23rd ITS World Congress, Melbourne, Australia, 10–14 October 2016

Paper number EU-TP0220

Open source big data landscape and possible ITS applications

Adam Warski[1*], **Tomasz Szymański**[2]

1. SoftwareMill, Poland. Address: ul. Na Uboczu 8/87, 02-791 Warszawa, Poland; tel.: +48 660 703 691, e-mail adam.warski@sofwaremill.com

2. SoftwareMill, Poland

Abstract

Big data is taking the world by storm, and the ITS industry is becoming one of the key players as the users, vehicles, and the road infrastructure are becoming rich data producers and consumers. In our paper, we describe the freely available open-source big data tools that can be used for data ingestion, analysis, and storage, such as Apache Hadoop, Spark, Kafka, and Cassandra. They are already widely used in IT, and ITS can certainly leverage the existing work. We also differentiate between big data and fast data solutions, when the amount of data is not huge, but must be processed in near real-time. A common data gathering and processing architecture is shown, together with possible applications in the ITS domain, when the data is gathered from users, vehicles or transport infrastructure.

Keywords:

Big data, Open-source, Data science

Introduction

As both the vehicles and the transportation infrastructure become more intelligent, they produce an ever-increasing amount of data just waiting to be used. The same is true for transport users: almost everybody is now equipped with a powerful, portable mobile computer (smartphone), capable of reporting multiple metrics in real-time over the Internet. The ITS industry is definitely not the only one where this is true – it is a general trend, the "big data revolution," which is taking the world by storm. This would not be possible without huge technical support; hence, a number of tools are being developed for data influx, storage, analysis, and visualization.

Many of these big data tools are open-source, meaning their source code is freely available and they can be used in a commercial setting free of charge. Such open-source projects often have large user and developer communities, with a lot of learning materials available. In some cases, the tools are gaining so much popularity that they become a de facto standard in their area of use. Apart from community and developer support, there are also options for commercial support if need be.

In our paper, we would like to first define what big data is and what kinds of resources are required to process it, depending on the actual amount of information available. Then we would like to present a couple of the most popular open-source big data tools and typical deployment architectures, with examples of how they can be leveraged in ITS.

Big data & fast data

The term *big data* is often overused in situations where the data can be in fact processed on a single computer. That is why the term *fast data* was introduced to emphasize the fact that there is a shift from analyzing data offline in huge batches, to analyzing data in real-time as it comes (often in the form of micro-batches) and making decisions based on the current stream of information. In total, the amount of data can be huge, and can still be processed offline, e.g., daily, but at any given time, the amount of data coming into the system is much smaller and can be managed in a simpler IT setup.

Of course, it is also possible to have a combination of both: when we want to analyze a stream of data, arriving in amounts exceeding the processing capabilities of a single machine. Until recently, the support for such scenarios was very limited. However, the domain of data stream processing systems is evolving very rapidly, making it possible to build solutions that analyze data streams from thousands of users or devices in near real-time.

There is no precise definition of big data, but anything above 100s of gigabytes can be considered such. If your data is smaller than that, you can still leverage the many advances in open-source machine learning, data analysis, and visualization, without the need to invest in costly server clusters. Very often, a simple setup of a couple of commodity servers can handle such loads.

Fast data tools

For the medium/small data case, there are a number of tools available that you can leverage to process the data using your machine's resources to the maximum. Single-server setups are not only much simpler in setup and management; they also have the possibility to get a global view of data at any time, something that is not always available (or at least costly) in multi-node setups. An increasingly popular architecture is modeling systems in a reactive way [1], which automatically adjusts the various stages of your data processing pipeline, so that no component is flooded with data, which prevents, e.g., out of memory errors. This technique is called *backpressure*.

Some tools worth mentioning here are definitely the many stream processing libraries, such as Akka Streams [2] or ReactiveX [3]. These libraries, first of all, provide a concise, developer-friendly programming interface for defining where data should be read from, how it should be transformed and analyzed and where it should be sent next. Secondly, they have advanced mechanisms for fault-tolerance and managing system resources (also in the form of the already mentioned backpressure) so that developers can focus on the problem at hand instead of low-level programming details.

Big data tools: processing

If your data comes in terabytes, then you will most certainly need to spread the computation across a cluster of machines. The first open-source big data tool that needs to be mentioned here is Hadoop [4]. It allows the processing of data in batches, mostly offline. Its main strength is scalability, as its data processing capabilities are only bound by the number of servers. Its main weakness is that data is typically processed in large intervals, for example, overnight, or even once every week or so. While still useful in many situations, it is often necessary to analyze the data faster.

