

Animated visualization of flights using ADS-B messages

Thanh Long Tran
Vrije Universiteit Amsterdam
t.l.tran@student.vu.nl

ABSTRACT

More and more aircraft are adapting the ADS-B technology making it possible to track flights more accurately. Enthusiasts all around the world collect these messages transmitted by aircraft making it available to the public. The sheer size of the information that is produced by planes this way is enormous. The arrival of Big Data era has made it possible to process such a large dataset in a relatively short amount of time. The purpose of this paper is to create an interactive animated visualization using the ADS-B message dataset provided by OpenSky Network. It walks through the steps of analyzing and processing the dataset to prepare it for the animation. To address this challenge methods using Big Data tools on a large Hadoop cluster is presented. The pipeline that the data is going through contains several algorithms that processes it. The paper details how the Ramer-Douglas-Peucker and other simple algorithms can be used to greatly reduce the size of the data without losing important information. The final product is a web based application that uses the processed data to create an interactive animated flight visualization.

Keywords

Flight visualization, Big Data, Ramer-Douglas-Peucker, ADS-B

1. INTRODUCTION

Due to the number of airborne aircraft at a given time reaching tens of thousands nowadays the processing power required to track these flights is getting higher as well. The automatic dependent surveillance – broadcast (ADS-B) is a surveillance technology that is used for tracking flights by periodically broadcasting information about the aircraft such as its position. This technology is more accurate than the traditional radar technology and allows the broadcast additional information.

ADS-B is a relatively new technology and the aircraft are in the process of adopting it. From 2017 onwards, the use of a ADS-B transmitter is mandatory for most of the aircraft in Europe and the United State requires some aircraft to equip the technology by 2020. There are clear benefits to switching to ADS-B from Radar technologies. The pilot will be more

aware if the aircraft's surrounding as it is able to receive traffic, weather and flight information. In addition, the messages broadcasted by the transmitters are unencrypted, which means that anyone with the appropriate sensor can record the messages and decode them.

With the high amount of airborne aircraft in the sky and the high frequency of broadcasts by a single transmitter the number of messages that are broadcasted by planes is enormous. Processing these messages requires an equally huge amount of compute power.

The OpenSky Network is a community-based receiver network that collects ADS-B messages from more than 750 sensors around the world. I have acquired a small subset of their full message dataset. This includes all the messages from the 18th of September 2016 to the 24th of September 2016. This is roughly 590 Gigabytes of compressed messages.

This paper aims to provide an insight into how one could use Big Data technologies to process this large dataset on a powerful cluster. In this project I have created an interactive animated visualization of the flights and explore the steps taken to process the data and achieve the end goal.

2. RELATED WORK

2.1 Flight visualization

Multiple animated flight visualizations have been created before. These visualizations were either videos or real-time flight trackers. Both NATS¹ and ItoWorld² have created a rather spectacular video of the European flights, but since these are static videos they lack interactivity. The FlightRadar24³ website offers a very powerful live flight tracking service. It provides real-time information about thousands of aircraft using ADS-B messages. The aim of this project is to create a spectacular visualization of the flights while providing interactivity with the time and the flights as well.

Christopher Hurter et al. present novel methods for big data exploration and analysis of the Air Traffic Control (ATC) domain. [1]. They have developed a visualization tool called FromDaDy⁴ that creates interactive visualizations of numerous aircraft trajectories. This powerful tool uses image

¹ <https://nats.aero/blog/2014/03/europe-24-air-traffic-data-visualisation/>

² <https://vimeo.com/11739091>

³ <https://www.flightradar24.com/>

⁴ FROM DATA to Display

based visual analytics technologies to explore ATC datasets. They are giving example queries such as finding overseas flights or finding standard procedures and showing how it can be done in a matter of minutes.

They also present density-map techniques to reduce clutter in the dataset and therefore increase visual scalability. The graph-based technique call graph bundling is a powerful data aggregation technique that they are using to cluster the flights instead of showing the original flight path.

Three different possible use cases of animation to support the visual analysis of air-traffic datasets is also presented. The key idea of the focus-and-context technique is to deform and distort the visualization locally to reveal hidden information. The KDEEB algorithm is a time dependent dynamic bundling algorithm which allows is to recognize the connectivity pattern between US cities. Finally, flow visualization is the one that is the most closely related to this paper. This provides a local insight into the flight patterns by visualizing the fine-grained information. Further tuning the parameters of this visualization can help put an emphasis on different aspects of the dataset. The goal of this paper is to create a similar visualization.

