    context.write(new Text(year + month + date), tuple);
}

}
--

package com.kinnar.bigdataproject.daily_delay_cancel;

import org.apache.hadoop.mapreduce.Reducer;
import java.io.IOException;
import org.apache.hadoop.io.Text;

public class DailyDelayCancelReducer extends Reducer<Text, DelayRatioTuple,
Text, DelayRatioTuple> {

    private DelayRatioTuple tuple = new DelayRatioTuple();

    @Override
    protected void reduce(Text key, Iterable<DelayRatioTuple> values,
Context context)
        throws IOException, InterruptedException {
        int total = 0;
        int delayedTotal = 0;
        int cancelledTotal = 0;

```

CONGESTION IN THE SKY

```

        for (DelayRatioTuple dt : values) {
            total += dt.getFlightsCount();
            delayedTotal += dt.getDelayedFlightsCount();
            cancelledTotal += dt.getCanceledFlightsCount();
        }

        double delayPercentage = ((double) delayedTotal / total) * 100;
        double cancelledPercentage = ((double) cancelledTotal / total)
* 100;

        tuple.setFlightsCount(total);
        tuple.setDelayedFlightsCount(delayedTotal);
        tuple.setDelayPercentage(delayPercentage);
        tuple.setCanceledFlightsCount(cancelledTotal);
        tuple.setCanceledPercentage(cancelledPercentage);

        context.write(key, tuple);
    }
}
--

package com.kinnar.bigdataproject.daily_delay_cancel;

import java.net.URI;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class DailyDelayCancelApp {
    public static void main(String[] args) throws Exception {

        Configuration conf1 = new Configuration();

        FileSystem hdfs =
FileSystem.get(URI.create("hdfs://localhost:9000"), conf1);

        Path output = new Path(args[1]);
        // delete existing directory
        if (hdfs.exists(output)) {
            hdfs.delete(output, true);
        }

        Job job1 = Job.getInstance(conf1);
        job1.setJarByClass(DailyDelayCancelApp.class);
        job1.setJobName("Daily delay ratio with delay > 15 minutes and
cancelled flights ratio");

        FileInputFormat.setInputPaths(job1, new Path(args[0]));
        FileOutputFormat.setOutputPath(job1, new Path(args[1]));

        job1.setMapperClass(DailyDelayCancelMapper.class);
        job1.setCombinerClass(DailyDelayCancelReducer.class);
        job1.setReducerClass(DailyDelayCancelReducer.class);

        job1.setInputFormatClass(TextInputFormat.class);

```

CONGESTION IN THE SKY

```
job1.setOutputFormatClass(TextOutputFormat.class);

job1.setMapOutputKeyClass(Text.class);
job1.setMapOutputValueClass(DelayRatioTuple.class);

job1.setOutputKeyClass(Text.class);
job1.setOutputValueClass(DelayRatioTuple.class);

if (!job1.waitForCompletion(true)) {
    System.exit(1);
}

}
```

11.6. Average distance covered and airtime done by each carrier

```
package com.kinnar.bigdataproject.avg_dist_carrier;

import org.apache.hadoop.io.Writable;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

public class AverageCountTuple implements Writable {

    private int flightCount = 0;
    private int distCount = 0;
    private int airTime = 0;
    private double averageDist = 0.0;
    private double averageAirTime = 0.0;

    public int getAirTime() {
        return airTime;
    }

    public void setAirTime(int airTime) {
        this.airTime = airTime;
    }

    public int getFlightCount() {
        return flightCount;
    }

    public void setFlightCount(int flightCount) {
        this.flightCount = flightCount;
    }

    public int getDistCount() {
        return distCount;
    }

    public void setDistCount(int distCount) {
        this.distCount = distCount;
    }

    public double getAverageDist() {
        return averageDist;
    }

    public void setAverageDist(double averageDist) {
        this.averageDist = averageDist;
    }

    public double getAverageAirTime() {
        return averageAirTime;
    }

    public void setAverageAirTime(double averageAirTime) {
        this.averageAirTime = averageAirTime;
    }