That is where Apache Spark [5] comes into play. Spark is an increasingly popular platform for large-scale data processing, outperforming Hadoop by orders of magnitude in many tasks, and offering capabilities not available to Hadoop clusters. That is achieved by relying on in-memory computations, making use of the vast amounts of RAM computers are equipped with, as well as a more modern computational model.

Apart from traditional batch computations, Apache Spark also offers other components crucial to modern big/fast-data application development. The first is Spark Streaming, which makes it possible to process data in near real-time, using micro-batching. With this technique, if the incoming amounts of data are huge, the power of the whole cluster is used; with the cluster still being able to serve queries related to the data almost immediately after it enters the system.

The second is Spark ML, a library for scalable machine learning. It has built-in support for computing basic and advanced data statistics, classification, clustering, recommendations (collaborative filtering), as well as initial support for solving optimization problems, which are very common in the transportation industry.

Both Spark and Hadoop offer SQL connectors and interfaces, as well as support for languages such as Scala, Python or R, making it possible for data scientists to leverage their existing knowledge easily to scale their solutions from small to big data.

Spark and Hadoop can also be combined in a single system using the so-called lambda architecture [6]. In this setup, there are two layers: the batch layer and the speed layer. The batch layer stores all incoming information, and then processes data in large batches (e.g., using Hadoop). Once the data is processed, the system is able to respond to queries using the entire information available. However, there is a huge latency as to when these precise queries can be answered. The speed layer processes data in real-time and is often able to give only approximate answers to queries – however, with very low latency. In the speed layer, tools such as Apache Spark or Apache Storm [7] are often used. For data ingestion and buffering, the previously discussed Apache Kafka messaging platform is a popular choice.

Big data tools: ingestion & storage

Data processing components are only one piece of the puzzle. As data comes into a system, it needs to be properly buffered: you cannot always control the rate at which information flows in the system. For example, in the morning rush hours, you'll probably receive much more data than during the day. We do not want, of course, for the system to break when there is a sudden spike of data. Ideally, the system should scale itself (which is possible to achieve, e.g., using Mesos [8]), but if that is not available, it should at least react to the increased load and process the data at a rate that will not overwhelm it. Still, we do not want to lose data, hence the need for a buffering component. A popular solution here is to use Apache Kafka [9]: a high-throughput distributed messaging system (which is the backbone of many well-known internet services, such as LinkedIn). Due to its reliability and performance, it often serves as "first-contact" data ingestion platform, which then feeds other systems.

When data ingestion and analysis is solved, we still need a way to store the data reliably. Here various NoSQL systems are very popular, since they often offer great scalability features, making it possible to store more data just by adding servers. Two notable systems are Apache Cassandra [10] and Riak [11]. Both are great platforms for storing vast amounts of data, coming either from Hadoop processing, Spark streaming processing or directly from Kafka.

Possible ITS applications

Possible applications of the above technologies in the ITS industry come to mind almost spontaneously. There are a lot of devices, and even more are coming, which can be installed in vehicles, be it cars, buses, tracks or trains, which can transmit various data: from vehicle telemetry, through current location and speed, to weather conditions or even recognized infrastructure elements or geographical features. It is possible using the described open-source tools to build a reliable analytical system that can scale itself during the morning rush hours, when the vehicle utilization is at its highest, down to the night hours when the roads are empty. The system could send the data both for real-time analysis, to predict traffic jams depending on the weather or specific road congestion, as well as to a batch-processing component, which could take a global view on data to detect less obvious patterns in how infrastructure is used and how it can be improved. The processed data can then be stored reliably and served back to the user's devices to aid them in their everyday journeys.

ITS applications 1: detecting destination hot spots in New York taxi data

One of the data sources that are made openly available by New York City are all taxi trips from past years, both for the Yellow Taxi and Green Taxi systems [12]. Yellow taxis operate mostly in Manhattan, while Green taxis are only allowed to pick up passengers from other boroughs and drop them off anywhere.

Here we will show how to rapidly build a system to detect taxi drop-off hotspots. We will define a hotspot as an area to which many more passengers traveled than to neighboring areas in a given window of time. The concrete parameters that we used in our tests are:

- 500m x 500m areas, spaced by 100m
- An area needs to have 2.5x more passengers than neighboring areas to be considered a hotspot
- We are looking at all drop-offs within 30-minute windows, spaced by 5 minutes

It is quite possible that using other settings the results would be quite different, but we leave potential experiments up to the reader.