2.2 Douglas-Peucker algorithm

The project that this paper covers mostly relies on the Ramer-Douglas-Peucker algorithm (or Douglas-Peucker) which can be used to find a similar curve of a given set of points. While the original algorithm works very well there have been multiple attempts to optimize the algorithm.

Jon Vaughan et al. propose three parallel implementations of the Douglas-Peucker algorithm using multitasking techniques on a Sequent Symmetry computer [2]. They are comparing all the parallel implementations to the original sequential algorithm and to each other.

The first algorithm relies on parallelizing the calculation of the maximum offset for the current line segment. The second implementation parallelizes the processing of separate line segments. Both algorithms introduced some kind of load imbalance on the processors. The former algorithm was efficient at the beginning of the process while the latter was more efficient at the end. The third implementation is the combination of the two.

The results of the tests performed show that all of the optimized algorithms yielded improvements. Particularly, the third implementation can achieve a speed up of 7. The improvement the algorithm achieved is remarkable, but unfortunately cannot be applied to the cluster that is available.

3. RESEARCH QUESTIONS

The main goal of this project is to create an animated visualization of flights while also providing interactivity. To achieve this goal, I am going to use the raw ADS-B messages provided by OpenSky encoded in AVRO format. In order to process all this data, I need to utilize large scale infrastructures and technologies. This raises some questions and problems regarding the project.

- How to use the raw ADS-B data to extract information about the flights?
- How to identify the flights from the raw ADS-B data?
- This is an interactive animation that should be able to run on weaker computers as well. How to reduce the size of the dataset without losing too much information about the flights?
- How to animate the visualization of the flights without the need to use a lot of resources?

The end result will be a web based application that renders the animated visualization of the flights and allows interaction to a certain degree.

4. PROJECT SETUP

4.1 Technology

Analyzing and processing such a large dataset requires a lot of compute power. With this in mind, I have to choose the appropriate tools to tackle this project. Fortunately, I was given the opportunity to do my research work on the large Hadoop cluster of SurfSARA⁵. However, it is essential to make things work in a local environment first before going big in the cluster.

For large-scale data processing on the cluster I chose to use Apache Spark⁶ engine for Java because of its sheer performance and easy-to-use application programming interface. In addition, Spark has support for the AVRO file format in which the flight messages are stored on the cluster.

The messages are stored as raw data which need to be decoded before I could start processing them. The java-adsb⁷ java library is developed by OpenSky and provides a convenient way to decode these raw messages to a usable format. With the combination of Spark and java-adsb, the dataset can be quickly decoded.

In order to gain a deeper insight into the dataset itself one would create plots and graphs to visualize the data and analyze it. The matplotlib⁸ Python plotting library can quickly produce rich quality figures. I used this library to

⁵ <https://userinfo.surfsara.nl/>

⁶ <https://spark.apache.org/>

⁷ <https://github.com/openskynetwork/java-adsb>

⁸ <https://matplotlib.org/>

plot flights to analyze its properties, find irregular data and do experimentations with the algorithms.

For the visualization itself the D3.js⁹ JavaScript library most suitable. D3.js is an incredibly fast data-driven library for creating rich web based data visualizations. More importantly the d3-geo-projections module provides a simple and powerful interface to project spherical coordinates into a flat map.

4.2 Understanding the data

Before I can effectively start working on the algorithms it is essential to understand what it is exactly that I am working with. The dataset consists of almost 8.4 billion messages and to start analyzing it I first downloaded a small fraction of the dataset onto my local computer and inspected it.

Table 1. The number of messages of each type

Message Type	Count
ADSB_AIRBORN_POSITION	574810807
ADSB_AIRSPEED	1823438
ADSB_EMERGENCY	5169223
ADSB_IDENTIFICATION	63201928
ADSB_STATUS	20442848
ADSB_SURFACE_POSITION	4414411
ADSB_TCAS	5106
ADSB_VELOCITY	554039575
ALL_CALL_REPLY	2799627967
ALTITUDE_REPLY	941869748
COMM_B_ALTITUDE_REPLY	932401372
COMM_B_IDENTIFY_REPLY	379043405
COMM_D_ELM	21759
EXTENDED_SQUITTER	45933229
IDENTIFY_REPLY	366631994
LONG_ACAS	115357637
MILITARY_EXTENDED_SQUITTER	24882577
MODES_REPLY (UNKNOWN)	22564
SHORT_ACAS	1565463515

I first sampled the first 10 Gigabytes of messages. At first glance it was immediately clear, that not all the messages contain useful information for this project. After that I created a short statistic on the number of messages, which shown in Table 1. Out of these messages the airborne and surface positions messages contain the position information which are need for the animated visualization. Furthermore, the ADSB_IDENTIFICATION messages can be used to identify the airlines of the flights, which enables flight filtering and

interactivity. These messages only make up about 7% of the whole dataset and the rest can be practically considered useless data and can be thrown out.

