Big Data Infrastructure

Martin Neumann, Ian Marsh, Bjorn Bjurling, Ahmad al-Shishtawy

1 Scope, goal and audience

This report's aim is to give an overview of the technology components currently being used in big data environments as of late-2017.

The focus is on *open source based* products that have company or foundation support. Within the big data processing platforms, the Apache foundation is the one most synonymous supporting free development, support, and hosting of solutions. The reason for restriction within this report is that open source-based solutions have a strong presence on the market, and many commercial solutions are based on them.

The report provides an overview of the technologies used as well as the general computational models they implement. An important aspect of the technology, is we differentiate where possible between batch and stream processing, where the former is typically file by file, and the latter is record by record, omitting storage where possible.

The audience of this document is the technical person, but without specific knowledge in distributed systems or machine learning. To maintain a readable report length, we have kept each paragraph "bite-sized" to keep details to a minimum, but still informative. Our goal is to provide the reader with the necessary background in order to read the ever-growing volume of literature, courses and consultancy options available to companies considering investing in big data practices. An annotated reference list is given at the end of this document, which should be used for further information on most topics within these 30 pages.

2 Contents

3 Big data background

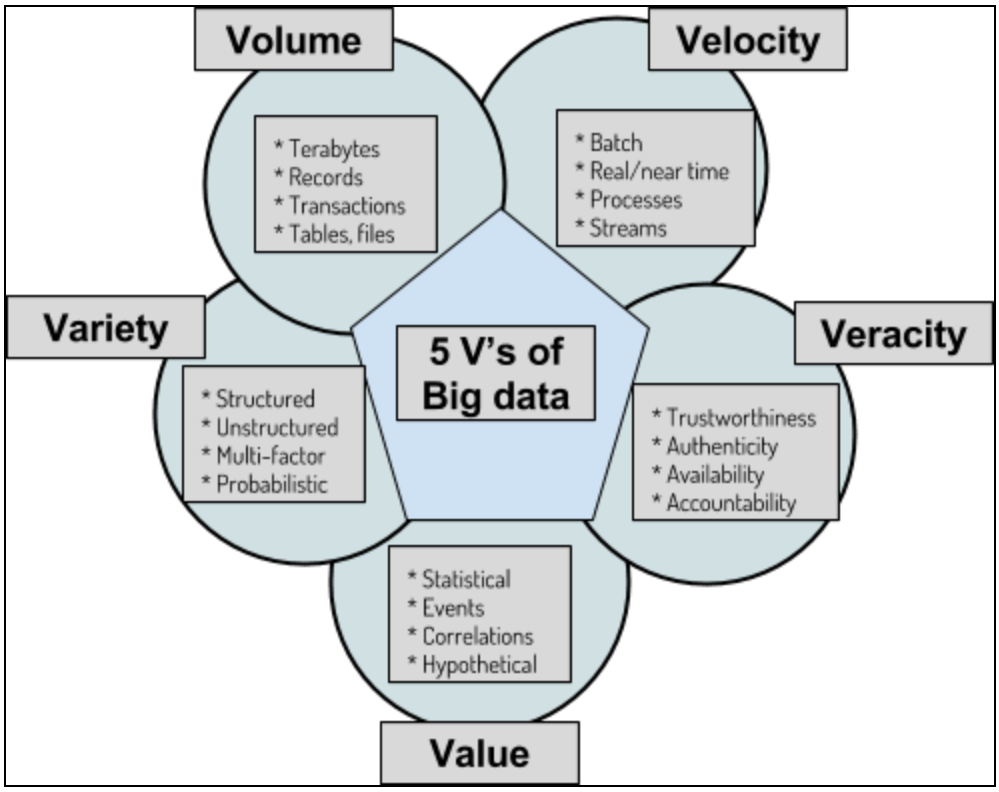

Figure 1: Big data processing

In order to understand the general ideas of big data within the workplace, IBM initially defined four words using the letter "V". With the addition of a fifth, "Value", the terminology became a standard manner in which to start describing big data practices. In the next paragraphs we explain what each "V" stands for, see figure 2 too.

I. ***Volume*** refers to the vast amounts of data that is generated. 90% of the world's data was created in the last 2 years. This could either be in form of large files such as HD-Video of large amounts of small files such as transaction logs. The data commonly is too big to fit into a single machine so storage and access become challenging. Distributed job processing such as Hadoop or Spark

transparently process large datasets by assigning sections of the data code onto several machines in a cluster. The traffic flow data from Trafikverket covers 13 years, per minute resolution at about 500 metres comprises 400 TBytes. To put some further figures on well-known industries, WalMart process 100 million customer purchases per week [W1]. Google data center performance metrics produce 3 million sample points per second from their machines, monitoring [G1].

II. ***Velocity*** refers to the speed at which new data is generated and how fast it to move through the analytics pipeline. The New York Stock Exchange captures about 1 TB of trade information daily. Companies need to process this data in real time in order to react to changes in the market. The data volume flowing through these data streams often changes over time, where peak periods often reach a multiple of the average volumes. Flink is a streaming framework that achieves high rates by using in-memory, data and parallel processing. Data from cars and trucks indicating their position, status and service levels generate large data amounts flowing into a system. Other high velocity data rates from Twitter is around 6K/sec [T1], Google searches are in the 40K per second [G1]. The stream of data from the Large Hadron Collider (LHC), is around 300 GB/s before filtering and 300 MB/s after. CERN accumulates 25 PB per year [LHC1, LHC2].

III. ***Variety*** refers to the different forms of data that we collect and use. Data comes in different formats: structured, semi-structured or unstructured. Semi-structured data has some structural component such as identifiers but also contains unstructured data such as tag clouds. 80-85% of all the world's data is now unstructured text, audio, video, click streams, log files etc. The STRADA accident data from the Swedish police comprises different types of input. This is mostly because the data is hand-entered, by a police, rescue staff or hospital staff, however it is in a database with well-defined records.

IV. ***Veracity***. The average billion-dollar company loses about $130 million a year due to poor data management [4]. Veracity refers to the quality of data. Data is often incomplete, inconsistent or biased in nature. This is especially true when data from different sources is combined. iii) A typical example is where the space dimension is not quite correct (i.e. not a GPS location). iv) In BADA the "Ds_reference", a grid reference does not give us sufficient accuracy to infer vehicle position.

