A8 Report

Bishwajeet Dey

November 14, 2017

1. Introduction

This report discusses the design decisions and the corresponding runtimes for solving A4 - On-Time Performance Data.

2. Environment

Table 2.a Local Mode Environment

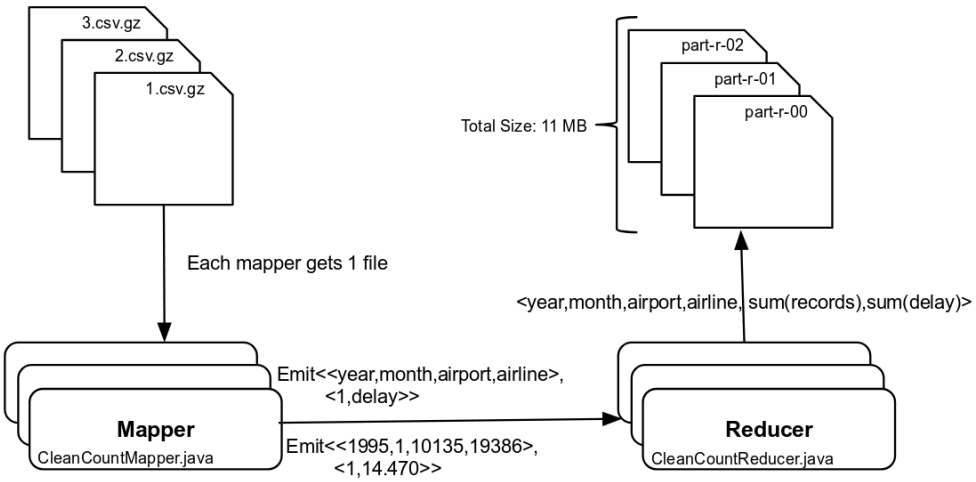
Property	Value
java-version	1.8.0_101
OS	GNU/Linux
arch	x86_64
cores	2
threads-per-core	2
model	Intel(R)Xeon(R)CPUE5-2686v4@2.30GHz
Memory	16.17398G
Disk	SSD

Table 2.b AWS Environment

Property	Value
OS	GNU/Linux
arch	x86_64
Disk	SSD
master-node	1
core-node	2
machine-type	m3.xlarge

Complete instance information can be found at the official site - https://aws.amazon.com/ec2/instance-types/

3. Architecture



- The program starts from `Boot.java`. The program uses only 1 job.
- The class which does sanitization of each record is located in `Sanitizer.java`. It returns a record `<year, month, airport, airline, normalized-delay>` wrapped in an object of type `SanitizedRecord` when the record is valid. Record splitting uses `CSVParser`. The delay is normalized as `arrivalDelayMin/CSRElapsedTime` for flights which were not cancelled. For cancelled flights, the normalized delay is `4.0`.
- **CleanCountMapper** receives each record from a file. When a valid record is found, it emits `<<year, month, airport, airline>, <1, normalized-delay>>`. I restricted the mapper to receive each input file per map task since `gz` format is unsplittable and most of the files expand to sizes of around 150 MB which is neither too large or small for a mapper.
- **CleanCountReducer** receives the record key as `<year, month, airport, airline>` from the mapper and aggregates the value as total counts and delays for that month. Thus, for a given airport and airline, it has aggregated the number of flights and the corresponding aggregated normalized delay. There is no limit on the number of reducers.

- The total size of all the output files after running the job is 11 MB which is small enough to be loaded in R. Further processing is carried out in R.