    @Override
```

CONGESTION IN THE SKY

```

        public String toString() {
            return "AverageCountTuple{" + "Total Flights=" + flightCount + ",
Total Distance=" + distCount
                + ", Total Air Time=" + airTime + ", Average
Distance=" + String.format("%.2f", averageDist)
                + ", Average Air Time=" + String.format("%.2f",
averageAirTime) + '}';
        }

        @Override
        public void write(DataOutput dataOutput) throws IOException {
            dataOutput.writeInt(flightCount);
            dataOutput.writeInt(distCount);
            dataOutput.writeInt(airTime);
            dataOutput.writeDouble(averageDist);
            dataOutput.writeDouble(averageAirTime);
        }

        @Override
        public void readFields(DataInput dataInput) throws IOException {

            flightCount = dataInput.readInt();
            distCount = dataInput.readInt();
            airTime = dataInput.readInt();
            averageDist = dataInput.readDouble();
            averageAirTime = dataInput.readDouble();
        }
    }
}

--

package com.kinnar.bigdataproject.avg_dist_carrier;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;

public class AverageMapper extends Mapper<Object, Text, Text,
AverageCountTuple> {

    private AverageCountTuple tuple = new AverageCountTuple();

    @Override
    protected void map(Object key, Text value, Context context) throws
IOException, InterruptedException {
        String[] tokens = value.toString().split(",");

        if (tokens[0].equals("Year"))
            return;

        String carrier = tokens[8];
        int dist = 0;
        int flightTime = 0;

```

CONGESTION IN THE SKY

CONGESTION IN THE SKY

```

        try {
            dist = Integer.parseInt(tokens[18]);

            flightTime = Integer.parseInt(tokens[6]);
        } catch (Exception e) {
            return;
        }
        tuple.setFlightCount(1);
        tuple.setDistCount(dist);
        tuple.setAirTime(flightTime);

        context.write(new Text(carrier), tuple);
    }
}
--

package com.kinnar.bigdataproject.avg_dist_carrier;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class AverageCombiner extends Reducer<Text, AverageCountTuple, Text,
AverageCountTuple> {

    private AverageCountTuple tuple = new AverageCountTuple();

    @Override
    protected void reduce(Text key, Iterable<AverageCountTuple> values,
Context context)
        throws IOException, InterruptedException {

        int totalFlight = 0;
        int totalDist = 0;
        int totalAirTime = 0;

        for (AverageCountTuple dt : values) {
            totalFlight += dt.getFlightCount();
            totalDist += dt.getDistCount();
            totalAirTime += dt.getAirTime();
        }

        tuple.setAirTime(totalAirTime);
        tuple.setDistCount(totalDist);
        tuple.setFlightCount(totalFlight);

        context.write(key, tuple);
    }
}
--

package com.kinnar.bigdataproject.avg_dist_carrier;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class AverageReducer extends Reducer<Text, AverageCountTuple, Text,
AverageCountTuple> {

```

CONGESTION IN THE SKY

```

private AverageCountTuple tuple = new AverageCountTuple();

@Override
protected void reduce(Text key, Iterable<AverageCountTuple> values,
Context context)
    throws IOException, InterruptedException {

    int totalFlight = 0;
    int totalDist = 0;
    int totalAirTime = 0;

    for (AverageCountTuple dt : values) {
        totalFlight += dt.getFlightCount();
        totalDist += dt.getDistCount();
        totalAirTime += dt.getAirTime();
    }

    double avgDist = (double) totalDist / totalFlight;
    double avgAirTime = (double) totalAirTime / totalFlight;

    tuple.setAirTime(totalAirTime);
    tuple.setDistCount(totalDist);
    tuple.setFlightCount(totalFlight);
    tuple.setAverageDist(avgDist);
    tuple.setAverageAirTime(avgAirTime);

    context.write(key, tuple);
}
--

package com.kinnar.bigdataproject.avg_dist_carrier;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import java.io.IOException;
import java.net.URI;

public class AverageMain {

    public static void main(String[] args) throws IOException,
InterruptedException, ClassNotFoundException {

        Configuration conf = new Configuration();

        FileSystem hdfs =
FileSystem.get(URI.create("hdfs://localhost:9000"), conf);

        Path output = new Path(args[1]);
        // delete existing directory
        if (hdfs.exists(output)) {
            hdfs.delete(output, true);
        }
    }
}

```


CONGESTION IN THE SKY

