FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

A Practical Guide To Big Data

Student	Dominik Kaisers Student number: 3094093 Course: Master Information Systems Engineering FH Aachen, Aachen
Supervisor	Prof. Dr.Eng. Martin R. Wolf FH Aachen, Aachen

Aachen, November 22, 2017

Abstract

The goal of this thesis was to create an introductory guide into the field of Big Data that focuses on the practical aspects of applying Big Data technologies and methods. The guide was to be designed to help interested individuals to start off in the field.

Based on actual real world applications and implementations of Big Data, gathered from scientific publications, a basic foundation of knowledge compiled. It included how Big Data is being applied in real word scenarios, what technologies are being used and what methods of data analysis are prevalent in the field. Furthermore, research on the effectiveness of different forms of communication regarding the conveying of information was surveyed to help in choosing a basic structure for the guide.

The guide was to be accompanied by a set of pre-installed and pre-configured Big Data tools. These tools were used in various examples of Big Data processing and analysis that the guide was based around. To keep the focus of the guide on the actual application of Big Data, the goal was to require no additional effort in order to use the tools.

In the end, the resulting guide showed that educational material about Big Data with a focus on the practical aspects can be produced, but that additional research may be required to provide a more realistic software infrastructure with less limitations and constraints. A ready to use Big Data infrastructure was discovered to be a good foundation for any practically oriented research into Big Data as its creation was single most time consuming aspect of this thesis.

Zusammenfassung

Das Ziel dieser Masterarbeit war die Erstellung einer einführenden Anleitung in das Fachgebiet Big Data mit einem Fokus auf den praxisorientierten Aspekten der Anwendung von Big Data Technologien und Methoden. Die Anleitung sollte darauf ausgelegt werden, interessierten Personen einen einfachen Start in das Fachgebiet zu bieten.

Basierend auf realen Anwendungen und Umsetzungen von Big Data, zusammengestellt aus wissenschaftlichen Publikationen, wurde ein grundlegendes Wissensfundament erarbeitet. Dieses beinhaltete wie Big Data in realen Szenaren eingesetzt wird, welche Technologien hierfür genutzt werden und welche Methoden zur Datenanalyse in der Praxis vorherrschend sind. Zusätzlich wurden Forschungsergebnisse inspiziert, welche sich mit der Effektivität von verschiedenen Formen der Kommunikation bezogen auf die Vermittlung von Informationen auseinandergesetzt haben, um bei der Entscheidung für eine grundlegende Struktur der Anleitung zu helfen.

Die Anleitung war von einer vorinstallierten und vorkonfigurierten Zusammensetzung verschiedener Big Data Tools zu begleiten. Diese Tools wurden in unterschiedlichen Beispielen zur Big Data Verarbeitung und Analyse verwendet, auf welchen die Anleitung basiert. Um den Fokus der Anleitung auf die tatsächliche Anwendung von Big Data zu fokussieren, wurde es als Ziel angesehen, dass kein zusätzlicher Aufwand betrieben werden müsste um die Tools benutzen zu können.

Letzten Endes zeigte die resultierende Anleitung auf, dass informatives Lehrmaterial über Big Data mit einem Fokus auf den praxisorientierten Aspekten erstellt werden kann, aber auch, dass zusätzliche Forschung nötig sein könnte um eine realistischere Software-Infrastruktur bereitstellen zu können, die weniger Eingrenzungen und Restriktionen aufweist. Eine solche gebrauchsfertige Big Data Infrastruktur wurde als eine gute Grundalge für jede Form von praxisorientierter Forschung über Big Data angesehen, da die Erstellung der Infrastruktur im Falle dieser Masterarbeit der Aspekt war, welcher die meiste Zeit gekostet hat.

Eidesstattliche Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe.

Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Ort, Datum	*Unterschrift*

List of Figures

Contents

1 Introduction

There has been an exponential increase of research into and interest in the field of Big Data over the last few years. One of the driving forces of that trend is the ever increasing amount of data that is being generated. It has been estimated that in the near future the total amount of data that has been produced "will double about every other two years". (Chen, Mao, and Liu 2014, p.1) Combined with Big Data application's shown ability to work with extremely different sets of requirements and prerequisites, it would be negligent of companies and other organisations not to utilise the available data to gain a competitive advantage. (De Mauro, Greco, and Grimaldi 2016, p.7)

The number of scientific publications regarding Big Data is clearly on the rise as well. (Wienhofena, Mathisena, and Romanb 2015, p.1) But as Oussous et al. (2017, p.17) state, most of the research does not focus on Big Data as a complete infrastructure being applied in a real-world context. It constrains itself to a single aspect of the topic instead. Overall, Pospiech and Felden (2012, p.6) found that a great majority of research is purely technical in its nature. This is a problem for interested people starting out in the field and the problem this thesis is trying to tackle. The goal is to create an introductory guide to Big Data focusing on the practical aspects of its application. Providing the necessary tools in a pre-installed and pre-configured state together with example data sets, the guide should explain how to extract new information from a set of data utilising Big Data technologies and methods.

