

P6 - Cause for Emergency.

Gabriele Di Bernardo
Vrije Universiteit
Department of Computer Science
g.dibernardo@student.vu.nl

Sleiman Sleiman
Vrije Universiteit
Department of Computer Science
sleiman@outlook.com

ABSTRACT

In the last decade, we assisted to a massive adoption of the Automatic dependent surveillance – broadcast (ADS-B) [14] on board of modern aircraft. ADS-B is a technology that allows an aircraft to determine its current position and broadcast it over the ether. ADS-B messages can be received from the ground station and by other aircraft: in this way airplanes in flights can use the ADS-B signals in order to coordinate each other.

The ADS-B has been created as secondary surveillance radar system in order to supplement the primary surveillance radar that suffers from some weakness like limitations at low altitudes and in bad atmospheric and weather conditions (land-based systems) or they are limited to large commercial or military aircraft (airborne system) [23].

The OpenSky Network is a community-based on off-the-shelf sensors which continuously collects air traffic surveillance data. [19]. The OpenSky Network project started in 2012 as a research project between Arma Suisse (Switzerland), University of Kaiserslautern (Germany), and University of Oxford (UK) with the objective of improving security, reliability, and efficiency of the airspace utilization by providing *open-access* to public of the air traffic control data collected through the sensors. At the moment the OpenSky Network involves thousands of sensors connected to the internet by volunteers, industrial partners and academic organizations. OpenSky Network’s sensors continuously collect (mostly) ADS-B messages; the obtained data is kept by OpenSky, originating, in this way, the largest air traffic surveillance dataset of its kind in the world. This dataset is freely accessible by researchers and end-users that can interact with the collected data for academic and scientific reasons [18].

In this document, we describe a systematic and scalable approach in order to identify aircrafts that have been subject to emergency situations using a portion of the OpenSky Network dataset. In particular, our goal is to detect flights that in response to an emergency situation made an emer-

gency landing analyzing a portion of the ADS-B messages collected by the OpenSky Network. In order to have an insight and a better understanding of the reasons that caused the emergency landing, we exploit the historical archive of the public Twitter data.

1. INTRODUCTION

After World War II, commercial and civil aviation grew rapidly, and always more advanced aircraft have been used to transport millions of passengers around the world. From the mid-’50s the attention put on aircraft safety and air traffic control drastically increased [11]. This was possible also because of new technologies that were designed at first for military purposes. The technology adopted from the military context and used for the air traffic control consists namely in the radar. What today is called *primary radar* provides continuous surveillance of air traffic and they can detect and report the position of anything that reflects its transmitted radio signals including, depending on its design, aircraft, birds, weather and land features.

The necessity to identify each aircraft more easily and reliably than the primary radar led to the adoption in civil aviation of another technology developed initially for military purposes. This technology is now known as *secondary surveillance radar* and it basically consists of a device equipment mounted on board of the aircraft. This device, known as *transponder*, is a radio transmitter/receiver that is used for replying to signals from an interrogator; the interrogator can be a ground station co-located with a primary radar or another aircraft.

US Federal Aviation Administration’s Next Generation (NextGen) upgrade proposes some radical improvements to increase the capacity and safety of the aviation. In particular, NextGen proposes to migrate the air traffic control system from a radar-based system with radio communication to a satellite-based one. A key component of this upgrade is the Automatic Dependent Surveillance-Broadcast (ADS-B) system. Aircraft equipped with ADS-B transponder broadcast ADS-B messages constantly.

ADS-B messages transmit different types of aircraft-related information; some of the information that are broadcast from the aircraft are:

- *Aircraft position (airborne and surface). This kind of message contains information regarding the aircraft current position: namely latitude, longitude and altitude.*

- *Aircraft identification information like ICAO code or airline callsign.*
- *Velocity.*
- *Emergency/priority status.*

The ADS-B equipment is widely adopted in modern aircraft and in airplanes of the commercial's fleets; the ADS-B will be mandatory in the US by 2020 and in Europe from 2017.