V. ***Value*** is about generating knowledge from the data. This is a big challenge since most analytical tools, visualisation techniques and algorithms do not scale. As a result highly specialized frameworks and algorithms are necessary that are only applicable in narrow conditions. The value in the safety information is high, as accidents cost in people's lives, recuperation, insurance, reparation and so on. Anything that can reduce the number of accidents is beneficial for all society. The value of the traffic flow data is also high, particularly in "diagnosing" congestion, but can be used in infrastructure planning and even extensions to the highways.

4 A typical data processing pipeline

The Hadoop ecosystem consists of modules that help program the system, manage and configure the cluster, manage data in the cluster, manage storage in the cluster, perform analytic tasks. The majority of the modules we will describe are the components and related technologies. Figure 2 shows a flow diagram for a typical data processing pipeline. Data is generated by a process and then stored. Visualisation and analytic jobs read the data and enrich, modify or visualize it. A general agreement on how the data should be structured allows for the different components to interact. The whole structure executes on a hardware or virtualized environment, Figure 2.

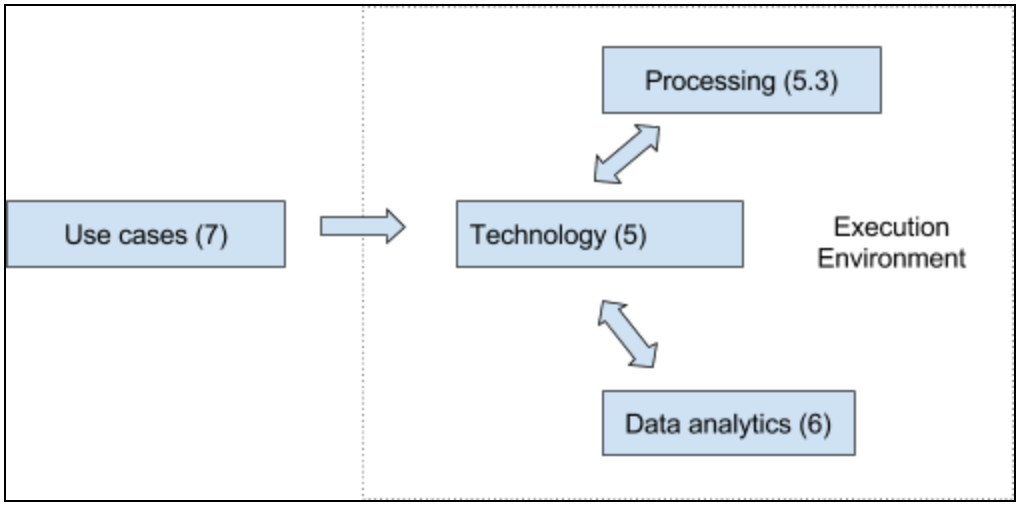

Figure 2: A data processing pipeline, section numbers in parenthesis

We will consider the technology in the next section, both examples of batch and streaming processing. Some words on the background on analysis is given.

Visualisation is treated in a page and for the data sources, we give 2 from the BADA project: The hazard warning system and the traffic flow examples are in section 7.

5 Technologies

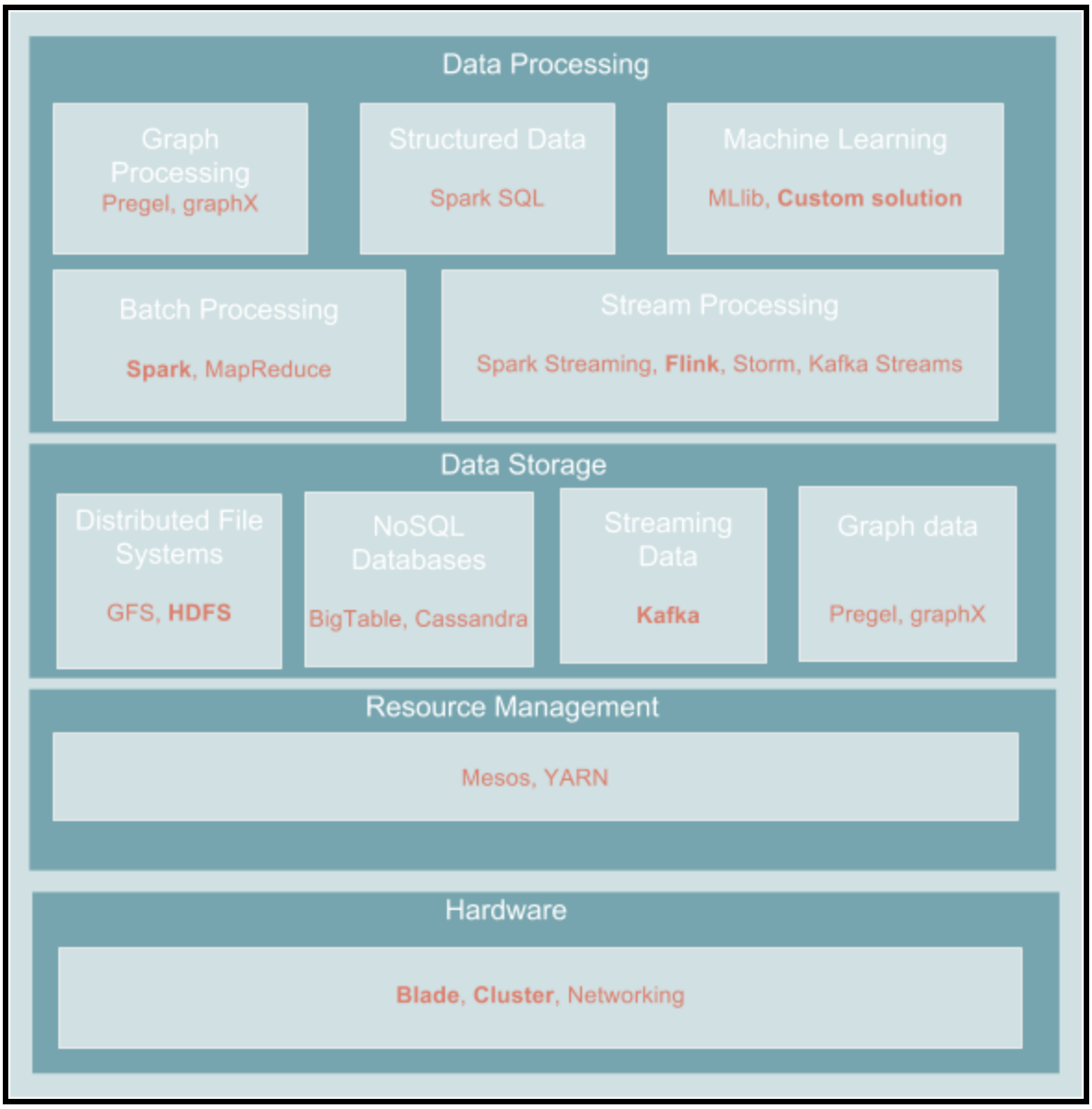

Figure 3: Data processing in a nutshell,
bold sections are covered in this report (source: Amir Payberah)

5.1 Server technologies

A single machine or personal computer is typically classified as a single CPU with multicore architecture, with typically 250GB to 1T disk and between 8-32GB of main memory, so why use multiple machines in a rack-mounted configuration?