```
// Create a new Job
Job job = Job.getInstance(conf, "wordcount");
job.setJarByClass(AverageMain.class);

// Specify various job-specific parameters
job.setJobName("myjob");

FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, output);

job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);

job.setMapOutputKeyClass(Text.class);
job.setMapOutputValueClass(AverageCountTuple.class);

job.setMapperClass(AverageMapper.class);
job.setCombinerClass(AverageCombiner.class);
job.setReducerClass(AverageReducer.class);

job.setOutputKeyClass(Text.class);
job.setOutputValueClass(AverageCountTuple.class);

// Submit the job, then poll for progress until the job is
complete
System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

11.7. Recommendation System using RMS (Root Mean Square)

```
package com.kinnar.bigdataproject.rms_carrier;

import org.apache.hadoop.io.Writable;
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

public class RMSCountTuple implements Writable {

    private int arrDelay = 0;
    private int depDelay = 0;
    private int totalFlight = 0;
    private double rms = 0.0;

    public int getArrDelay() {
        return arrDelay;
    }

    public void setArrDelay(int arrDelay) {
        this.arrDelay = arrDelay;
    }

    public int getDepDelay() {
        return depDelay;
    }

    public void setDepDelay(int depDelay) {
        this.depDelay = depDelay;
    }

    public int getTotalFlight() {
        return totalFlight;
    }

    public void setTotalFlight(int totalFlight) {
        this.totalFlight = totalFlight;
    }

    public double getRms() {
        return rms;
    }

    public void setRms(double rms) {
        this.rms = rms;
    }

    // Returning just rms value for recommendation system. If just to show
the
    // result use the below return
    // @Override
    // public String toString() {
    //     return "{" +
    //         "arrDelay=" + arrDelay +
    //         ", depDelay=" + depDelay +
    //         ", totalFlight=" + totalFlight +
    //         ", rms=" + String.format("%.4f", rms) +
    //         '}';
    // }
```

CONGESTION IN THE SKY

```
// }

@Override
public String toString() {
    return String.format("%.4f", rms);
}

@Override
public void write(DataOutput dataOutput) throws IOException {
    dataOutput.writeInt(arrDelay);
    dataOutput.writeInt(depDelay);
    dataOutput.writeInt(totalFlight);
    dataOutput.writeDouble(rms);
}

@Override
public void readFields(DataInput dataInput) throws IOException {
    arrDelay = dataInput.readInt();
    depDelay = dataInput.readInt();
    totalFlight = dataInput.readInt();
    rms = dataInput.readDouble();
}
}
--

package com.kinnar.bigdataproject.rms_carrier;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import java.io.IOException;

public class RMSMapper extends Mapper<Object, Text, Text, RMSCountTuple> {

    private RMSCountTuple tuple = new RMSCountTuple();

    @Override
    protected void map(Object key, Text value, Context context) throws
IOException, InterruptedException {
        String[] tokens = value.toString().split(",");

        if (tokens[0].equals("Year"))
            return;

        String src = tokens[16];
        String dest = tokens[17];
        String carrier = tokens[8];

        int arrDelay = 0;
        int depDelay = 0;

        try {
            arrDelay = Integer.parseInt(tokens[14]);
            depDelay = Integer.parseInt(tokens[15]);
        } catch (Exception e) {
        }

        String newKey = src + "-" + dest + "\\t" + carrier;
```

CONGESTION IN THE SKY

```
        tuple.setArrDelay(arrDelay);
        tuple.setDepDelay(depDelay);
        tuple.setTotalFlight(1);

        context.write(new Text(newKey), tuple);
    }
}
--

package com.kinnar.bigdataproject.rms_carrier;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
import java.io.IOException;

public class RMSCombiner extends Reducer<Text, RMSCountTuple, Text,
RMSCountTuple> {

    private RMSCountTuple res = new RMSCountTuple();

    @Override
    protected void reduce(Text key, Iterable<RMSCountTuple> values,
Context context)
        throws IOException, InterruptedException {

        int total = 0;
        int arrDelay = 0;
        int depDelay = 0;

        for (RMSCountTuple tup : values) {
            total += tup.getTotalFlight();
            arrDelay += tup.getArrDelay();
            depDelay += tup.getDepDelay();
        }

        res.setTotalFlight(total);
        res.setArrDelay(arrDelay);
        res.setDepDelay(depDelay);

        context.write(key, res);
    }
}
--

package com.kinnar.bigdataproject.rms_carrier;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
import java.io.IOException;

public class RMSReducer extends Reducer<Text, RMSCountTuple, Text,
RMSCountTuple> {

    private RMSCountTuple res = new RMSCountTuple();

    @Override
    protected void reduce(Text key, Iterable<RMSCountTuple> values,
Context context)
        throws IOException, InterruptedException {
```