The algorithm to compute the hotspots is quite straightforward. First, we parse the .csv files available from New York City's website. For each month, there are about 1.6 million data points for green taxis and 12 million data points for yellow taxis. Using our definitions, that is not yet big data, and we should easily be able to process this on a single machine. Still, we might create a "fast data" solution, using high-level constructs, and thus speeding up development, and allowing us to do online, streaming analytics.

The data needs to be cleaned as sometimes it contains invalid pickup or drop-off points (in the middle of the ocean). After that is done, each trip is assigned to all areas that it belongs to (using our parameters each trip is assigned to (500/100)^2 = 25 areas), and to each window (again using our parameters that is 30/5 = 6 windows). These operations multiply the data by a factor of 150, but that still should not be a problem for the tools that we have chosen. Once a trip, assigned to a specific area, is added to a window the counter for that area is incremented.

Once all trips for a given window are taken into account (we know this using wall-clock or trip drop-off time), we can "close" the window and find potential hotspots. To do that we scan all area counters looking for one that has many more hits than the eight neighboring areas.

In our tests, we have created two implementations: one using Akka Streams [2] and one using Apache Flink [13]. The code is available on Github [14]. While the Akka version can only be run locally, Flink offers the potential to distribute the computation across the cluster, without any code changes. This was not necessary for this data set, but the same code can be thus reused for much bigger inputs.

Both libraries offer similar combinators to manipulate the data and provide a clear, analyst-friendly API. Looking at the code, it is quite clear what transformations happen and why. Also, thanks to the backpressure support both in Akka Streams and Flink, the computation has bounds on the memory it consumes, making sure no out-of-memory errors occur.

An important feature of the Flink code to note is that it contains only declarative specifications on how to manipulate data; there are no low-level processing details, especially dealing with concurrency. The same code can process data using a single thread, on multiple cores on a single machine as well as deployed on a Flink cluster and run on multiple machines. This makes it possible to scale our data pipeline as the amount of data increases.

What are the results of computing the hotspots (see Figure 1 below)? Some are quite obvious: a constant hotspot is, for example, New York's Penn station (this is, in fact, the most popular hotspot, most of the time receiving over 3x more passengers than neighboring areas), Battery Park/South Ferry transport hub, and the World Trade Center region. In the map below, you can also see some hotspots in La Guardia Airport - these are mostly before Thanksgiving when a statistically significant number of people were traveling. One interesting example is a one-time hotspot on W 90st St near Central Park, on the 17th of November, early in the morning (7:40am). Some event must have been held there which attracted a much-elevated number of passengers.