1. Data does not fit a single machine
2. Single machine cannot process the data in time
3. Data too complex to analyse on single machine

A server system is designed to handle requests opposed to clients which issue them. In practice this means CPUs with many cores, more main memory and faster interconnect than our client/laptop systems. Hypervisor support for virtualisation is of course necessary in server systems. Power supplies and power smoothing are also necessary in a server system, as reliability is a core issue.

Clusters are typically built from a rack-mounted PC with server-like configurations, specific cooling, modular architecture, hot-swappable and energy efficient. A common architecture is called *blade,* which has IEEE size compliance. A blade server requires a blade enclosure, which holds multiple servers, a 1 unit (480mm x 44mm) being the standard physical configuration. A rack is 42 units which typically contains 128 servers.

Figure 4: Probability of server failure over its lifespan

How is data transferred within a cluster? Blade servers generally include integrated or optional network interface controllers. These are typically Ethernet or host adapters for Fibre Channel storage systems. Converged network adapters combine storage and data via one Fibre Channel over Ethernet (FCoE) interface are commonplace. In many blades at least one interface is embedded on the motherboard and extra interfaces can be added using add on (mezzanine) cards. A blade enclosure can provide individual external ports to which each network interface on a blade will connect. Alternatively, a blade enclosure can aggregate network interfaces into interconnect devices (such as switches) built into the blade enclosure or in networking blades. A Cisco UCS B200 M4 blade server, achieves up to 80 Gbps throughput using Fibre interconnect [CiscoBlade]. The market is dominated by Cisco, which through acquisitions, has 40% of the US market [wikipedia].

Access speeds: In terms of flexibility and speed storing data in memory is a large gain. Essentially the difference between memory and disk access is considerable, as shown.

```
L1 cache reference ......................... 0.5 ns
Branch mispredict ............................ 5 ns
L2 cache reference ........................... 7 ns
Mutex lock/unlock ........................... 25 ns
Main memory reference ...................... 100 ns
Compress 1K bytes with Zippy ............. 3,000 ns = 3 µs
Send 2K bytes over 1 Gbps network ....... 20,000 ns = 20 µs
SSD random read ........................ 150,000 ns = 150 µs
Read 1 MB sequentially from memory ..... 250,000 ns = 250 µs
Round trip within same datacenter ...... 500,000 ns = 0.5 ms
Read 1 MB sequentially from SSD* ..... 1,000,000 ns = 1 ms
Disk seek ........................... 10,000,000 ns = 10 ms
Read 1 MB sequentially from disk .... 20,000,000 ns = 20 ms
Send packet CA->Netherlands->CA .... 150,000,000 ns = 150 ms

(* Assuming ~1GB/sec SSD)
```

Table 1: Access speeds for computing resources

Disk storage is the building block on which all other components in big data processing systems depend. Storage has two main roles: first, it acts as connector between

different systems. In some sense, storage in this role is used in a real time setting, where different processes can coordinate in space (e.g. same file, socket ID) and time. A classic example is one process writing its output to storage, whilst another reads the same data. Reader-writers are sometimes called producer-consumers in some literature. One requirement in this role is that the storage is relatively fast, in seek, read and write times. Storage's second role is to act as a persistent place so data needed for future use is not lost. Failure, rollbacks and of course analysis are just some of the uses of stored data. In some cases the storage for longer term needs is not as fast as the first role, and may be larger in capacity.

One important aspect that all big data systems have in common is that they treat data as immutable. This is quite a paradigm shift compared to local or cloud-based storage. Once a datapoint is written it cannot be modified. The chief reason for immutable storage systems is that systems are distributed, and allowing for random read-write operations access would degrade performance. This however does not mean that data cannot be updated. Most system have a versioning system that allow to write new versions of the data that can then be accessed.

5.2 Data storage

Big Data platforms require a storage system capable of storing large datasets in a reliable way and serve them efficiently when they are processed. Many new data storage systems have been proposed to fulfill the demands of different Big Data use cases. We can classify these storage systems into two classes. 1) Batch data stores (or data at rest), and 2) Streaming data stores (or data in motion). The main difference is that batch data stores are optimized to store and serve very large amounts (Volume) of data for long period of time. While streaming data stores are considered as a temporary storage designed to deal with high arrival rate of new data (Velocity) and delivering fresh data to its consumers as fast as possible while guaranteeing that data is not lost before it being processed.

5.2.1 Batch storage : The HDFS file system

HDFS is the current industry standard in storage in big data systems. The hadoop file system (HDFS) is an open source implementation derived from the Google File System (GFS). It is built to handle large files. A typical file in HDFS is gigabytes to terabytes in

size. It provides high aggregate data bandwidth and can scale to hundreds of nodes in a single cluster. HDFS applications need a write-once-read-many access model for files. A file once created, written, and closed need not be changed. This assumption simplifies data coherency issues and enables high throughput data access.

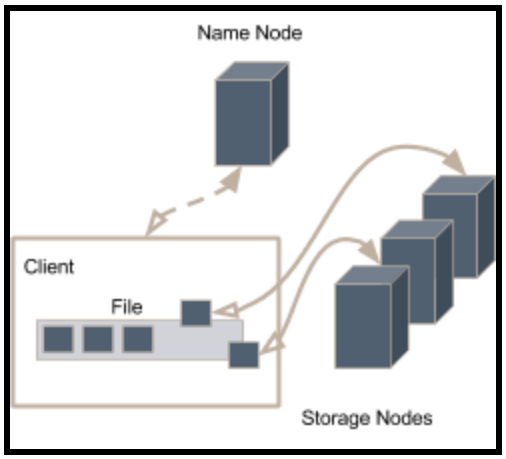

Figure 5: Relationship between name and storage nodes

HDFS runs in a cluster of machines, consisting of storage nodes and a name node. Files stored in HDFS are split into blocks of a fix size, distributed and replicated between the storage nodes. The meta information about each file as well as the location of all of its blocks is stored at the namenode. When reading from HDFS a client asks the name node for the locations of the blocks and reads them directly from the storage nodes. When writing the client request a number of blocks from the namenode and receives a list of storage nodes where they are written. Reading and writing can be done to all involved storage nodes in parallel, so read operations usually bound by network throughput instead of disk speed. The inner workings of the read and write operation are usually hidden by the client software. HDFS has become the basis for most big data systems and has good support and tooling. Many companies, such as Cloudera and Hortonworks offer solutions and support for HDFS.