4. Analysis

Table 4.a Top 5 Active airports

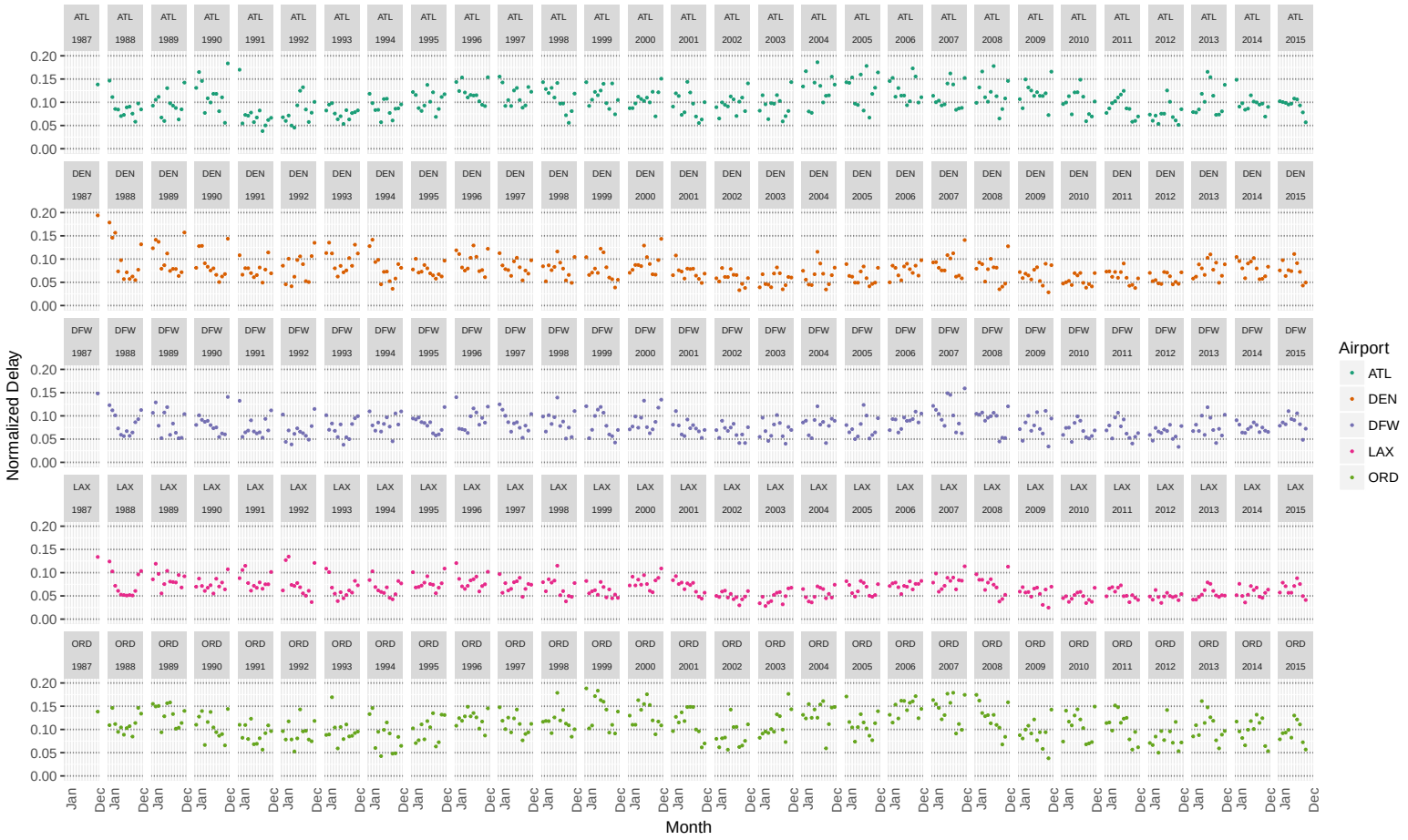
id	code	airport
10397	ATL	Hartsfield-Jackson Atlanta International
13930	ORD	Chicago O'Hare International
11298	DFW	Dallas/Fort Worth International
12892	LAX	Los Angeles International
11292	DEN	Denver International

Table 4.b Top 5 Active Airlines

id	code	airline
19393	WN	Southwest Airlines Co.
19790	DL	Delta Air Lines Inc.
19805	AA	American Airlines Inc.
20355	US	US Airways Inc.
19977	UA	United Air Lines Inc.