It also became apparent when analyzing the data that the messages are not necessarily ordered by time, because all the flights in the 10 Gigabyte sample data were lacking in information. Therefore, I instead extracted all the position messages of the transmitter 3c674f and worked in this set of messages in the local environment. All the positions broadcasted by this transmitter is plotted in Figure 1.

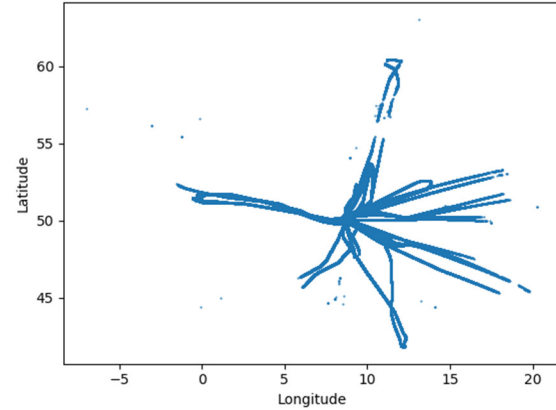


Figure 1. All the position messages of the aircraft with ICAO24 3c674f

4.3 Overview

There are two bigger parts of the project. The data processing phase, whose aim is to reduce and restructure the dataset into a smaller format which can be used for the animation. This phase consists of a pipeline of data processing elements. Each element is a Spark application usually run on the cluster unless the dataset is small enough to run on the local environment. The pipeline consists of the following elements:

1. **Position extractor:** as discussed in section 4.2 I only need a small fraction of the whole dataset. This application extracts all the position messages from the dataset.
2. **Unrealistic filter:** as we will see later, there is a considerable amount of noise in the dataset. This application tries to filter this noise out.
3. **RDP reducer:** this application applies the Ramer-Douglas-Peucker algorithm to the flights.
4. **Flight splitter:** The path of the flight is interpolated over the positions points. The final visualization application needs to know when the flight ends and this application attempts splits the positions into distinct flights.

⁹ <https://d3js.org/>

5. **Csv to json**; The final visualization will work with json data. This application will covert and restructure the data.

Additionally, the last element will also take in the output of another Spark application that extracts the call signs from the dataset. The call sign messages are used to identify the flights and the airlines they belong to. The outline of the pipeline is shown on Figure 2.

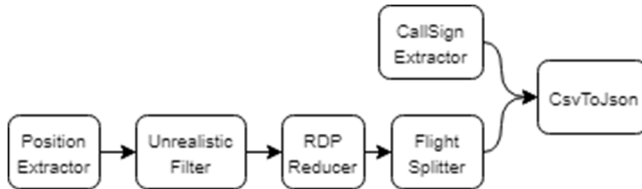


Figure 2. The Spark application pipeline

4.4 Extracting the data

We saw in section 4.2 that only a small fraction of the whole dataset is useful for this project. The java-adsb library makes it easy to decode and tell the types of the messages. This information is used to filter out the unnecessary data and only keep the positions messages.

One would expect that a single position information is stored within a single ADS-B message but that is not the case. There are two types of positions messages: odd and even. These messages need to be read and decoded in the correct order to successfully extract the position information. This is called the compact position reporting¹⁰ (CPR) format. The general goal of CPR is to encode coordinate decimals using less bits.

The java-adsb library provides a decoder to extract position information from the messages. However, the messages need to be fed to the decoder in a sequential manner, which makes parallelization with Spark less straight-forward. The messages first need to be sorted by their time of arrival to the central server. Once the messages are ordered, they are grouped by the ICAO24 identifier. These groups are then distributed to the worker nodes by Spark and the java-adsb library then decodes the raw messages and returns the positions.

As mentioned before, from the resulting dataset I have extracted the positions of the aircraft with ICAO24 3c674f and used this sample for working in the local environment. The positions of this aircraft can be seen on Figure 1.

4.5 Reducing data noise

At a quick glance at Figure 1 there is a big number of noticeably out-of-place dots on the plot. After a closer inspection these turn out to be noise in the dataset and naturally this noise needs to be filtered out.