At the moment there is also some debate regarding the adoption of the ADS-B technology since it is not considered secure namely because the ADS-B transponder emits messages in an unencrypted fashion [13, 17].

The OpenSky Network [7] provides the biggest dataset of ADS-B messages collected by sensors maintained by volunteers. Most of the sensors are located in Europe capturing more than 30% of the commercial air traffic.

In this paper, we present our work that consisted of analyzing a portion of the OpenSky Network with the objective to find flights that for emergency reasons made an emergency landing. The data available to us consists of the messages of the entire September 2015 (about 200 GB), and a week of data from September 2016 (about 600GB). In order to analyze this large amount of data we are going to process the dataset on a large scale infrastructure; in particular, we used the SURFSara Hadoop cluster.

2. RELATED WORK

In this section we present an overview of the papers that has been consulted while approaching our project work.

2.1 Landings & flights schedule anomalies detection

We are interested in detecting flights that had an emergencies and that eventually made an emergency landings; a good starting point is filter the dataset looking for flights that emitted *emergency messages*; however this approach could not be optimal and we cannot rely exclusively on it because of the sparsity of the ADS-B messages stored in dataset. For this reason, we focused on detecting flights that had an intermediate stop or more. In general, we looked for flights that present some anomalies when compared to the regular flight's schedule. In order to accomplish this, we used namely the ADS-B position messages.

In the literature, there are many papers and research projects that describe techniques for detecting anomalies in moving objects [21, 22]. However these approaches cannot suite well in our context; in particular airplanes can take different trajectories according to the weather condition, size and type of the aircraft, airlines.

In a very recent paper [15] Junzi Sun, Joost Ellerbroek, Jacco Hoekstra propose some machine learning techniques in order to categorize aircraft on a large-scale. They were able to identify the different flights phases of an aircraft through clustering algorithms and fuzzy sets theory. In particular they took as input some features of an aircraft in flight that can be easily inferred from the dataset (like ICAO, time, latitude, longitude, altitude, heading and speed) and from these features they determined the different phases of a flight (cruise phase, climbing phase or landing phase). However, as stated in the paper, their approach is not able to separate flight data into further detailed flight phases, such as taxiing,

take-off, landing, and initial climbing/descending; in fact detecting a landing or a take-off from an ADS-B messages dataset can be a task extremely tedious and difficult.

We are also interested in detecting landings and we needed to find a valid approximation that can help us to detect possible anomalies and emergency landings. In [12] they described an approximation based on a professional pilot's experience that can be used in order to detect a landing phase of an aircraft.

Starting from the methodology described in [15] and the approximation illustrated in [12] we designed our own approximation algorithm to detect landings and possible emergency and unscheduled landings.

3. RESEARCH QUESTIONS

In this project we want to answer the following main questions:

- How to detect flights that had an emergency during a certain period by analyzing a set of ADS-B messages?
- Is it possible to detect emergencies' cause by querying Twitter's historical database?
- How we can enhance our basic flight knowledge acquired from the ADS-B in order to get a clearer representation of our data and be able to query Twitter?

In the rest of the paper we describe how we approached these question, our efforts and how we tried to answer to these research questions.

4. PROJECT SETUP

In this section, we describe the setup and the technologies used both locally and on the SURFSara infrastructure in order to analyze the data set throughout multiple phases.

4.1 Data Understanding

As a first attempt to understand the ADS-B dataset we started exploring the tools and the data provided by the OpenSky Network sample package on Github [5]. The package consists of a sample dataset in addition to Java tools to parse, split and sort ADS-B AVRO files. In addition to the ADS-B guide provided by Junzi Sun [1] this package helped us a lot in understanding the structure and the type of the ADS-B messages especially when it comes to emergency messages.

4.2 Data Exploration

Because of the size of the input dataset we need to process it on a large-scale infrastructure. In particular for this project we are required to analyze this large amount of data on the SURFSara [10] Hadoop cluster.