4.1 Normalized delay of Top 5 Airports

Fig 4.1 Normalized delay of top 5 airports from 1987−2015

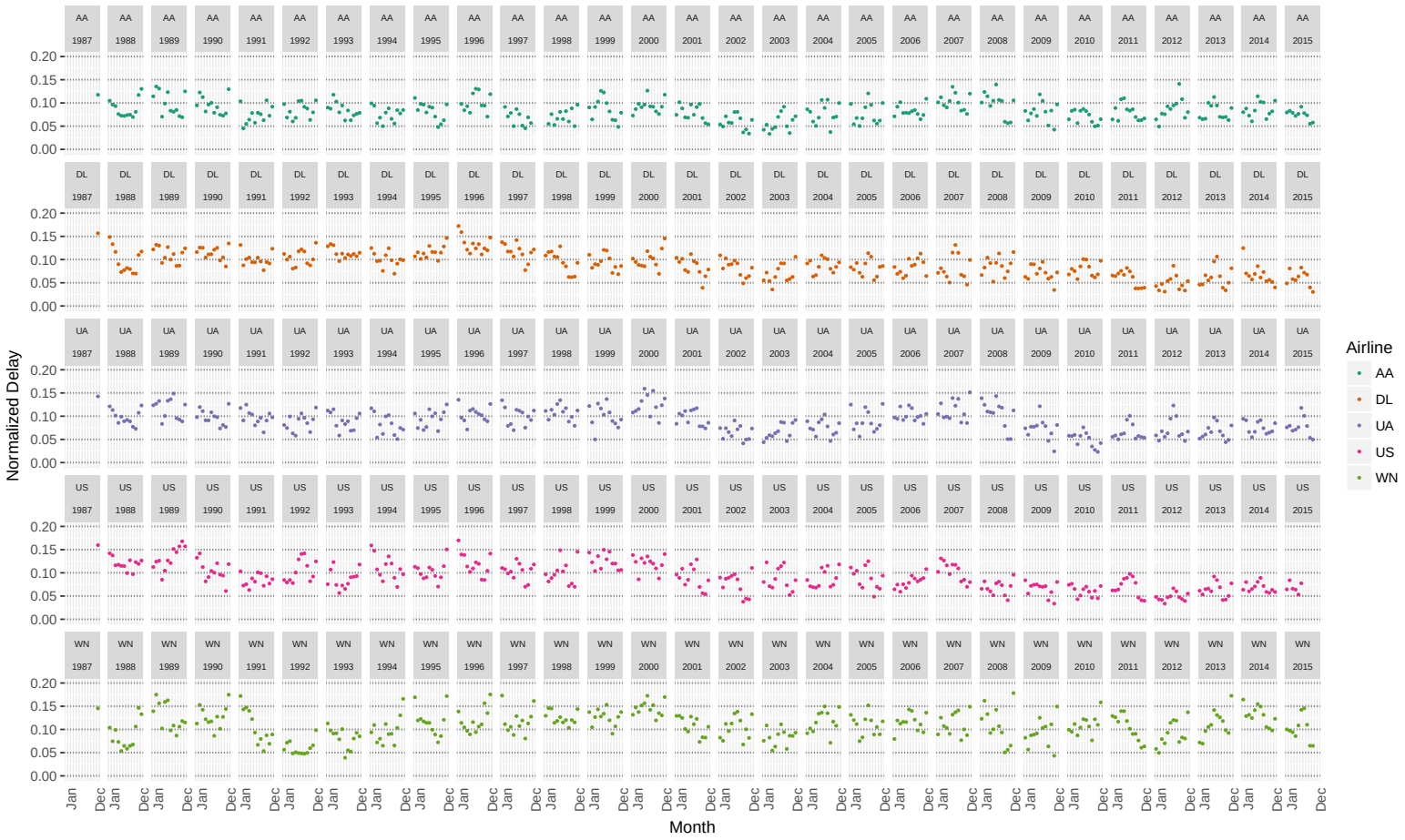


The plot in `Fig 4.1` shows the distribution of mean delay of each month from 1987-2015 for the top 5 airports. The x-axis shows months for the past 30 years. Months are grouped into years. The y-axis has the normalized delay for each month. Each point in the graph represents the median delay of that airport across all airlines.

From the `Fig 4.1` plot, it is clear that the delays across airports have reduced. Among the top 5 airports, there seem to be heavy delays for most of the months in the year - 2000. Post 2000, all airport delays are smaller as compared to pre 2000. This is clear in the graphs of airports like Los Angeles and Denver where the hurricane months of June, July and August do not seem to have much effect on the overall normalized delay of each month.

4.2 Normalized delay of Top 5 Airlines

Fig 4.2 Normalized delay of top 5 airlines from 1987−2015



The plot in `Fig 4.2` shows the distribution of mean delay of each month from 1987-2015 for the top 5 airlines. The x-axis shows months for the past 30 years. Months are grouped into years. The y-axis has the normalized delay for each month. Each point in the graph represents the median delay of that airline across all airports.

From the `Fig 4.2` plot, it is clear that there is a gradual reduction in airline delays over the past 30 years. From the year 2000 onward, the airline delays were comparatively lower.