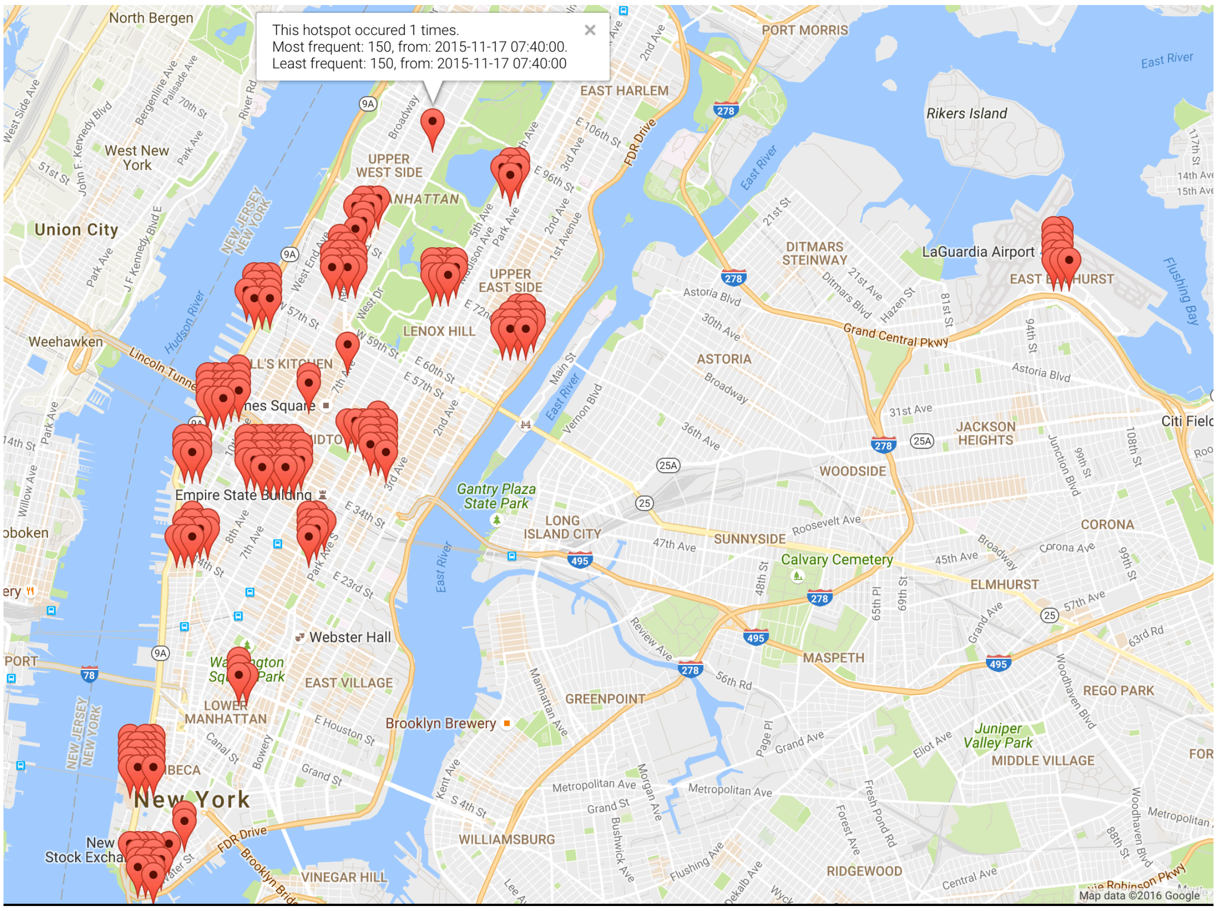
This hotspot occured 1 times.
Most frequent: 150, from: 2015-11-17 07:40:00.
Least frequent: 150, from: 2015-11-17 07:40:00

Figure 1. Example hotspot analysis output for November 2015, New York Yellow taxis

ITS applications 2: architecture of speed data analysis system

To see how these tools could be used in a more concrete ITS system, we will examine how a system for analyzing the speed of vehicles and detecting traffic jams could be architected. First, of course, we need a source of data. In a traditional setting, this would come from vehicle detectors built into the roads, or roadside sensors or cameras. However, a much better source of data nowadays is coming from either devices installed in the vehicles or the mobile phones of transports users. These devices are capable of sending various types of data in real-time; here we are interested in the current GPS position, speed, plus possibly accelerometer and magnetometer indications to make the positioning more precise.

The data is sent over the Internet, through mobile connections. As these are not always available, the device or mobile application has to sometimes buffer the data until a connection is available, and the backend system must allow for data arriving late.

The first component our system needs are endpoints for receiving data. Their main role is to accept connections from thousands of concurrent users and forward for further processing.

They do not store any state for themselves, but they need to be highly available and scalable – after all, it is quite important that the system can receive data. To dynamically start and stop the data-receiving endpoints (as demand grows or shrinks) we might use the already mentioned Apache Mesos [8], which governs the infrastructure. The endpoints (API servers) also need to handle backpressure, in order not to overwhelm any single node; it is always better to reject new connections than to accept too much, and bring even a single node to a halt.