In order to explore the data, we needed to be able to iterate fast and constantly having result feedback from our program. Therefore we decided to work on a small part of the dataset and to mimic the SURFSara's environment locally by installing Spark and Hadoop HDFS locally in addition to PySpark and Jupyter Notebook [2] for fast feedback. This setup not only helped us to explore the data but also to ship to production quickly.

4.3 Data Processing Phase

The actual data processing phase on the entire dataset has been performed, as already mentioned, in the SURFSara infrastructure. The Hadoop cluster at SURFSara consists of 170 data/compute nodes; these nodes have 1370 CPU-cores for parallel processing. The cluster offers the Hadoop 2.0 core-functionalities:

- HDFS, a Java-based file system that provides scalable and reliable data storage, and it was designed to span large clusters of commodity servers.
- YARN (Yet-Another-Resource-Negotiator), provides API to develop any generic distributed application, enabling Hadoop to support more varied processing approaches and a broader array of applications. Yarn it also responsible for resources scheduling and handling resource requests. Because of HDFS and YARN we can think to Hadoop like the distributed operating system for big data applications.

For our project on top of Hadoop we decided to use Apache Spark; Spark is an open source cluster computing framework that offers over 80 high-level operators that make it easy to build parallel applications. Spark powers a stack of libraries including SQL and DataFrames, MLlib for machine learning, GraphX, and Spark Streaming.

We decided to use Spark first due to its large ecosystem as newcomers we wanted a framework with enough documentation and tools. Spark offers a wide array of tools out of the box or with little configuration like Spark SQL, MLlib and Spark Streaming. Having this arsenal of tools easily accessible allows us to shift focus from the infrastructure to the data. Moreover, the speed and convenience brought by RDD give Spark an underhand over Hadoop Map Reduce [16, 20, 24].

4.4 ETL

The Extract, Transform, Load (ETL) phase is the first phase of our large-scale processing pipeline. In particular we designed a Spark job that given as input an ADS-B messages dataset it reorganizes data in a convenient way that can make it easier to process or visualize it. We want to organize and correlate the ADS-B messages according to the specific civil flights for a certain date. In order to accomplish this several steps are involved.

Since we are mostly interested in detecting emergencies related to the civil aviation the first necessary step is to understand which aircraft actually belong to an airlines company. In order to decode each ADS-B messages we used (and we slightly changed some Java classes in order to use it on top of Spark) the java-adsb library provide by the OpenSky Network [4]. Each ADS-B message contains the ICAO24 code, a code that uniquely identify a specific aircraft. There is a specific type of ADS-B messages named *identity messages* that actually provide more accurate information regarding the aircraft that emitted the message. It is possible to obtain the flight *callsign* from an ADS-B identification data frame. From the identification messages we can filter the entire dataset with messages that contains ICAO24 that belong to airlines' callsigns. A list of airlines' callsigns can be easily found online; for example we used the one provided by [3].

For each day of the dataset, we can now understand which aircraft made a certain flight for a specific airline. We can infer the position messages of each flight for a specific day looking for ADS-B messages that contain the aircraft's ICAO24 that were emitted after the first identification message for a certain callsign and before the last occurrence of an identification message with the same callsign. In this way we can have a good approximation of the ADS-B messages emitted by an aircraft when it made a specific flight for a certain airline.

Once we identified each flight we can immediately check if one of these flights emitted emergency messages; in the case that some occurrences of emergency messages have been found for some flights we produce as output a list of flights (that emitted at least one emergency message) with their position messages; if no position messages are available the positions are approximated by the ADS-B sensor's coordinate (available to us for each ADS-B message). This output (in JSON format) contains flights that emitted emergency messages and each of the callsign (together with the date) of these flight will be utilized to query Twitter in order to discover the cause of the emergencies.

The rest of the flights from our dataset that did not emit any emergency messages will be reorganized in a convenient way; in particular for each flight we keep its ADS-B position messages (both airborne and surface). These flights will be put as output in a compressed format (AVRO compression) and they will be ready for further processing.