4.3 Distribution of delay across Top 5 Airports and Top 5 Airlines

Fig 4.3 Nomalized delay distribution of top 5 airports from 1987

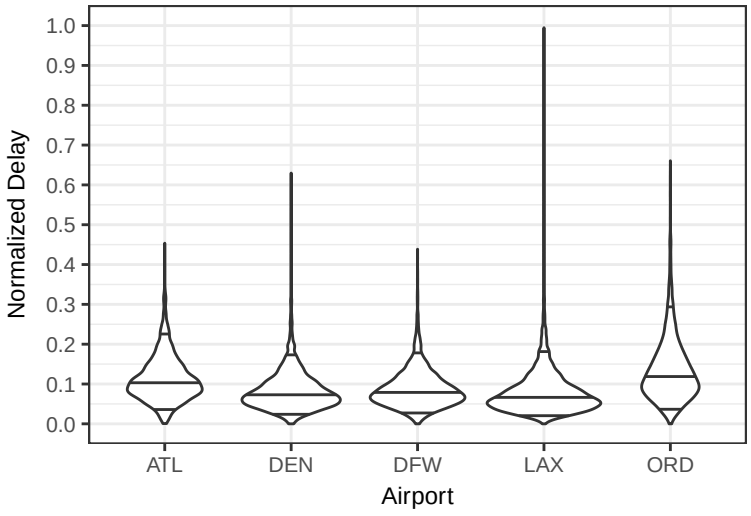
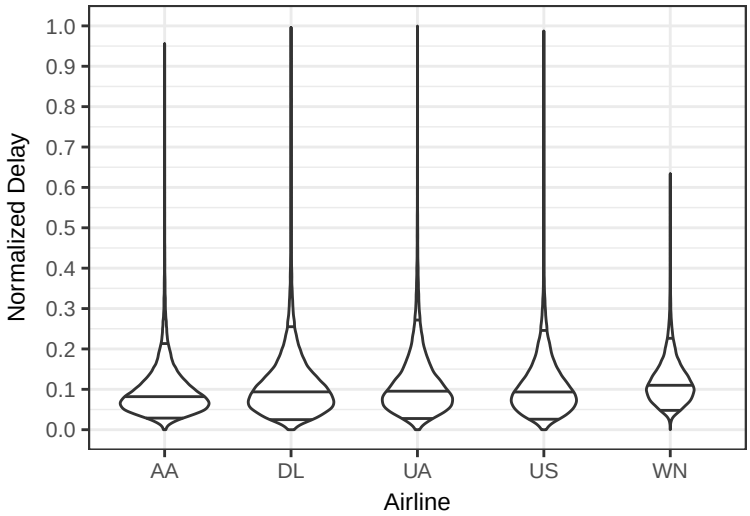


Fig 4.4 Normalized delay of top 5 airlines from 1987−2015



The graph in `Fig 4.3` plots the distribution of normalized delays for the top 5 airports The lines within each of the violin plot represent the quantiles at [**0.05, 0.50, 0.95**]. The width of the violin plot is proportional to the number of points which have the same normalized

delay. The y-axis has the normalized delay. So, each point in the distribution is the monthly mean for that airline and airport. Delay is clipped at **1.0** which leads led to loss of less than 0.001% of points.

This plot helps us correlate the actual distribution with the plot from `Fig 4.1`. Consider an airport like **LAX** whose median delay from `Fig 4.3` is the lowest. However from this plot, it is clear that the bottom 5 percentile of delays is worse than other airports. **ORD** has consistent delays across the whole delay spectrum. The median delay at **ORD** is much worse than the rest of the top 5 airports. Overall, **DFW** shows comparatively smaller deviation of mean delays as compared to the rest of the top airports.

The graph in `Fig 4.4` plots the distribution of normalized delays for the top 5 airlines The lines within each of the violin plot represent the quantiles at [**0.05, 0.50, 0.95**]. The width of the violin plot is proportional to the number of points which have the same normalized delay. The y-axis has the normalized delay. So, each point in the distribution is the monthly mean for that airline and airport. Delay is clipped at **1.0** which leads led to loss of less than 0.001% of points.