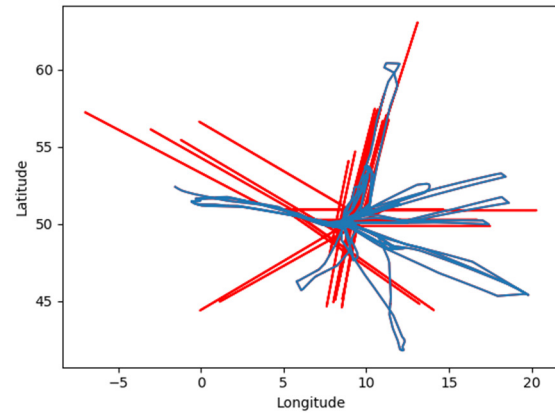


Figure 3. Data noise of the flight 3c674f

The main idea of the data noise reduction algorithm is to check whether the speed needed to move from one position to the next is too high for a plane to realistically achieve. Calculating the speed is rather straight-forward since the distance and the time is known. However, calculating the distance between two spherical coordinates is not as simple. It is better to first convert these coordinates to cartesian coordinates. Although the planet earth is not a perfect sphere, due to its scale and the small area of positions the differences and inaccuracies are negligible.

In addition to checking the speed, it is also a good idea to simply remove positions that are just too far away from sensors or that are outside of the examined area which is roughly Europe.

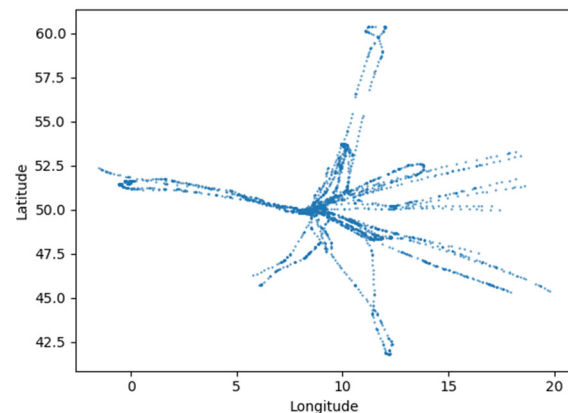


Figure 4. Positions points of the flight 3c674f after applying the Ramer-Douglas-Peucker algorithm

An example result of running this algorithm on the sample flight (3c674f) is shown on Figure 3. It is a plot of the path by connecting the positions together. The red path includes

¹⁰ <http://adsb-decode-guide.readthedocs.io/en/latest/content/cpr.html>

the data noise, while the blue path excludes it. The simple algorithm clearly removes these red spikes, as a result removing the data noise and creating a realistic path for the flight.

4.6 Removing redundant information

Since there are practically no obstacles in the sky, once a plane reaches its flight altitude, it will fly in a straight line for the most part. While the plane is in the air it will keep transmitting its location frequently. This creates a sequence of points that can be approximated with a single line without barely losing any information.

The Ramer-Douglas-Peucker algorithm is the most suitable algorithm for this case. The purpose of this algorithm is to simplify a curve defined by a set of points by finding a similar curve using the subset of the points. The exact algorithm will not be discussed in this paper. In short, the algorithm finds and removes points from the set which have a small effect on the curvature of the path.

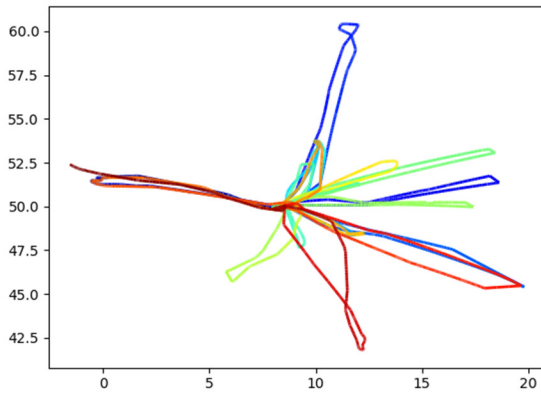


Figure 5. Path of the flight 3c674f after applying the Ramer-Douglas-Peucker algorithm

The algorithm has a single epsilon parameter which controls the granularity of it. The bigger the epsilon the more points will be removed and the less accurate the new curve will be.

A plot of one of the results is shown on both Figure 4 and Figure 5. It is clear from the former figure that the position points became a lot more spaced out, especially where the points formed a straight line. Comparing the path in Figure 5 and the blue path in Figure 3 one would not be able to easily tell the differences. Due to the difference between the scale of Earth and the scale of the plot, the small inaccuracies are invisible to the eye.