4.5 Anomalies detection

One of the output of the ETL phase is a list of flights that did not emit any emergency messages. For each of this flight, we have the ICAO24 of the aircraft, the callsign, the date and a list of locations indicating the path that took each flight.

For each of this flight, we want to understand if it made an emergency stop or in more general if its path diverged from the regular schedule; for this task we designed the second stage of our large-scale pipeline.

4.5.1 Detecting landings

In order to detect landings we implemented an algorithm starting from the algorithm described in [12]; in particular they stated that according to a professional airline pilot an aircraft can be considered in landing phase if it descends 1500 feet. We did some investigations on a smaller portion of the dataset available to us and we discovered that actually this could be considered a good approximation. In particular we identified a recurrent pattern: each aircraft keeps an average cruise altitude and when it descends 1500 feet (from its average altitude) it is going to land.

We implemented an algorithm that tries to identify aircraft landings. In particular given the list of position messages that belong to a flight the algorithm try to approximate how many stops (and on which airport) the aircraft made. This algorithm relies on the average altitude of each aircraft when it made a certain flight: in particular before executing the algorithm we calculate the average cruise altitude (we consider a certain altitude a cruise altitude if it is at least of 5000 meters) for each aircraft and for each callsign that the aircraft used. In this way we want to infer the average cruise altitude for an aircraft when it uses a certain callsign. In order to detect a possible landing beside

airborne position messages we use the surface position messages and we also take into account if a certain aircraft has a "silent" period where no location messages are being sent after the possible landing. After detecting that an aircraft is in landing phase we simply take the position message with the minimum altitude and we approximate as the landing airport the nearest airport from the last location message received. This is an approximation that is highly dependent by the last location message that we have for a certain flight; in fact, it depends on the antenna location, the obstacles between aircraft and the antenna, and the distance between the ADS-B receiver and the destination airport. The algorithm has been designed to detect one or more landings for the same flight; in this way could be possible to detect a flight that had multiple stops.

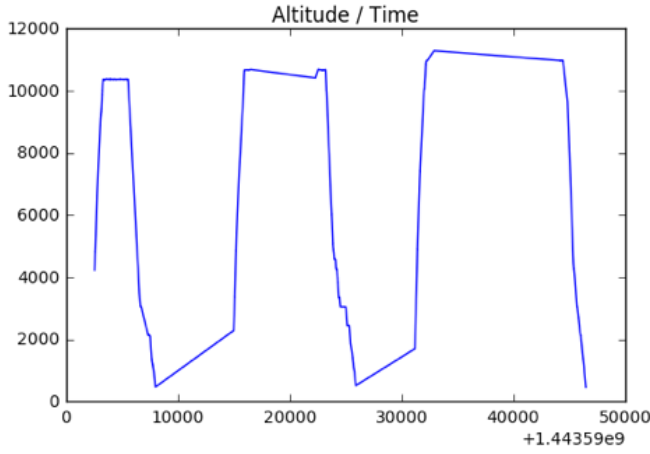


Figure 1: Altitude for aircraft 406d8e during an entire day from September 2015.

The algorithm considers as departure airport the nearest airport from the first location message received for a certain flight. The nearest airport is calculated as the distance between two coordinates with the *Haversine formula*.

Test the correctness of this algorithm is not an easy task; however some tests have been performed. In particular using a list of civil airports in Europe (without small and secondary airports) and using the dataset from September 2016 we checked the outcome of the algorithms using flightradar24.com [9]; in particular for each day of the dataset we ran the algorithm and we checked if the departure and destination airports produced by the algorithm for each flight are the same of those reported by flightradar24. The algorithm detected a correct departure airport and destination airport for up to 76% of daily flights (considering flights that the algorithm reported having only one final stop). We believe that this algorithm can be a good approximation when there is abundant information for each flight. Its simplicity allows the algorithm to be executed in a fast way when it is required a good approximation in the minimum amount of time (for example in streaming application). However since it is a naive implementation it can be error-prone in cases where the messages in the input dataset are drastically affected by noise or where the data is really sparse and not continuous.