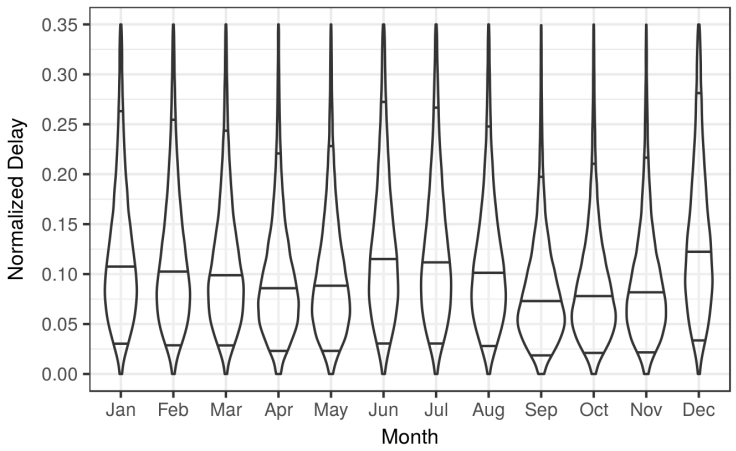
This plot helps us correlate the distribution of points which we saw in plot `Fig 4.2`. From the plot, we realize that the jitter which we saw in **Southwest Airlines** plot is consistent with `Fig 4.4`. Inspite of flying to fewer destinations, **SouthWest Airlines** beats other airlines in the number of flights, has worse yet consistent delays. **American Airlines** seem to have better normalized delays as compared to the rest of the top airlines.

4.4 Seasonal Trends

The graph in `Fig 4.5` plots a distribution of normalized monthly delay of flights for the whole dataset grouped by month. So, on the x-axis we have month and on the y-axis - normalized delay. The width of the violin plot is proportional to the number of points which have the same normalized delay. The lines within each of the violin plot represent the quantiles at [**0.05, 0.50, 0.95**]. The normalized delay is clipped at `0.35`. Less than 0.03% of the points were clipped in `Fig 4.5`.

September seems to have the lowest median delay followed by **October**. **December** has the worst delays because of the holiday season. The hurricane season effects the flights during the months of **June, July, August**. The median delay as well as the distribution during the hurricane season closely matches with **January**, which catches the tail of the holiday season.

Fig 4.5 Seasonal Normalized Mean Delay Distribution from 1987-2015

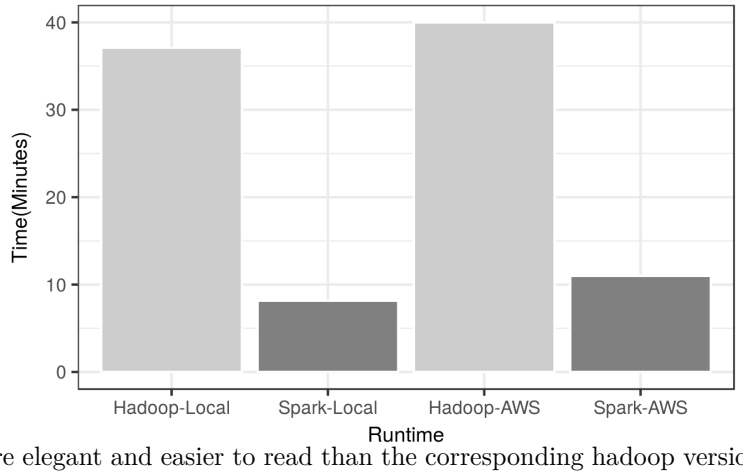


4.5 Execution Times

The graph in `Fig 4.6` plots the runtimes of the program with 2 different frameworks - Hadoop and Spark. On the x-axis, we have the runtimes of Hadoop and Spark on local as well as AWS. The y-axis records the execution time in minutes.

From the graph, we conclude that Spark is 4`X` faster than Hadoop in local and AWS. Spark in local mode did not shuffle data because the output key contains a tuple of `<year, month, airport, airline>` which can be computed on a per file basis. Thus, in local I had 330+ files. The local partition results were merged at the `Driver`. At AWS-Spark, 33 files were obtained which suggests a shuffle & merge for the final output. Spark reduces locally before sending data across partitions. Hadoop spends significant time spilling out the results. It finally does a reduce which leads to a lot of data sent across to the reducer. The scala code for Spark is more elegant and easier to read than the corresponding hadoop version.

Fig 4.6 Comparision of runtimes - Hadoop Vs Spark



5. Conclusion

This report presented the analysis of the On Time performance dataset. We analyzed the trend in delays from 1987-2015. After visualizing the trend, we came to the conclusion that the delay across airports and airlines have gradually decreased. The distribution of delays from the top 5 airports and airlines helped us visually compare delays across airports and airlines. The seasonal trend analysis showed that delays affect all airlines and airports due to factors like weather and holidays. Finally, frameworks like Spark can drastically reduce the time spent for cleaning/summarizing the input data for the next step.