4.7 Clustering positions

Since the final aim of this project is to create an animated visualization of flights it is obvious that the animation will involve some kind of interpolation of the position points. Therefore, the positions of a single aircraft have to be clustered

into groups where each group represents a single flight. The positions in a single group will be interpolated but two positions from different groups will not be interpolated.

Often there are large gaps in time between two sequential positions. This gap is either due to the plane flying over an uncovered area or because it is a completely different flight. An algorithm is needed that is able to tell the difference between the two and cluster positions into distinct flight.

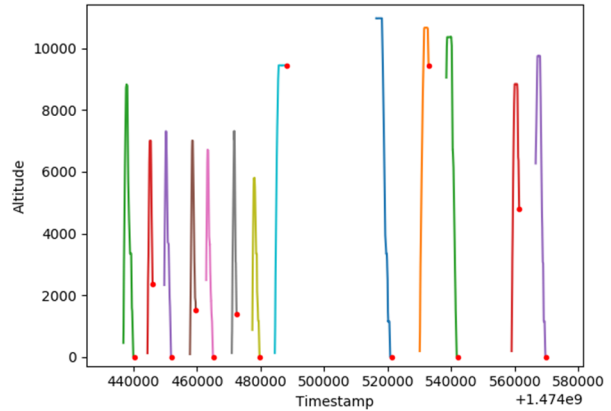


Figure 6. Clustered positions of the aircraft 3c674f

There are multiple conditions that can be checked to determine where the flight ends or cut in half.

- One obvious indicator are the surface position messages. Aircraft only broadcast this message when they are not airborne. Although this message is a clear boundary between two flights, aircraft do not always broadcast it when they land or before they take off.
- The time difference between two positions is a good indicator whether the flight should be cut or whether is a need to further investigate. If the time difference is small enough it is safe to assume that the two positions belong to the same flight. If the gap is too big it needs to be further investigated.
- If the time gap between two positions the application tries to tell how the path would “look like” if the positions points were to be connected with lines. In particular it looks at the angles formed by the lines. This algorithm is especially good at differentiating two flights that fly in different directions. It clusters the two positions to same flight if the plane stays roughly stays on the same path.

Initial experiments yielded results shown in Figure 6 and Figure 7. Both plots contain red dots where the algorithm detected the end of a flight.

4.8 Identifying flights

In order to enable flight filtering based on airlines, the flights need to be identified and assigned to an airline. The original dataset contains the so-called call sign messages which are basically identification messages. These messages contain the flight identifiers which are the flight codes one would see on a flight schedule.

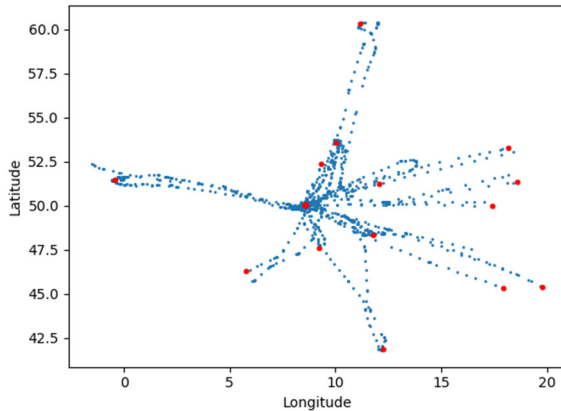


Figure 7. The positions where the splitter algorithm detected the end of a flight

To reduce the size of the resulting data the extractor does not simply decode all the messages. The same transmitter can be used for multiple flights, as a result it will transmit different flight codes at different times. Instead the extractor marks timestamps when a transmission of a flight code starts and when the flight code changes. This way the flight identification data for the whole dataset was only 29 Megabytes big. After extracting the flight codes from the dataset, matching them to the flights themselves is only a matter of overlapping the time intervals of the flight codes with the interval of the flights.

Generally, the first three characters of the flight codes are the ICAO codes that identify the airlines. There are exceptions in the cases of military aircraft and similar outliers, but their number is not big enough make a big difference. There are numerous sources on the world wide web which all if not most of the airlines and their respective ICAO. I have decided to use the tool developed by ICAO¹¹ themselves as it is a very convenient REST interface¹² to query the airline name associated with the ICAO code.

4.9 Restructuring the data

When all the necessary data has been acquired it is important to present it to the visualization client in a format that can be quickly process while keeping the footprint to a minimum. The animation visualization will be a web based

application, so it makes sense to use the JSON format as it basically represents a JavaScript object.