The described algorithm has been used in the anomalies detection phase of the large-scale pipeline of this project. The flights detected by the algorithm with stops number different than two are put in output in JSON format and their callsign will be used to query Twitter in order to understand if there is an actual anomaly for each flight, and in that case understand the causes of the emergency.

4.5.2 Detecting anomalies from flights schedule

After running the landing detection algorithm and put as output the flights for which the algorithm has detected more than two stops we can try to cluster the remaining flights (the flights the algorithm detected exactly one departure airport and one landing airport) on daily basis. In order to accomplish this the different flights are clustered for each day of the week. In particular since we have determined the departure and destination airport we can learn a good approximation of the actual flights schedule. We clustered each flight per day of the week and in this way anomalies in schedule can be easily detected. The flights that present anomalies in the schedule then are reorganized in JSON format and put in output by the Spark application: the authenticity of the anomalies and the possible causes will be inferred by a Twitter query on the tweets historical dataset.

This approach can be work properly only if the landing detector algorithm produced an approximately good result.

4.6 Data Enhancement Visualization Phase

At this stage the data is small enough to run it locally thus the technology used in this phase doesn't really matters however we decided to use GoLang to crawl the flight data and reduce the noise and positions. As for the user interface we used Vuejs as a framework in addition to a small set of Javascript tools.

5. RESULT VISUALIZATION & ENHANCEMENT

After running the large-scale pipeline on the input dataset we ended up with some JSON files containing an array of flights that emitted an emergency message or flights that the implemented algorithm detected some anomalies during the covered period. The first attempt to plot this dataset was to draw a polyline representing the flight path on Google Maps as illustrated in figure 1.



Figure 2: First flight path plot.

The first plot implies that either the results are wrong or the data still needs some filtering. In this section we describe the process we use in order to improve the data visualization of the results achieved by the data processing phase.

5.1 Data Enrichment

In order to improve the details related to each flight (like the departure, destination, duration and airline) we created

 Track Flight **DLH441** / **DLH441**


George Bush Intercontinental (IAH)		Frankfurt Main (FRA)
Aircraft A388	Airline Lufthansa	Journey time 09h45m
Flight no. DLH441	Callsign DLH441	Arrival day Next day
Service type Normal passenger	Seats 526	Freight class Containers
Freight capacity 15.3 tons	Passenger classes First Class, Economy, Business Class, Shuttle	

Following the crawling process we decided to build a query to search Twitter’s Historical API [6]. The query is structured as follows:

```
{
  "publisher": "twitter",
  "streamType": "track_v2",
  "dataFormat": "original",
  "fromDate": "201609010000",
  "toDate": "201601300000",
  "title": "flight-emg-2016",
  "serviceName": "Flight",
  "rules": [
    {"tag": "squawk", "value": "squawk (7700 or 7600)"},
    {"tag": "airline", "value": "{airline1} {airline2}.."},
    {"tag": "flight-DLH441", "value": "DLH441"},
    {"tag": "flight-{callsign}", "value": "{callsign}"},
    {"tag": "flight-{callsign}", "value": "{callsign}"},
  ]
}
```


```
"matching_rules": [{
    "tag": "flight-DLH441",
    "id": 5042011855517844061
}]}
```

After knowing the departure and destination we noticed that some flights actually contain multiple trips; in particular after arriving at destination some flights come back to the original destination airport keeping the same callsign. For this reason we needed to isolate the trip that contains the emergency messages in order to have a proper representation of the flight.