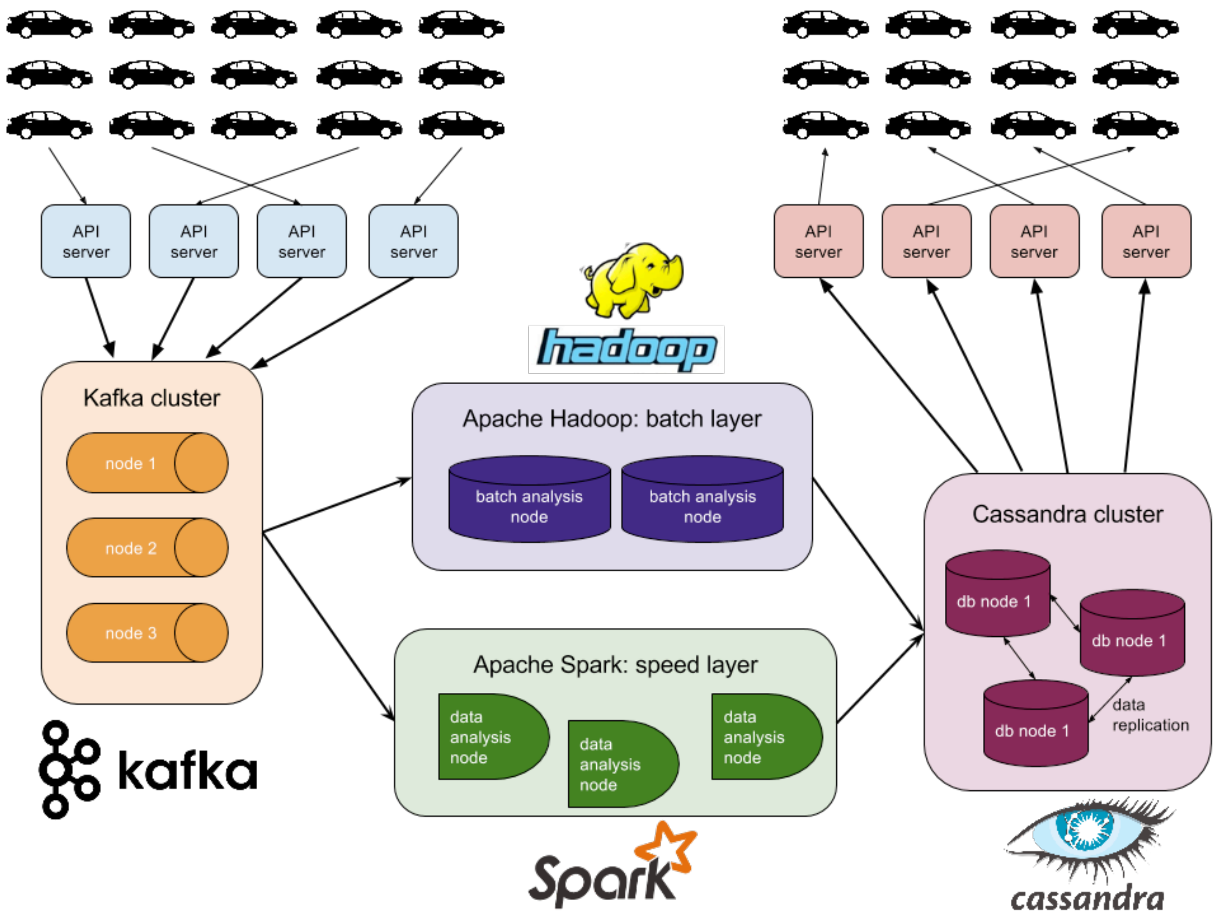

Figure 2. Common data processing architecture applied to ITS

High availability, scalability, backpressure are all traits of a *reactive system*. To implement the endpoints, we could use technologies such as the Play Framework [15] or Akka HTTP [2] combined with Akka Streams. These components are shown in Figure 1 as API servers: receiving (and also sending) data back to the system users.

Each raw received message (consisting at least of a vehicle id, current GPS position, speed, acceleration, and timestamp) needs to be persistently stored for further online and offline analysis. Here, a very safe bet is to use Apache Kafka [9], which is a clustered, scalable message broker, designed from the ground up for streaming use-cases. In Apache Kafka, data is partitioned among a number of servers for scalability, and then replicated on multiple servers for reliability. However, we must be aware that old data is purged from Kafka; that is we cannot use it for historical data storage. All systems, which further analyze the data, will rely on the content coming in from Kafka.

To detect traffic jams and analyze congestion on roads, we first need to group data coming from individual vehicles into road segments (where each segment is, e.g., a stretch of road between two intersections, or a stretch of road of a given length). Here we have many possibilities. One of them is using another Kafka feature, Kafka Streams [16]. Kafka Streams is a library, which makes it easy to transform, join, and aggregate data already present in Kafka, writing it back to a Kafka topic as well. In our case, we need to first assign a road segment to each data point, as well as some form of a speed index (for example, slow - medium - fast). Then we can group the readings into intervals of 5, 10 and 15 minutes using Kafka Stream's windowing features, and emit that to yet another stream stored in Kafka. In each window, the speed of the vehicles has to be somehow averaged, for example by computing the average, median, standard deviation or a percentile to get a good overall measure. All of this can be done using high-level, developer friendly programming interfaces.

Having such transformed data, we can feed the road-segment and speed-index pairs to Spark and use Spark Streaming to detect traffic jams at scale. To do that, we need to find consecutive road segments that are "slow." We could also use an online machine-learning model to analyze the live traffic data and predict where traffic jams might form using Spark ML and warn users.