The rest of the introduction will concern itself with a more in-depth definition of the problem space, a more detailed description of the goals for this thesis and a description of how these goals are being approached. After that, the following chapters of the thesis will focus on the information gathered as basis for the guide, the conception and implementation of the guide itself and a small evaluation of the end result based on previously defined criteria. The introductory guide itself can be found in the appendix.

1.1 Problem Space

The main problem this thesis concerns itself with is the lack of information focusing on how to apply the tools available in a Big Data software infrastructure with a realistic example. Before going into more detail on how this is a problem and why, a few other key issues need to be addressed in order to establish a baseline of knowledge about Big Data.

Therefore this section will start with providing a definition of the term Big Data, as it is widely used with differing intentions. As De Mauro, Greco, and Grimaldi (2016, p.129) show, there are scientific works that only include the data itself into their understanding of Big Data while others expand the field to include the technologies, methods and impact of Big Data as well. Precise terminology is important to accurately define what should be part of the guide. Additionally, more details on the impact of Big Data are being given. On the basis of those two points, the significance of the stated problem can be derived.

1.1.1 Definition of Big Data

If one looks at any research article on the topic of Big Data, chances are high that it will include a discrete usages of the term Big Data. While there is some consensus found within those usages, they still differ slightly from one another. This results in the term being vague and used with diverging intentions in mind. (De Mauro, Greco, and Grimaldi 2016, p.122)

This thesis forgoes to create its own definition, but instead utilises the one derived by De Mauro, Greco, and Grimaldi (2016) in their meta study of the research field. They analysed 1437 articles about Big Data to find commonalities between the usage of the term and what features it includes. They summarised their findings into the following definition: “Big Data is the Information asset characterised by such a High Volume, Velocity and Variety to require specific Technology and Analytical Methods for its transformation into Value.”

This definition highlights the different aspects of Big Data that are often being looked at singularly, but are intrinsically intertwined. The data itself forms the basis of any Big Data application. On the other side of the equation stands the value that is to be extracted. It is the overall goal of any Big Data application to create such value from the data. The third and final element of any Big Data application are the technologies and methods utilised. Choosing the appropriate tools is based heavily on the combination of the available data and aspired value. Depending on their composition, different architectures for the Big Data infrastructure are necessary for the extraction of value to work.

1.1.2 Big Data's Impact

Big Data's impact in recent years has been enormous, mostly driven by its vast number of application fields. As De Mauro, Greco, and Grimaldi (2016, p.7) have pointed out, correlation analysis has been one of the main applications and can be found in "a range of domains, from epidemiology to economics". They further make the point that with those kinds of possibilities, companies need to utilise Big Data throughout their whole organisation in order to stay competitive in the market.

Another point to make is about data utilisation. With the possibilities shown by Big Data today, in the future we will want to utilise all of the available data to extract previously unknown insights into a domain. (Chen, Mao, and Liu 2014, p.205) As previously stated, the rate at which data is being generated is rising constantly and the types of data will only get more varied and more complicated with time. In combination both points indicate that Big Data's future impact will be even bigger as it has been already.

In conclusion, Chen, Mao, and Liu (2014, p.34) predict that "Big Data will not only have the social and economic impact, but also influence everyone's ways of living and thinking, which is just happening."

1.1.3 The Need for a Practical Guide

With this baseline of knowledge about Big Data and its impact established, the need for a practical guide can be easily derived from the previously stated points. In summary, the demand for expertise in the field will keep on rising with more and more organisations changing various aspects of their business to incorporate Big Data into the decision making process.

As shown in the definition of the term Big Data, the process of extracting value out of vast amounts of data is a complex one. Depending on the type of data available in the specific use case and the specific value that is desired, appropriate technologies and methods need to be chosen. As current information mostly looks into a singular aspect of Big Data (Oussous et al. 2017, p.17), it is not a good resource for starting out in the field.

This is where the practical guide, that is being designed as the overall goal of this thesis, has its distinguishing feature. It aims to give an introduction into the field of Big Data that covers a wider field of relevant topics and how they work together. Through providing the necessary software infrastructure and exemplary data, interested people can directly jump into working with Big Data on a practical level instead of focusing on theoretical or technical aspects. The accompanying software infrastructure has the additional benefit of allowing further experimentation beyond the examples used in the guide.

In the end the guide should provide a basic understanding of the tools involved that allows for further experimentation and further learning about the possibilities of Big Data in more detail. Understanding what the available tools can do and how to use them enables people to apply the knowledge in other use cases that are more related to the respective circumstances of an interested person. The ability to keep using the provided software infrastructure for such experimentation further supports people's ability to learn about Big Data on their own terms.

1.2 Goals

The overall goal for this thesis is straightforward and has been mentioned multiple times already: the creation of an introductory guide to the field of Big Data that focuses on the practical aspects of applying the available tools. In order to clarify in more detail what exactly should be included in that, this section defines a number of sub goals that define the boundaries of this thesis and the guide itself.

- The technologies and methods used in the guide should have relevance in real world scenarios to make it easier to apply any knowledge gained from the guide to other use cases.
- The technologies and data sets used in the guide have to be available freely and reliably.