The main idea of restructuring the dataset is to use a Structure-of-Arrays¹³ form instead of an Array-of-Structures form. This restructuring can be thought of a transposition of the data. Instead of having an array of objects where each object contains the longitude, latitude and altitude coordinates, the data is structured into an object that contains three arrays. The arrays contain all the longitudes, latitudes and altitudes and a coordinate is identified by the index of the elements. This removes the redundant property names from the JSON text effectively greatly reducing the size of the files.

Another important aspect to consider is the format the text representation of the numbers. All the numbers (timestamps, longitudes, latitudes, altitudes) are double numbers. Timestamps are rather large numbers, which Java represents using scientific notation. In case of timestamps the scientific notation actually uses more characters than normally writing down the number. Furthermore, the positions broadcasted by the aircraft are very precise and contain information up to sixteen decimal places. Since the animation is rather large scale, that kind of accuracy is unnecessary. As a result, rounding all the numbers to the fifth decimal place and removing potential trailing zeroes result in sufficient accuracy and a much smaller footprint.

4.10 Animation techniques

The data is now ready to be used for an animation visualization. There are two visualization techniques that I have considered for this project the advantages and disadvantages will be discussed in the section.

4.10.1 Generating an animation

As we saw in section 2 there are multiple videos about flight animation have already been created. The animations created by NATS are created using 3D software and ItoWorld created their animation using their own software. However, since there is a cluster available, it can also be utilized to generate a video in a distributed manner.

In theory the worker nodes can generate frames for the video by distributing the data by time. And when the workers are done, the master node can collect the data and stitch together the frames to create a full video out of it. Unfortunately, this would need a lot more time as it requires a lot of graphical programming skill. In the end I have decided to create a web based application instead.

4.10.2 Interactive application

When creating such an application the main problem is how will the data be fed into the visualization. The data set

¹¹ International Civil Aviation Organization

¹² <https://www.icao.int/safety/iStars/Pages/API-Data-Service.aspx>

¹³ <https://www.youtube.com/watch?v=qBxeHkvJoQQ>

is still not small enough to just download the full dataset to the client computer and have it work on that. The application needs to fetch parts of the dataset that it is currently animating or that it will animate as time progresses.

The aim of this application is to be as interactive as possible. It allows the adjustments to the speed of time. It cannot allow to set the speed to high because the application will not be able to keep up by constantly downloading the data. It also allows filtering the flights by the airlines. These aspects were all taken into consideration when the data structure and the communication pattern were designed.



Figure 8. A frame of the animation

The flights are partitioned into 4-hour interval chunks. This grouping is not as simple as one would think because there are flights that can overlap with multiple 4-hour intervals. Another important thing to consider is that the application allows quick jumping in time, so it has to know about the flights that started in the past but has not finished before the time the user jumped to. The simple solution is to cut the long flights up into smaller flights and put these into different partitions. In the dataset these will appear as different flights, but in the animation it will look like one single flight. This also solves the problem of jumping in time because whenever the user quick jumps to another time, the application will quickly load the corresponding partition that contains all the flights that are currently airborne at that time.

To make the animation even smoother, the application buffers the next partition in the background anytime one chunk of flights start animation.

The animation needs a map as a background for us to be able to tell where the planes actually are. Natural Earth¹⁴ is a

public domain map dataset containing tightly integrated vector and raster data. Using this dataset and a combination of tools¹⁵ for conversion the D3.js library can draw the detailed outlines of the European countries in SVG format using the Mercator projection. The D3.js library is also used to project the GPS coordinates on to the map inside the web application.

There can be a lot of planes on the map at the same time. In order to make the animation as smooth as possible the planes are represented with a simple dot. The larger the radius of the dot is, the higher the plane is in altitude. Figure 8 shows an example frame of what the animation looks like.

5. EXPERIMENTS

Some of the Spark jobs in the pipe have one or more parameters and depending on this result they can yield different results. Therefore, there is a need to conduct small experiments with different input parameters to get the best possible outcome of the pipe elements.

5.1 Position extraction

The message decoding and position extraction has no input parameters, so this part of the pipeline does not need much experimentation. However, it is worth considering the scale of the size of the dataset in question.

OpenSky's network of ADS-B receivers do not cover all of Europe, only part of it. Yet in just a span of a week they have recorded over 8 billion messages which has a size of 590 Gigabytes in a compressed format. After extracting all the positions messages, that were about seven percent of the whole dataset, and decoding the position coordinates the resulting dataset consisted of more than 1.08 billion GPS coordinates and occupied roughly 75.3 Gigabytes of disc space.