The raw Kafka data stream, as well as the data transformed by Kafka and Spark, can then be written to a database. Here, Apache Cassandra is often a good choice. It provides high data durability as the data is replicated and very good scalability - the total storage capacity and read/write throughput scales linearly with the number of nodes. Now we can serve data back to the clients, showing them the current (and predicted) traffic jams in their area, using similar endpoints as described in the beginning for receiving data.

We can also batch-analyze the data offline using Apache Hadoop [4] to detect traffic patterns, for example, we can analyze how traffic jams originate (this can be done by constructing a graph of connected, slow road segments and traversing them in time), or how traffic jams move. Using an offline machine-learning model, such a system can suggest adjustments to how traffic lights should be regulated, where to widen/build new roads, or give hints on how to improve the public transport network.

Open-source tools, open data?

There are a lot of open-source big- & fast-data tools available, but what about data itself? That is usually where the value of the many fast-growing companies lies. However, there are increasing movements in opening access to data as well. Prominent examples here are Copenhagen [17] and New York City [18]; the latter makes a number of data sets publicly available, such as subway system turnstile statistics, detailed information on taxi trips or violations of speed cameras spread across the city. This makes it possible both for enthusiasts to analyze and correlate the data in ways interesting, e.g., to public transport planners, but also for businesses to make informed development decisions. As the city owns the transport infrastructure, it is logical that it makes infrastructure usage data available for free to aid its further development.

Many cities are following suit - however, the amount of data is still quite limited, and almost all of the open data available is historical, suitable for offline analysis and processing. One development we might see in the coming years is the availability of *live, streaming open data*. The technology barriers are disappearing; it is very often a matter of investment from the cities. Some live data feeds that might be made open are, for example, live feeds of subway cars, buses, and taxis positions, or the feeds coming from sensors installed at intersections.

However, the biggest source of open data might be transport users via their mobile phones. Just as you track your fitness activities, why not track (in an anonymized, secure way) how you use public and private transport? Which routes you take, how fast you go, this can all be put to great benefits to city planners. A simple mobile application, which sends current speed & position, activated whenever you enter your car, could be provided by the city, for the city's benefit.

About SoftwareMill

SoftwareMill [19] is a software house with a diverse portfolio of customers, from a variety of industries: transportation, logistics, communication, banking, energy, and entertainment. This has allowed us to gain first-hand experience developing a wide range of systems, including various big-data and fast-data processing projects. We think that it would be very valuable to share our experiences. We believe that by combining our technical big data knowledge with ITS domain experts, we could unleash a huge potential in developing next-generation data-driven applications.

Our company is a Select Consulting Partner of Lightbend [20], which develops the Lightbend Reactive Platform, a set of programming languages (such as Scala) and tools for building high performance, resilient, elastic, and message-driven applications. We are also consulting partners of Confluent [21], the company behind Apache Kafka, as well as of Datastax [22], the company developing Apache Cassandra; a number of our developers are certified Cassandra architects. SoftwareMill, Lightbend, Confluent, and Datastax cooperate to deliver top-notch consultancy and development services in various areas, including big data development.

References

1. The Reactive Manifesto, http://www.reactivemanifesto.org
2. Akka Streams, http://akka.io/docs/
3. ReactiveX, http://reactivex.io/
4. Apache Hadoop, http://hadoop.apache.org
5. Apache Spark, http://spark.apache.org
6. The Lambda Architecture, http://lambda-architecture.net
7. Apache Storm, http://storm.apache.org
8. Apache Mesos, http://mesos.apache.org
9. Apache Kafka, http://kafka.apache.org
10. Apache Cassandra, http://cassandra.apache.org
11. Riak, http://basho.com/products/
12. New York City taxi data, http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml
13. Apache Flink, https://flink.apache.org/
14. Source code used for hotspot analysis using Scala, Akka and Apache Flink, https://github.com/softwaremill/its-pres
15. Play Framework, https://www.playframework.com/
16. An introduction to Kafka Streams from the Confluent blog, http://www.confluent.io/blog/introducing-kafka-streams-stream-processing-made-simple
17. City of Copenhagen open data, http://data.kk.dk/
18. New York City open data, https://data.ny.gov/
19. SoftwareMill, https://www.softwaremill.com
20. Lightbend, https://www.lightbend.com
21. Confluent, http://www.confluent.io/
22. Datastax, http://www.datastax.com/