5.2.2 Stream ‘storage’ : Kafka

Apache Kafka is an open-source stream processing platform that provides a unified, high-throughput, low-latency platform for handling real-time data.

Most applications using streaming data have strict latency requirements. For high speed trading for example, information that is several minutes old is not important anymore. As

a result it is often enough to persist the data for a limited amount of time to allow for failure recovery. It's not necessary to store data indefinitely. In cases where the data is also used for batch processing it's typical to stream it into a different storage solution such as HDFS or Cassandra.

Its storage is essentially a publish-subscribe message queue and its design stems from transaction log processing. Apache Kafka the main message broker used for big data streaming applications. There are connectors and clients available for many systems that allow an easy integration into a streaming architecture.

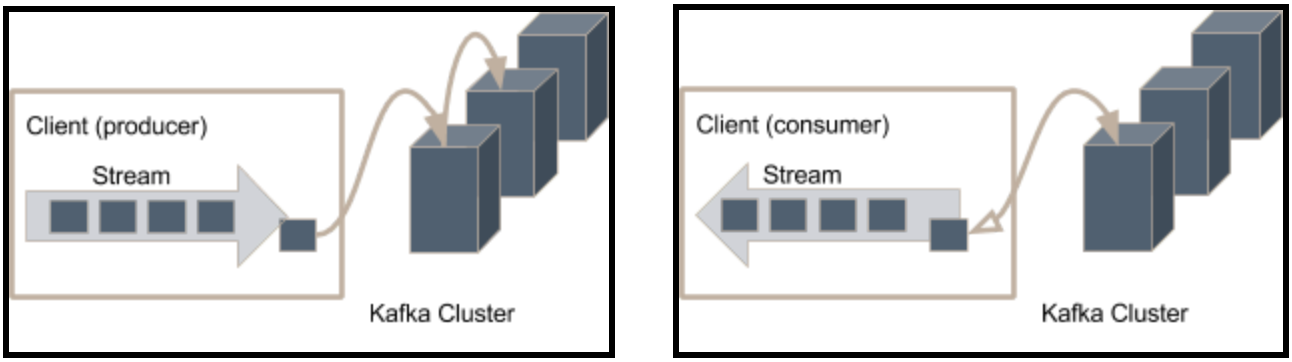

Figure 6: Relationship between clients and the Kafka cluster

Kafka runs in a cluster of machines as shown in Figure 6. When a client writes to it, it needs to specify a **topic.** For streaming application the **connector** aspect of storage becomes the main focus. Therefore these storage systems are also called message brokers.

Data is written and consumed on a per record base at very high speed by many different actors. The Kafka ecosystem at LinkedIn handles over 800 billion messages per day. At the busiest times, LinkedIn's system processes 13 million messages per second. LinkedIn runs over 1100 Kafka brokers organized into more than 60 clusters. Kafka is written both in Scala and Java.

An example of a producer in java is below. We include code to illustrate what is below the concepts in this document, plus a start point on how to read source code.

```
import java.util.Properties;
import org.apache.kafka.clients.producer.Producer;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerRecord;

public class SimpleProducer {

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

  // Check arguments length value
  if(args.length == 0){
   System.out.println("Enter topic name”);
   return;
  }

  //Assign topicName to string variable
  String topicName = args[0].toString();

  // create instance for properties to access producer configs
  Properties props = new Properties();

  //Assign localhost id
  props.put("bootstrap.servers", “localhost:9092");

  //Set acknowledgements for producer requests.
  props.put("acks", “all");

  //If the request fails, the producer can automatically retry,
  props.put("retries", 0);

  //Specify buffer size in config
  props.put("batch.size", 16384);

  //Reduce the no of requests less than 0
  props.put("linger.ms", 1);

  //The buffer.memory controls the total amount of memory available to the producer for buffering.
  props.put("buffer.memory", 33554432);

  props.put("key.serializer",
   "org.apache.kafka.common.serialization.StringSerializer");

  props.put("value.serializer",
   "org.apache.kafka.common.serialization.StringSerializer");

  Producer<String, String> producer = new KafkaProducer
   <String, String>(props);

  for(int i = 0; i < 10; i++)
   producer.send(new ProducerRecord<String, String>(topicName,
     Integer.toString(i), Integer.toString(i)));
        System.out.println(“Message sent successfully”);
            producer.close();
  }
}
```

https://www.tutorialspoint.com/apache_kafka/apache_kafka_simple_producer_example.htm

Figure 7: Java producer example code

5.3 Processing frameworks

5.3.1 Batch processing

The basics are that certain amounts of data are read and processed. The classic usage is the file, which can range from a few bytes to gigabytes. In a batch system, the whole file is read, often sorted, and then processed. It may be matched, merged or “compared” with other sources, correlated or deeper processed, for example two faces for similarity. Larger amounts of data can be read, but files are typical. Batch processing frameworks are Hadoop, Spark, Hopsworks and many considered in this document. In a stream processing context the data is compared record-by-record in a database, or line-by-line in a file or event-by-event in a messaging system. One example is next.

5.3.2 Stream processing

In Batch processing, programs execute for a finite amount of time until all the available data are processed and results are produced. On the other hand stream processing is designed to continuously execute to process infinite data (such as sensor measurements or user interactions with a web service) as long as data is being produced. Batch processing can be viewed as a way to analyze and get insights from large volumes of historical data while stream processing is viewed as a tool to rapidly analyze and react to the current situation. Examples of stream processing frameworks include Flink, Spark streaming, Storm, and Kafka streams.

5.4 Database implementation

How data is consumed often affects on how it should be stored. The traditional database approach (SQL) requires some changes in a very large data setting. If searching over a very large number of records is necessary, indexing the data becomes

essential. Traditional database technology does not scale to the volume or velocity required by many big data applications since transactions, plus the ability to change data rapidly become unscalable. `Cassandra` and `elasticsearch` offer big data solutions in 2017. For data amounts larger than 500GB, we think Zeppelin+HopsWorks is a better option. A BADA safety example is given in the first use case 6.1. Larger data amounts are being done in the queue detection of second use case, 6.2. Technically, very large databases use key value stores, rather than a SQL normalised tables.