CONGESTION IN THE SKY

```

int total = 0;
int arrDelay = 0;
int depDelay = 0;

for (RMSCountTuple tup : values) {
    total += tup.getTotalFlight();
    arrDelay += tup.getArrDelay();
    depDelay += tup.getDepDelay();
}

double avgArrDelay = (double) arrDelay / total;
double avgDepDelay = (double) depDelay / total;

double rms = Math.sqrt((avgArrDelay * avgArrDelay) +
    (avgDepDelay * avgDepDelay));

res.setTotalFlight(total);
res.setArrDelay(arrDelay);
res.setDepDelay(depDelay);
res.setRms(rms);

context.write(key, res);
}
}
--

package com.kinnar.bigdataproject.rms_carrier;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import java.io.IOException;
import java.net.URI;

public class RMSMain {

    public static void main(String[] args) throws IOException,
        InterruptedException, ClassNotFoundException {

        Configuration conf = new Configuration();

        FileSystem hdfs =
        FileSystem.get(URI.create("hdfs://localhost:9000"), conf);

        Path output = new Path(args[1]);
        // delete existing directory
        if (hdfs.exists(output)) {
            hdfs.delete(output, true);
        }

        // Create a new Job
        Job job = Job.getInstance(conf, "wordcount");
        job.setJarByClass(RMSMain.class);

        // Specify various job-specific parameters

```

CONGESTION IN THE SKY

```

        job.setJobName("myjob");

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, output);

        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);

        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(RMSCountTuple.class);

        job.setMapperClass(RMSMapper.class);
        job.setCombinerClass(RMSCombiner.class);
        job.setReducerClass(RMSReducer.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(RMSCountTuple.class);

        // Submit the job, then poll for progress until the job is
complete
        System.exit(job.waitForCompletion(true) ? 0 : 1);

    }
}
--

package com.kinnar.bigdataproject.recommendation_sys;

import org.apache.hadoop.io.WritableComparable;
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

public class CompositeKey implements WritableComparable<CompositeKey> {

    private String srcDest;
    private String carrierInfo;

    public CompositeKey() {
        super();
    }

    public String getSrcDest() {
        return srcDest;
    }

    public void setSrcDest(String srcDest) {
        this.srcDest = srcDest;
    }

    public String getCarrierInfo() {
        return carrierInfo;
    }

    public void setCarrierInfo(String carrierInfo) {
        this.carrierInfo = carrierInfo;
    }

    public CompositeKey(String srcDest, String carrierInfo) {
        this.srcDest = srcDest;
        this.carrierInfo = carrierInfo;
    }
}

```

CONGESTION IN THE SKY

```

}

@Override
public void write(DataOutput d) throws IOException {
    d.writeUTF(srcDest);
    d.writeUTF(carrierInfo);
}

@Override
public void readFields(DataInput di) throws IOException {
    srcDest = di.readUTF();
    carrierInfo = di.readUTF();
}

@Override
public int compareTo(CompositeKey o) {
    int result = this.srcDest.compareTo(o.getSrcDest());
    if (result == 0) {
        String c1 = this.carrierInfo;
        Double rms1 = Double.parseDouble(c1.split("\\t")[1]);

        String c2 = o.getCarrierInfo();
        Double rms2 = Double.parseDouble(c2.split("\\t")[1]);
        return rms1.compareTo(rms2);
    }

    return result;
}

@Override
public String toString() {
    return srcDest + " : " + carrierInfo;
}
}
--

package com.kinnar.bigdataproject.recommendation_sys;

import org.apache.hadoop.io.WritableComparator;

public class GroupComparator extends WritableComparator {

    protected GroupComparator() {
        super(CompositeKey.class, true);
    }

    @Override
    public int compare(Object a, Object b) {

        CompositeKey ckw1 = (CompositeKey) a;
        CompositeKey ckw2 = (CompositeKey) b;

        return ckw1.getSrcDest().compareTo(ckw2.getSrcDest());
    }
}
--

package com.kinnar.bigdataproject.recommendation_sys;

import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.io.WritableComparator;