@a380fanclub / A380 fanclub

RT @greg787: There's a supposed hole in the aircraft. That's what the "ground mishap" was.
@lufthansa @Lufthansa_USA #DLH441

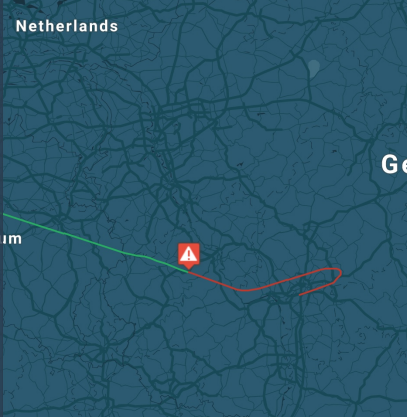


@greg787 / Greg D

Captain making his way down to check things out. @Lufthansa_USA #DLH441
<https://t.co/omEvTcVv4O>



Netherlands



Belgium

At the first stage we split the location into two parts (before and after) the first emergency timestamp (when it is present); then we sort the locations accordingly the post-emergency locations ascending by timestamp and the pre-emergency locations descending by timestamp.

5.3 Noise Filtering

	<input checked="" type="radio"/> top	<input type="radio"/> Preserve log
[HMR] Vue component hot reload shim applied.		
	Fri, 23 Sep 2016 06:12:36 GMT	50.22587585449219,7.192936947471217
3	Fri, 23 Sep 2016 06:12:37 GMT	50.225162829382946,7.19597068992821
2	Fri, 23 Sep 2016 06:12:37 GMT	50.22477722167969,7.197562769839638
2	Fri, 23 Sep 2016 06:12:39 GMT	50.22330073987023,-2.5258162214949325
	Fri, 23 Sep 2016 06:12:40 GMT	50.222991943359375,7.205296566611842
2	Fri, 23 Sep 2016 06:12:41 GMT	50.22209038168697,7.209258208043717
2	Fri, 23 Sep 2016 06:12:42 GMT	50.221710205078125,7.2108620991488485
	Fri, 23 Sep 2016 06:12:42 GMT	50.21238977050781,7.212307578638979
2	Fri, 23 Sep 2016 06:12:43 GMT	50.22074036679025,7.215048300253378
3	Fri, 23 Sep 2016 06:12:44 GMT	50.219995530985166,7.218314505912162
2	Fri, 23 Sep 2016 06:12:45 GMT	50.21971621755826,7.2195022170608105
	Fri, 23 Sep 2016 06:12:46 GMT	50.21905517578125,7.222282008120888
	Fri, 23 Sep 2016 06:12:46 GMT	50.21868896484375,7.2238721345600325
3	Fri, 23 Sep 2016 06:12:47 GMT	50.21831965042372,7.22551500470845

To filter the noise we measured the distance between each position using the Haversine formula then we filtered each location on the following basis

5

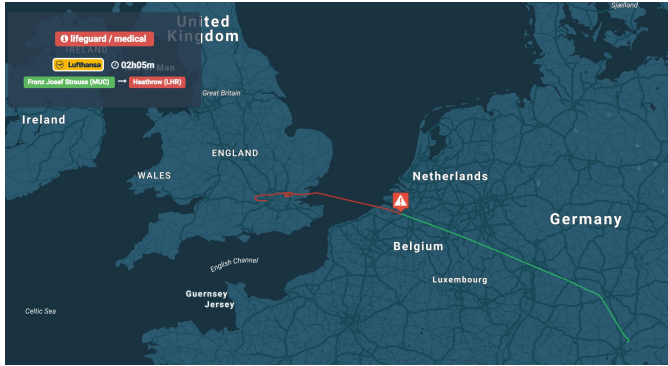


Figure 6: MUC, LHR trip plot after noise reduction

6. CONCLUSION

We tried to detect anomalies and their causes and for some flights (mostly from messages of the dataset from September 2016) we actually accomplished in this task. However we still have some issues that we would like to solve like detecting noisy emergency messages. One solution for detecting noisy emergency messages is by monitoring the behavior of the plane after the emergency message, another solution could be training a model from a set of emergency flights and regular flights.

Noise is also an issue for the detecting landings algorithm; in fact because of some noisy position messages a *false-positive intermediate stop* between departure and destination airport can be detected.