6 Data analytics

6.1 Exploratory data analysis

Is an important phase in understanding large datasets. Obtaining the tools, understanding the problem and team up to speed can be done via an early data exploration phase. That is read the data, the data, show simple time-series, histograms, correlation between 2 variables (scatter) or multi-variable correlation. An important aspect is to try and show the early results from the data in a trimmed down, command-line manner. In our cases, the first used case was i) a series of PDF plots, an interactive `plot.ly` plot and a series of plots as an animation. The goal was to show which roads are suitable for detecting traffic queue buildups. ii) Using logstash, kibana and elasticsearch (see below). The traffic flow data is 5 GB / month, and constitutes the sensor data of public highways in Sweden. We use this data to show how vehicles are distributed in space (roads, cities) and time (min, hour, day, weekly, seasonal).

6.2 Information design

Is the practice of presenting information in a way that fosters efficient and effective understanding. The term has come to be used specifically for graphic design for displaying information effectively. Information design is closely related to the field of data visualization and is often taught as part of graphic design courses Information design is explanation design. In the use case of queue detection within BADA, we need to be able to show space and time in plots, as the dependent/chosen `(x)` variables, and average speed, flow, density, as independent/calculated `(y)` variables. The space variable might also include map visualisations [vis]. (Interactive) data visualization, is a growing area, where users interact with the data, from simple mouseover events, to

complex reordering of the data in real time to expose possibly hidden interesting artifacts. We use visualisation in the traffic flow queue buildup scenario.

6.3 Descriptive statistics

Often overlooked for quantitative methods, descriptive statistics use linguistic terms or quantitative terms encoded into emotional terms “acceptable, poor, excellent”. These terms, such as the quality of a voice call or feedback at an airport (red-green smiley buttons). The value of distilling down complex and long calculations based on data to simple expressions, is useful, “we are at the tail of the traffic of queue” to further decision making. Management science relies heavily on descriptive statistics, as the parameters are well known, human intuition is a strong indicator of a correct “ground truth” in the absence of quantitative data and is understood by all.

6.4 Inferential statistics

With inferential statistics, conclusions that extend beyond the immediate data alone is the goal. An example from real life is to infer from a small sample of people what a population might think. Alternatively, inferential statistics help determine whether a difference between groups, is statistically significant important, or can be explained by plain randomness.

Three uses of the `t-test` in the BADA project are 1) using inferential statistics to infer between types of drivers, for example a driver of a truck or a car. (Recall that we have only flow data). Another example is infer between a fast or slow driver in the dataset. Again, we have no ground truth, but could confirm a faster driver by testing against the frequency of lane changes and not. Here, one would construct a hypothesis, “fast drivers change lanes more frequently” true or false?. A 3) use is to compare the average performance between two groups one method is the `t-test`. In BADA we use the `t-test` like this to verify if the density is really a queue, as it requires less samples than it’s larger `Z-test` brother.

6.5 Visualisation

The way the brain processes data, for example using charts or graphs to represent large amounts of complex data can be easier than reports or tables. Data visualization is a quick, easy way to convey concepts in a universal manner – and one can experiment with different scenarios by making slight adjustments. By identifying areas

that need attention or improvement, clarifying which factors influence customer behavior. Web-based tools using Javascript run in the browser, allow users access to any site on a public IP address, it also allows the provider to move the data into a cloud service (permitting caching, protection against hacks, DoS etc.). Also upgrades can be done without the consumers actually noticing. For example visualisation helps one understand function of two variables, `f(x, y)` with time (by animating) dynamics. Indeed, we use this idea to show queue buildups, showing speed, density and flow as a function of space and time (see use case 2).

6.6 Deep learning

TensorFlow is an open source software library for numerical computation using data flow graphs. The architecture makes computation to 1+ C/GPUs using a single API. Developed by Google for machine learning and deep neural networks research, it is useable to general applications too. In our case, within BADA, we will use TensorFlow to learn which queue buildups are normal, for a given space and time `(x, t)`, and which are anomalies for that place and time.

7 BADA use cases

7.1 Hazard warning

Figure 8 shows the architecture designed for the hazard warning BADA scenario. The datasource is a HDFS file containing a list of geolocations. They are read by a Flink process that applies a time delay to simulate an event stream and writes the data back to a kafka topic. Another Flink process is consuming the so called topic. For each record it contacts and external `openstreetmap` server to find the language surrounding the geocoordinate. Based on the result it flags the record to be in a rural, residential or if not found unknown area. It pushes the results into `elasticsearch`. The results can then be queried and visualized using `kibana`. All components, besides the external server, are executed on a Hopworks cluster (described below).

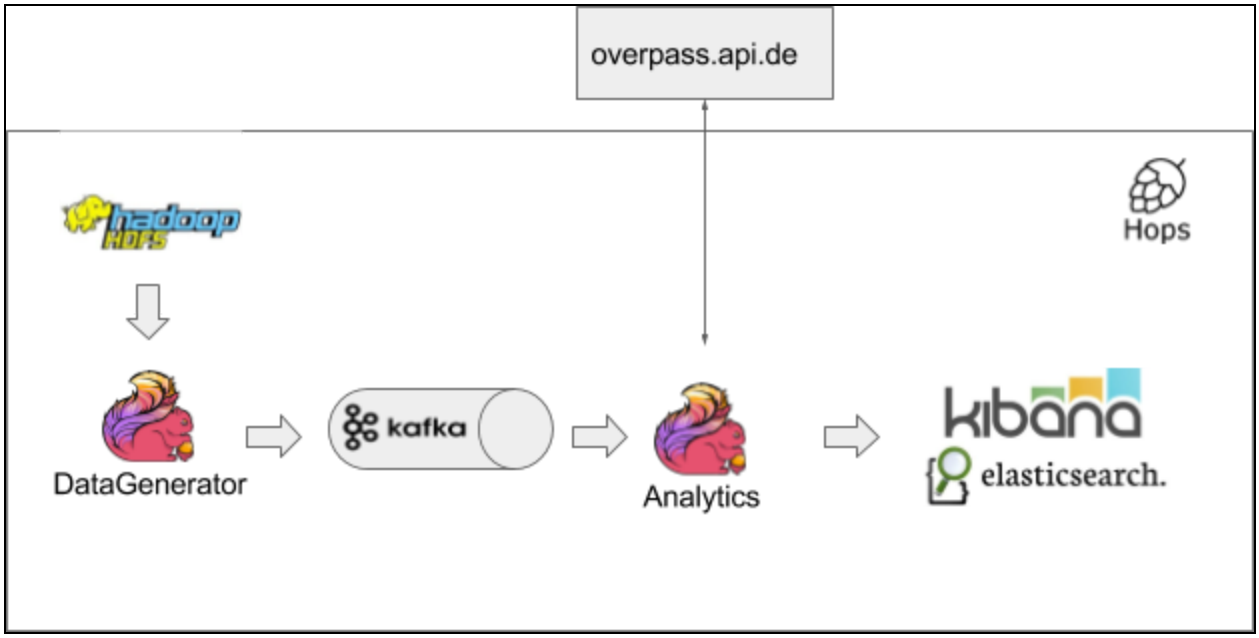

Figure 8 : Kafka streaming architecture

7.2 Traffic safety use case

In this case we visualise the STRADA dataset, which is a national information system of road transport accidents in Sweden. The key work we have done in this use case are:

- Data exploration:
- Gain insights into the data
- Know the limits of the dataset
- Elasticsearch/Kibana: Efficiently search and visualize large datasets

These are shown in the RISE-SICS demonstration shown in Figure 9.

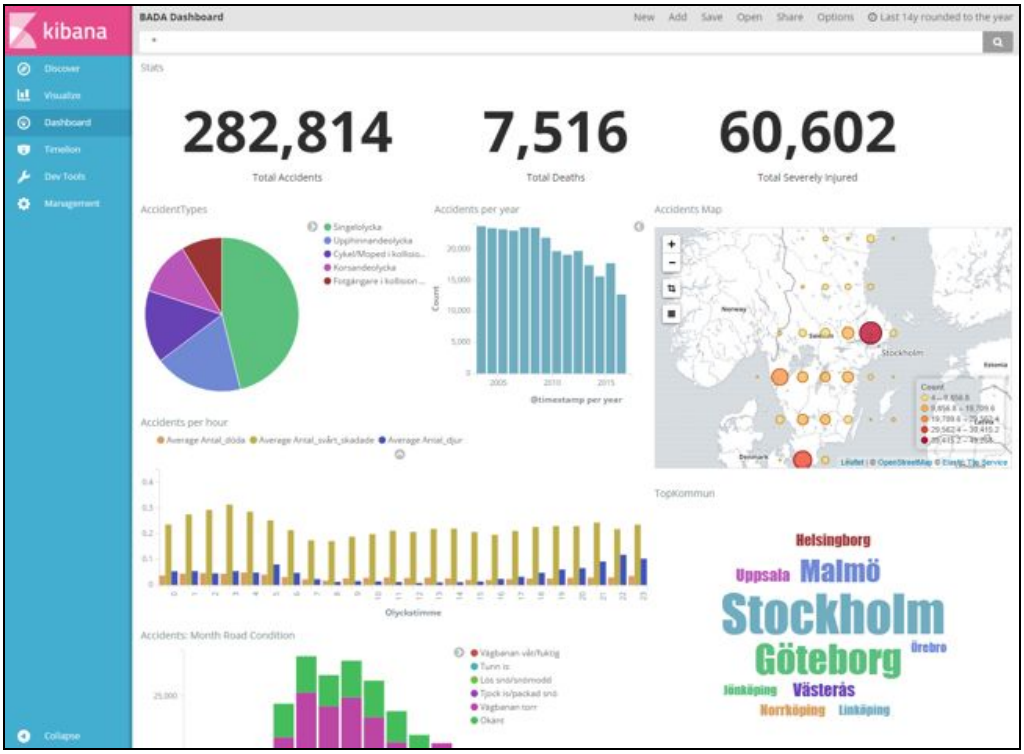

Figure 9: Traffic safety dashboard

7.3 Traffic flow example with queue buildup

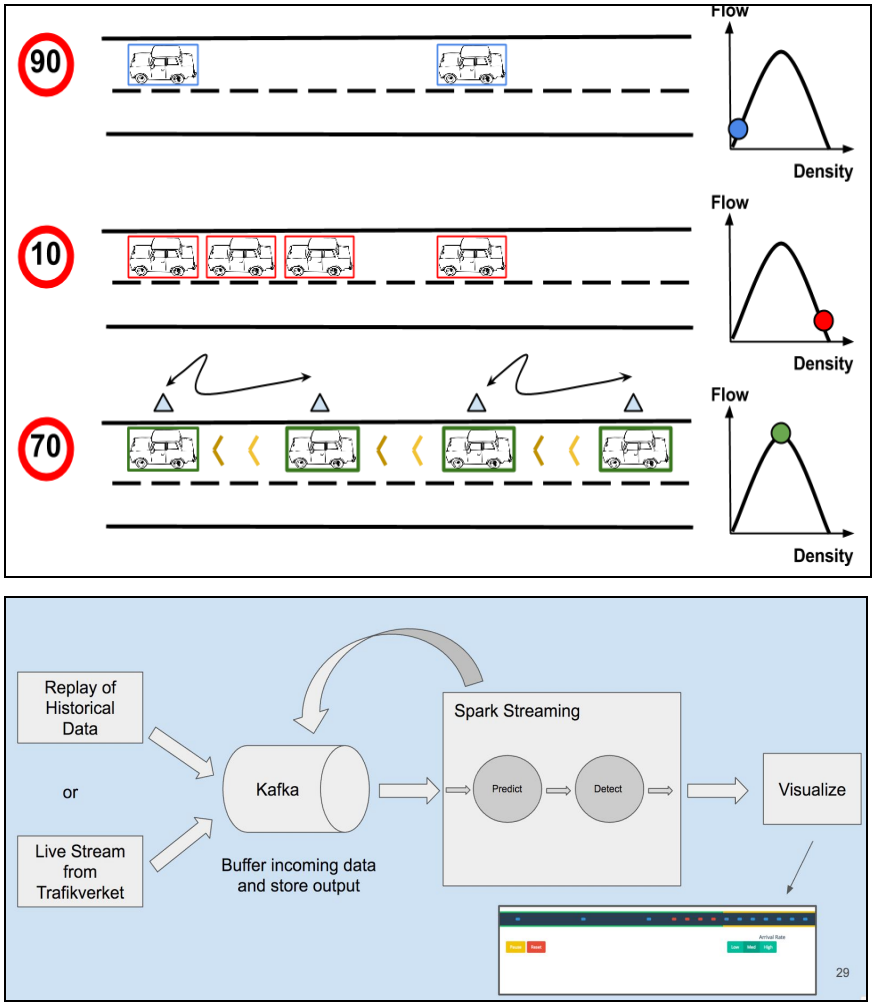

Figure 10: Traffic flow detection.
Top figure, road example, bottom figure, big data solution.