If every single aircraft were to be equipped with an ADS-B transmitter and the receivers were to have a coverage of 100% percent around the globe the amount of data produced by these aircraft would be unimaginably high.

5.2 Noise reduction

The noise in the dataset manifests in the final animated visualization quite spectacularly. In the initial versions of the animation there were flights that seemingly fly across the map with super speed. Naturally this meant that the noise reduction algorithm was too permissive.

The first version of the filter had a relatively high-speed threshold and did not remove positions that were out of frame. This threw out about 33 million records from the dataset which is only a very small fraction of the whole dataset. The results of these were not good enough. After greatly reducing the speed threshold and removing the out-of-frame positions the filtering yielded way more realistic results. But there were planes which were still seemingly

¹⁴ <http://www.naturalearthdata.com/>

¹⁵ ogr2ogr, geo2topo

jumping back and forth in a single path. This particular phenomenon happens very frequently over the Sardegna and Corse islands. The cause of this is unknown but I would be quite interested in finding out why are the positions so inaccurate in this area.

5.3 Douglas-Peucker algorithm

This polygonal approximation algorithm has a single input parameter called epsilon. Basically, the algorithm checks if the distance of a point and a particular line segment is smaller than epsilon or not. It will get rid of all the points which are closer to the line than epsilon since these points can be approximated with the line segment. The larger the epsilon, the more points will be removed from the curve.

Few experiments were conducted with different epsilon values and the outcome can be seen in Table 2. It is clear that the Ramer-Douglas-Peucker is a very effective algorithm when applied to plane trajectories. It can possibly remove more than 99% of the dataset while keeping the plane trajectories almost unchanged. This is the most important element in the pipeline because without such a data reduction and approximation the interactive animated visualization would not be possible. If one would want to achieve an even lower data size, the epsilon could be set to a higher value and the path would still stay relatively unchanged.

Table 2. Ramer-Douglas-Peucker experiment results

Epsilon	Reduced data size
100	~1.35%
200	~1.03%
300	~0.75%

In conclusion, I have decided that an epsilon of 300 produced a dataset that is sufficiently small enough to continue working on the dataset in the local environment. It is also small enough to prevent the web application from being too slow by constantly loading data. As we have previously seen in Figure 5. Path of the flight 3c674f after applying the Ramer-Douglas-Peucker algorithm Figure 5 the differences between the original and the reduced path is practically invisible to the human eye.

5.4 Flight detection

The algorithm of flight detection is quite straight-forward and as a result it did not require much fine tuning at all. The algorithm identified a total of 126251 flights.

The flights also needed to be associated with an airline in order to provide airline filtering. Using the ICAO application programming interface, the algorithm was able to identify 76% of the flights leaving 30256 flights unidentified. The main reason for a flight to be unidentified is that it is a private aircraft as the algorithm was able to identify even military

aircraft. A small fraction of the results of the flight identification is shown in Table 3.

Table 3. Airlines and the number of flights the belongs to them

Airline	Number of flights
Ryanair	382
Deutsche Lufthansa, A.G.	314
EasyJet Airlines Co. Ltd	248
Turk Hava Yollari (Turkish Airlines Co.)	221
British Airways	218
Air France	178
Scandinavian Airlines System	150
KLM Royal Dutch Airlines	130
Alitalia - Compagnia Aerea Italiana S.P.A.	129

5.5 The animation

Since it is a web based interactive animated visualization and only parts of the dataset are loaded into memory at a time, network communication speed is a key aspect to consider. As discussed in section 4.9 the dataset is partitioned into 4-hour interval chunks. The distribution of the flights among these partition is shown in Figure 9 and the size of the files is shown on Figure 10.

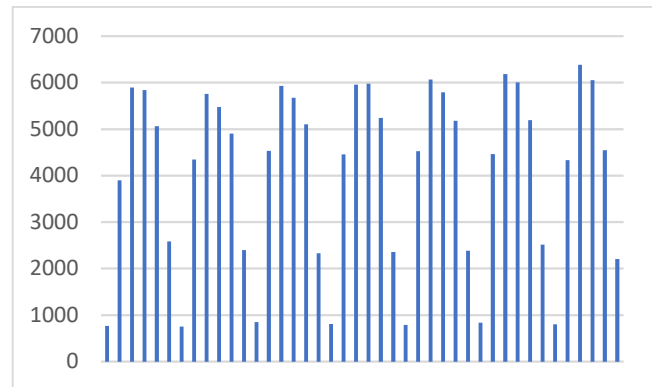


Figure 9. Distribution of flights between partitions. The bars represent the chunks in a chronological order and the Y axis is the number of flights

It is clear that this distribution is quite imbalance and some chunks will load very quickly while other chunks will load very slowly. If the flights were to be distributed equally the it would be the time interval of the partitions that is unbalanced, and the application would not have enough time to load a chunk. Experimentation were made on the local computer to see how long it takes to load chunks. An a relatively fast internet connection the bigger chunks could take up to 8-10 seconds to be downloaded and buffered.