Our implementation was able to identify some anomalies and some emergency situations and we found evidence on those on Twitter; however our implementation and in particular our landing detector algorithm and anomalies in schedule detector algorithm made some assumptions that in pair with the sparsity of the input dataset can produce a series of false positive.

We truly believe that with a major adoption of the ADB-S sensors on board of aircraft and with a greater spread of ADS-B receiver throughout Europe this will lead to new scenarios in this kind of analysis that will have a strong impact on research world and in general in improving human lives.

7. ACKNOWLEDGMENTS

We would like to thanks Peter Boncz, and Hannes Mühleisen for preparing and teaching this course. We have to say it is an unusual and challenging course that delivers knowledge in a very realistic and motivating method.

8. REFERENCES

- [1] <http://adsb-decode-guide.readthedocs.io/en/latest>.
- [2] <http://jupyter.org>.
- [3] <http://openflights.org/>.
- [4] <https://github.com/openskynetwork/java-adsb>.
- [5] <https://github.com/openskynetwork/osky-sample>.
- [6] <https://gnip.com/historical/>.
- [7] <https://opensky-network.org>.
- [8] <https://planefinder.net/data>.
- [9] <https://www.flightradar24.com>.
- [10] <https://www.surf.nl>.
- [11] Natca - a history of air traffic control, <http://www.natca.org/images/natca-pdfs/publications/atichistory.pdf> visited october 2016.
- [12] Hugo Wallenburg Alexander Renz-Wieland. Airport quality: holding and go-arounds.
- [13] Robert Mills Donald McCallie, Jonathan Butts. Security analysis of the ads-b implementation in the next generation air transportation system. *International Journal of Critical Infrastructure Protection*, 4:Pages 78–87, August 2011.
- [14] P.R. Drouilhet, G.H. Knittel, and V.A. Orlando. Automatic dependent surveillance air navigation system, October 29 1996. US Patent 5,570,095.
- [15] Jacco Hoekstra Junzi Sun, J. Ellerbroek. Large-scale flight phase identification from ads-b data using machine learning methods. *7th International Conference on Research in Air Transportation*, June 2016.
- [16] Umar Farooq Minhas Limei Jiao Chen Wang Berthold Reinwald Fatma Ozcan Juwei Shi, Yunjie Qiu. Clash of the titans: Mapreduce vs. spark for large scale data analytics.
- [17] Sameer Alam Leon Purton, Hussein Abbass. Identification of ads-b system vulnerabilities and threats. *Defence and Security Applications Research Centre, University of New South Wales Australian Defence Force Campus, ACT 2600*, 2010.
- [18] Markus Fuchs Matthias Schäfer, Ivan Martinovic. Opensky: A swiss army knife for air traffic security research. *Digital Avionics Systems Conference (DASC), 2015 IEEE/AIAA 34th*, 2015.
- [19] Vincent Lenders Ivan Martinovic Matthias Wilhelm Matthias Schäfer, Martin Strohmeier. Bringing up opensky: a large-scale ads-b sensor network for research. *IPSN '14 Proceedings of the 13th international symposium on Information processing in sensor networks*, pages 83–94, 2014.
- [20] Rohan Arora Satish Gopalani. Comparing apache spark and map reduce with performance analysis using k-means. *International Journal of Computer Applications*.
- [21] J. Eriksson Y. Wang G. Forman X. Liu, J. Biagioni and Y. Zhu. Mining large-scale, sparse gps traces for map inference: Comparison of approaches. *In Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, page Pages 669–677, 2012.
- [22] Sangkyum Kim Hector Gonzalez Xiaolei Li, Jiawei Han. Roam: Rule- and motif-based anomaly detection in massive moving object data sets. *Proceedings of the 2007 SIAM International Conference on Data Mining*.
- [23] Jinglu Qiao Yan Zhang. Ads-b radar system. August 19 2008. US Patent US 7414567 B2.
- [24] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark:

cluster computing with working sets. *HotCloud*,
10:10–10, 2010.