An example of batch processing, analysis is the congestion detection currently ongoing at RISE, Volvo, Scania and trafikverket. Using 11 years of data from Trafikverket and Volvo, we are working on queue detection [Ian1]. The general idea is to use the density changes in traffic to detect queue buildups. A similar project is at TU Berlin and the Flink startup Data Artisans [Art1].

The queue detection is based on the simple equation flow = density * speed, so where red cars are in the left figure and where the red circle is on the flow vs. density curve. Once detected, we will use the data to train our algorithm to remove the positions and times where queues always build up (Gothenburg 08:00 Monday morning). Once trained, we will use data from the traffic system, *streamed in real time*, into our system. This is shown in the right figure. Note, the system must be trained for the system, using Kafka to detect unusual queue accumulations.

7.4 Future use cases

7.4.1 Self-regulating traffic flow

A use case we envisage is a self-regulation of traffic flow. That is, vehicles regulate their distance in order to achieve optimal flow, *for each road section.* Information for which distance to maintain can be fed to the driver in one of three forms: 1) Via the vehicle itself, 2) by the roadside infrastructure, 3) or by an app to guide the driver. By using information about the macroscopic traffic flow and microscopic behaviour of the individual vehicle, it is possible to be at the optimal spacing, or density, to achieve the best flow, at any place or time. From the data we have and collaboration of the vehicle manufacturers and some research, we can fill the gap between completely autonomous vehicles, and the current situation of humans totally controlling the vehicles.

7.4.2 Extension of the hazard warning light scenario

By pressing the hazard warning sign, vehicles indicate that they are slowing down or stopping. We could extend this work by signalling to other vehicles that this is happening to the traffic flow. Currently Scania/Volvo trucks do not feed data into the Kafka queue, and this is clearly something that could and should be done in the future.

7.4.3 Mining 11 years of flow data

With 11 years of traffic flow, there is a large amount of data to be analyzed, with use cases of reducing congestion over sweden, towards a ‘smart motorway’ or better estimation of arrival times (GPS doesn’t take into congestion). Correlation of weather and traffic conditions is very much a topic in Sweden, sensors in tyres, road conditions.

8 A brief technology chronology

In 2002 Doug Cutting and Mike Cafarella crawled the Web and indexed content in order to start to produce an Internet search engine. They needed a scalable method to store the content of their crawling and indexing. The standard method to organize and store data at the time were relational database management systems (RDBMS), accessed via the SQL language. They quickly discovered neither were appropriate for Internet search engine data construction use, essentially cost, scalability and reliability (to failure) were their documented hurdles. In 2003/4 Google published two papers, one on the Google File System (GFS) [1] and a second on a programming model for clustered servers called MapReduce [2].

8.1 Hadoop

Cutting and Cafarella incorporated these technologies into a project they called Hadoop, named after a stuffed toy elephant. The GFS soon migrated into the Hadoop file system, or HDFS. Yahoo began using Hadoop and it soon spread to other organizations, it is still one of the predominant big data platforms even as of 2017.

1. The Hadoop Distributed File System (HDFS)
2. The MapReduce programing platform
3. The Hadoop ecosystem, tools to store and organize data

In 2012, version 2.0 of Hadoop was released as YARN, Yet Another Resource Negotiator. Again it's purpose is cluster management. YARN exists between the data and MapReduce, allowing additional tools to be inserted into the big data processing stack. Examples of tools include Spark and Giraph. YARN does not replace MapReduce, it solely provides a uniform way for tools to run on a Hadoop cluster.

8.2 Spark

Spark is designed to provide a flexible computing model that supports many of the multipass features that don't exist in MapReduce. It accomplishes this in order to reduce

the amount of data that is written to and read from disk. It is a complete replacement for MapReduce that includes its own work execution engine built on three core concepts:

1. Resilient Distributed Dataset (RDD) contain data that you want to transform or analyze. They can be read from an external source, such as a file or a database, or created by a transformation (next).

2. Transformations modify an existing RDD to create a new one. A filter that pulls ERROR messages from a log file is a transformation.

3. Actions analyze an RDD and return a single result. E.g., an action to count the number of results identified by the ERROR filter.

Significant work in Spark is done by the functional programming language, Scala [3]. It combines object orientation with functional programming, Spark has recently added support for Python, via PySpark, it is growing in popularity via using and sharing notebooks (commands, data, and plots). Commercial support for Spark is provided by many companies, in the US one of worthy of mention is [Databricks].

8.3 HopsWorks

The BADA project utilises HopWorks, a next-generation distribution of Apache Hadoop:

1. Hadoop as a Service
2. Project-based multi tenancy
3. Secure sharing of dataSets across projects
4. Extensible metadata supporting free text search
5. YARN quotas for projects

The key innovation that enables these features is a new architecture for scale-out, consistent metadata for both the HDFS filesystem and the YARN Resource Manager. The new metadata layer enables the support of multiple stateless NameNodes and TBs of metadata stored in a MySQL Cluster Network Database. It is a distributed, relational, in-memory, open-source database [Jim]. It enables HopsWorks to provide tools for designing extended metadata, whose integrity with filesystem data is ensured through foreign keys in the database. Extended metadata enables the implementation of

quota-based scheduling for YARN, where projects can be given quotas of CPU hours/minutes and memory, thus enabling resource usage in Hadoop-as-a-Service to be accounted and enforced. Support is provided by LogicalClocks in Stockholm.

8.4 Flink

Apache Flink is a platform for efficient, distributed, general-purpose data processing. It features powerful programming abstractions in Java and Scala, a high-performance runtime, and automatic program optimization. It has native support for iterations, incremental iterations, and programs consisting of large DAGs of operations. Flink Streaming is an extension of the core Flink API for high-throughput, low-latency data stream processing. The system can connect to and process data streams from many data sources like RabbitMQ, Flume, Twitter, ZeroMQ and also from any user defined data source.

8.5 Big data processing architecture

The platforms and tools discussed so far are all parts of an ecosystem (Figure 3) with different components interacting together and playing their role in the big data application. Two best-practice architecture emerged that outline how to connect these components together to achieve high throughput and low latency data processing. The Lambda [Lambda] Architecture proposes the use of two layers, a batch layer for managing historical data and a real-time layer to rapidly process incoming data. Queries are answered by combining batch view with the real-time view. The Kappa Architecture is a new vision where their community believes everything should be streaming-based. Data is stored as an append-only immutable log. Supporters of Kappa architecture argue that this simplifies the platform by eliminating the need for a separate batch processing system and storage.