Measurements were made both when there were no flight filters and there were a lot of planes on the screen and when flight filtering was applied. The results show that the load time is not affected by how many planes there are on the screen. During these measurements caching was disabled on the client and gzip compression was enabled on the host server that is located in Frankfurt. It is concluded that unless the internet connection is very slow, there should always be enough time for the application to buffer the flights.

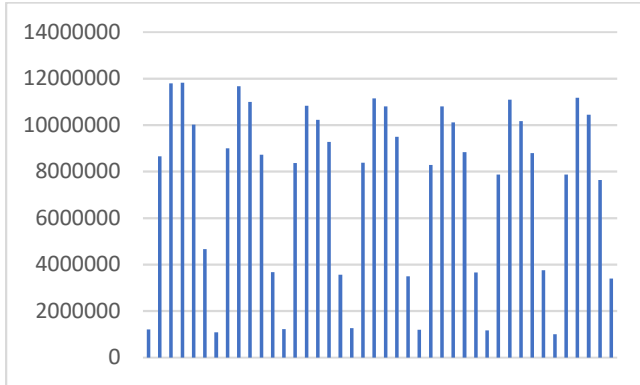


Figure 10. JSON file sizes of the partitions. The bars represent the chunks in a chronological order and the Y axis is the size of the file in bytes

6. CONCLUSIONS

The final product of this project is a fully featured web based animated visualization of flight position ADS-B messages. The application provides an interface to control of the speed of time, to jump between times and to filter the flights by airlines and flight codes. With the previously detailed setup and the efforts put in into the project that main goal was successfully reached.

All the questions presented in section 3 have been answered during the process of research and development of this project. I was able to process and extract flight information from an enormous dataset of ADS-B messages using existing Java tools and Big Data technologies on a large cluster computer. The size of the dataset was reduced to less than 0.01% of the size of the original dataset while keeping all the necessary information available or reproduceable. The flights were identified with the help of an external API using a separation algorithm. And finally, an interactive animated visualization was created that can run on most of personal computers because it does not use a lot of resources.

7. FUTURE WORK

Watching the final product running in action there are still obvious imperfections in some of the algorithms. The paths of a few flights are not smooth, and they seem to jitter or jump from one place to another midair. There are still a few

flights that seem to travel way faster than the fastest commercial airplane because there is still very little noise in the dataset. Due to the inaccuracies of the flight detection algorithm, some aircraft are shown to stand on the surface for quite a long time because the flights before and after were not separated.

Therefore, the obvious next step is to improve on these algorithms and make them cleverer and use the existing dataset in a better way. For example, the flight detection algorithm could use an external dataset of airports to determine whether an aircraft has landed or not. At the time of writing Natural Earth has a large database of airports that can be easily filtered to extract the larger ones.

There are also many options to increase the interactivity of the visualization. For instance, the airport data set can be used to filter the flights by the departure and arrival airports. The airports or the cities they belong to could be integrated into the map of the application. Right now, it is very hard to find out any kind of information about the flights on the map. Using external data sources, the application could provide additional information such as the schedule of the flight, the code of the flight, the path it is taking in the air or whether it was delayed or not.

Since more and more aircraft are adapting to using ADS-B transmitters more and more data will be produced by planes. Additionally, the coverage of receivers will only grow with time meaning that faster and better will be needed to process the large amount of data collected by these receivers. In the future there will be a need to also adopt existing technologies to the growing amount of data. This project will have to be improved for it to be able to handle such a large dataset.

Finally, this project could be used in future researches to identify flight patterns, detect aircraft irregularities or military aircraft. Similarly to the works discussed in section 2.1 image based flight queries and other techniques could be combined with this project to gain a deeper insight into the dataset.

8. REFERENCES

- [1] C. Hurter, S. Conversy, D. Gianazza and A. C. Talea, "Interactive image-based information visualization for aircraft trajectory analysis," *Transportation Research Part C: Emerging Technologies*, pp. 207-227, October 2014.
- [2] J. Vaughan, D. Whyatt and G. Brookes, "A parallel implementation of the douglas-peucker line simplification algorithm," *Software: Practice and Experience*, pp. 331-336, March 1991.