```

CONGESTION IN THE SKY

```

public class SecondarySortComparator extends WritableComparator {

    protected SecondarySortComparator() {
        super(CompositeKey.class, true);
    }

    @SuppressWarnings("rawtypes")
    @Override
    public int compare(WritableComparable a, WritableComparable b) {

        CompositeKey ck1 = (CompositeKey) a;
        CompositeKey ck2 = (CompositeKey) b;

        int result = ck1.getSrcDest().compareTo(ck2.getSrcDest());

        if (result == 0) {
            String c1 = ck1.getCarrierInfo();
            Double rms1 = Double.parseDouble(c1.split("\t")[1]);

            String c2 = ck2.getCarrierInfo();
            Double rms2 = Double.parseDouble(c2.split("\t")[1]);
            result = rms1.compareTo(rms2);
        }

        return result;
    }
}

--

package com.kinnar.bigdataproject.recommendation_sys;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import java.io.IOException;

public class SecondarySortMapper extends Mapper<LongWritable, Text,
CompositeKey, NullWritable> {

    @Override
    protected void map(LongWritable key, Text value, Context context)
    throws IOException, InterruptedException {
        // To change body of generated methods, choose Tools |
        Templates.

        String[] tokens = value.toString().split("\t", 2);

        try {
            String srcDest = tokens[0];
            String carrInfo = tokens[1];

            CompositeKey coKey = new CompositeKey(srcDest, carrInfo);

            context.write(coKey, NullWritable.get());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```


CONGESTION IN THE SKY

```

    }

}
--

package com.kinnar.bigdataproject.recommendation_sys;

import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.mapreduce.Partitioner;

public class KeyPartition extends Partitioner<CompositeKey, NullWritable> {

    @Override
    public int getPartition(CompositeKey key, NullWritable value, int
numPartitions) {

        return key.getSrcDest().hashCode() % numPartitions;

    }

}
--

package com.kinnar.bigdataproject.recommendation_sys;

import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.mapreduce.Reducer;
import java.io.IOException;

public class SecondarySortReducer extends Reducer<CompositeKey,
NullWritable, CompositeKey, NullWritable> {

    @Override
    protected void reduce(CompositeKey key, Iterable<NullWritable>
values, Context context)
        throws IOException, InterruptedException {
        // To change body of generated methods, choose Tools |
Templates.

        for (NullWritable v : values) {
            context.write(key, v);
        }

    }

}
--

package com.kinnar.bigdataproject.recommendation_sys;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import java.io.IOException;
import java.net.URI;

```

CONGESTION IN THE SKY

```
public class SecondarySortDriver {

    public static void main(String[] args) throws IOException,
ClassNotFoundException, InterruptedException {

        Configuration conf = new Configuration();
        FileSystem hdfs =
FileSystem.get(URI.create("hdfs://localhost:9000"), conf);

        Path outDir = new Path(args[1]);
        // delete existing directory
        if (hdfs.exists(outDir)) {
            hdfs.delete(outDir, true);
        }

        Job job = Job.getInstance();

        job.setJarByClass(SecondarySortDriver.class);

        job.setGroupingComparatorClass(GroupComparator.class);
        job.setSortComparatorClass(SecondarySortComparator.class);
        job.setPartitionerClass(KeyPartition.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        // Path outDir = new Path(args[1]);
        FileOutputFormat.setOutputPath(job, outDir);

        job.setMapperClass(SecondarySortMapper.class);
        job.setReducerClass(SecondarySortReducer.class);

        job.setNumReduceTasks(1);

        job.setOutputKeyClass(CompositeKey.class);
        job.setOutputValueClass(NullWritable.class);

        job.waitForCompletion(true);
    }
}
```