9 Summary

This report aimed to give a brief overview of the big data technology components. We chose open source projects. We focussed on *open source-based* products, most available from the Apache foundation.

The report provided an overview of the technologies and some principles, for example messaging passing and data pipeline. We differentiated between batch and stream processing with perhaps the prospect of harmonisation via the kappa framework.

Within 30 pages we have tried to give the basics of big data processing, with reference to the projects SICS/RISE is involved in. The annotated reference list next should be a place for the reader to continue with.

Good luck, SICS-RISE

10 Annotated reading list

General

[BD1] Big Data, for Better or Worse: 90% of World's Data Generated over Last Two Years. Report from a research institute in Norway, SINTEF, about when and how large amounts of data have been generated. An interesting read.

[SemiTech1] https://www.oreilly.com/ideas/ideas-on-interpreting-machine-learning. A nice technical overview, covers things not in this document.

[Transform1] http://www.amazon.in/Big-Data-Revolution-Transform-Think/dp/1848547927/262-1981668-3147756. Informative text on the revolution taking place now.

Commercial

[Jim] https://www.youtube.com/watch?v=pr_9jF-wL3M Strongly consistent metadata talk, SICS data science day. Motivates the design choice for HopsWorks. Technical in nature, but good presentation.

[I1] http://www.internetlivestats.com/twitter-statistics/

[I2] http://www.internetlivestats.com/google-search-statistics/. Simple rolling counter of the Internet use. Although shallow technically, it provides some quantitative counts on the Internet, updated on a second basis :-)

[G1] https://plus.google.com/+JamesPearn/posts/VaQu9sNxJuY. Somewhat old (2012), but provides some stats on Google's data centers.

[W1] http://www.businesspundit.com/stats-on-walmart. Some numbers on the data usage by Wal-Mart, the largest retailer in the world.

[N1] New York Stock Exchange Ticks on Data Warehouse Appliances.

[R1] The Rising Data Deluge Opportunity.

[B1] http://www.businesspundit.com/stats-on-walmart/

Use cases

[Amaz]
https://www.amazon.co.uk/Big-Data-Practice-Cases-Extraordinary/dp/1119231388.
The book argues promotes itself by many existing companies to fail to grasp the important steps in large data environments. It tries to fill the knowledge by showing how major companies are using big data every day. Companies include WalMart, LinkedIn (Kafka), NetFlix, Rolls-Royce, just to mention a few.

Spark

[OreillySpark]
http://www.amazon.in/Learning-Spark-Lightning-Fast-Data-Analysis/dp/9351109941.
Easy introduction to Spark via the popular Oreilly series.

[O1]
http://shop.oreilly.com/product/0636920022466. Very useful glossary on big data terms. Well worth having by the side at a start into the big data and ML practices.

[O2]
http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/WritingYarnApplications.html. From the Apache forum, it describes, at a high-level, the way to implement new Applications for YARN.

Technical publications

[T1] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, "*The Google File System*", Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles (SOSP 2003) https://research.google.com/archive/gfs.html
The file system has met Google's storage needs. It is widely deployed within Google as the storage platform for the generation and processing of data used by their services as well as research and development efforts that require large data sets. The largest

cluster to date provides hundreds of terabytes of storage across thousands of disks on over a thousand machines, and it is concurrently accessed by hundreds of clients. The paper presents file system interface extensions designed to support distributed applications, discuss many aspects of their design, and report measurements from both micro-benchmarks and real world use.

[T2] Jeffrey Dean and Sanjay Ghemawat, "*MapReduce: Simplified Data Processing on Large Clusters*", Proceedings of the 6th Conference on Symposium on Operating Systems Design and Implementation (2004). https://static.googleusercontent.com/media/research.google.com/en//archive/mapreduce-osdi04.pdf

[T3] Chen Zhang, Hans De Sterck, Ashraf Aboulnaga, Haig Djambazian, and Rob Sladek. "*Case study of scientific data processing on a cloud using Hadoop*". In High performance computing systems and applications, pp. 400-415. Springer Berlin Heidelberg, 2010.

Programming

[P1] The scala programming language**.** Popular programming language based on the JVM with functional extensions to Java, which makes it suitable for big data applications, and its adoption by companies like Databricks. A very good introduction at https://www.scala-lang.org.

[GK1] https://kubernetes.io/

Streaming

[K1] https://insidebigdata.com/2016/04/28/a-brief-history-of-kafka-linkedins-messaging-platform. History of the development of Kafka though LinkedIn's eyes. Six years of development and over 1.4 trillion messages processed, this is an account of the trials and tribulations of Apache Kafka and it's commercial route at LinkedIn.

[K2] https://engineering.linkedin.com/kafka/running-kafka-scale

[K3]
https://cwiki.apache.org/confluence/display/KAFKA/Kafka+papers+and+presentations

[Lambda] http://lambda-architecture.net
Lambda architecture is a software architecture pattern that combines batch and stream processing in the same framework. The batch layer is used to store and query large volumes of historical data. The real-time streaming layer is used to overcome latency problems in the batch layer by providing real-time view based on fresh data.

[Kappa] http://milinda.pathirage.org/kappa-architecture.com.
Kappa architecture is a software architecture pattern. Rather than using a relational DB like SQL or a key-value store like Cassandra, the canonical data store in a Kappa Architecture system is an append-only immutable log. From the log, data is streamed through a computational system and fed into auxiliary stores for serving. Kappa Architecture is a simplification of Lambda Architecture. A Kappa Architecture system is like a Lambda Architecture system with the batch processing system removed. To replace batch processing, data is simply fed through the streaming system quickly.

[K4]
https://hackernoon.com/a-super-quick-comparison-between-kafka-and-message-queues-e69742d855a8

Analytics and Visualisation

[V1] http://www.informationisbeautiful.net/

[MartinN]
https://docs.google.com/presentation/d/1bX8r2g3f4MeJNy-6uJO7e48hm2Eez6OjCjIEKnwxhro/edit#slide=id.g211632ad54_0_291

Courses

Data Intensive Computing, KTH, 2016 Amir Payberah, Seif Haridi, https://www.sics.se/~amir/id2221. Very complete course, with material online.

BADA

[Art1] http://training.data-artisans.com/exercises/connectedCar.html

[Ian1] https://docs.google.com/presentation/d/1NFDBYErmfwWzJTMt8p-bpTqUNX8C0HakZA-GM9eayGk/edit#slide=id.p