- People working through the guide should not have to focus on installing and configuring software, but instead should focus on the application of the taught knowledge. The software infrastructure required for the guide should be supplied as ready to use as possible.
- The examples utilised in the guide should reflect a possible real world scenario as much as possible to make it easier to apply the knowledge gained from the guide to other use cases.

These sub goals have in mind that the guide should mainly focus on applying Big Data tools. There is a lot of information on technical and theoretical aspects of Big Data, but a guided application of the available tools on an exemplary data set is hard to come by. In addition, relying on open and free tools as well as freely usable data sets further minimises the entry barrier for people interested in the field.

1.3 Approach

To ensure the achievement of the aforementioned goals, the creation of the guide was split into three phases. The first one focused on gathering all the information necessary in order to produce the guide with its accompanying software infrastructure. The second phase was concerned with the actual creation of the guide using the previously gathered information and the last phase consisted of a brief evaluation of the guide to ensure that it successfully serves its purpose. The criteria for the evaluation were specified before the creation of the guide itself, as these criteria had the possible benefit of helping to focus the guide on the right aspects of the topic.

The first phase of the approach was further subdivided into three areas of interest: existing implementations of real world Big Data applications, available technologies and methods for Big Data and, lastly, scientific insights into the communication of information. The combination of these three areas of interest forms a solid foundation on top of which the practically oriented guide could be created. It ensures that the content would relate to different types of real world scenarios and that the way the information is presented, is the most effective one in terms of easy and fast comprehension by the consumer of the guide.

The second and third phases of the approach were straightforward. The accumulated information of phase one was used to create a guide that includes technologies, methods and examples that represent possible real world applications to a certain degree. The content focuses on the technologies and methods and how they can be used in order to extract new insights out of the supplied exemplary data sets. Finally, the guide was tested against the previously defined evaluation criteria in order validate that the end product of the thesis achieved the set goals.

In terms of information procurement, literature research of scientific publications as well as the documentations of various Big Data technologies were the only sources of information.

2 Analysis of the Current State

This chapter is concerned with the gathering of all information necessary for the conception of the guide. There are two parts to this. First, the question of how Big Data is being applied in the real world was answered. This includes use cases, processes, technologies and methods that relate to Big Data. The resulting knowledge was used to concentrate the guide on the most important parts of Big Data to introduce people to the topic in a practically oriented manner.

Second, studies on the effectiveness of different ways of communicating information were looked at, focusing on written explanations and ones in the form of video. The results were used in the conception phase of the thesis to decide what medium the guide should use. Other factors like the effort involved in creating the guide or the ability to distribute the guide in different ways were taken into consideration as well.

2.1 Applications

In their scientific survey Chen, Mao, and Liu (2014, p.204) predicted a vast social and economic impact by Big Data. So much so, that it would better people's lives and that the public and private sectors would lose a significant amount of competitiveness by forgoing to utilise Big Data. Looking through examples of Big Data applications mentioned in different scientific works, there seems to be no foreseeable end to Big Data's potential usages. Oussous et al. (2017, p.3-4) mention use cases in the public power supply, healthcare, the Internet of Things, public utilities, transportation and logistics as well as politics.

And this list does not even include two of the most common proponents of Big Data opportunities: the finance sector and the internet sector (Chen, Mao, and Liu 2014, p.198). Especially companies whose main business lies within the realms of the internet have been utilising and advancing Big Data technologies and methods. Streaming services, social networks, online retailers and other internet services rely heavily on the values and insights Big Data can extract out of the enormous amounts of data that are generated by those services and their users. (Pääkkönen and Pakkala 2015)

Each of those applications consists of different interacting parts that form an overall software infrastructure. And each of the parts handles a specific challenge that arises when dealing with Big Data. Oussous et al. (2017, p.4-7) introduced a number of categories for these challenges that also work well to label the different parts of a Big Data infrastructure.

First of all there is the *management*, which is mainly concerned with the accessibility, storage and security of the data. The data itself needs to be *aggregated* from different sources that may be internal and also external. Another important part of any Big Data application is the *cleaning* of the data to ensure the reliability and quality of the source. And last but not least there is the *analysis* part to extract new value and insights out of the data using techniques appropriate for Big Data.

In fact most Big Data applications are structured in a very similar fashion. The biggest difference between individual Big Data implementations stems from the type of data being analysed. Batch processing of very large data sets requires different technologies and methods than the processing of streaming data. Besides that, the differences are mostly nuanced and possible to implement within the same available technologies.

Because of this, there have been multiple attempts to create reference architectures for Big Data software infrastructures that are based upon real world examples. Pääkkönen and Pakkala (2015) looked at exemplary Big Data implementations by Facebook, LinkedIn, Twitter, Netflix and others. The resulting reference architecture includes abstract solutions for both types of data and ranges from the data sources to the final usage of the extracted insights. The insights can be used for visualisations that can help organisations to make decisions as well as be implemented directly into end user applications. A good example for this is the recommendation system of Netflix which analyses the behaviour of the users to recommend shows and movies to watch. The results are directly integrated into the end user interface.

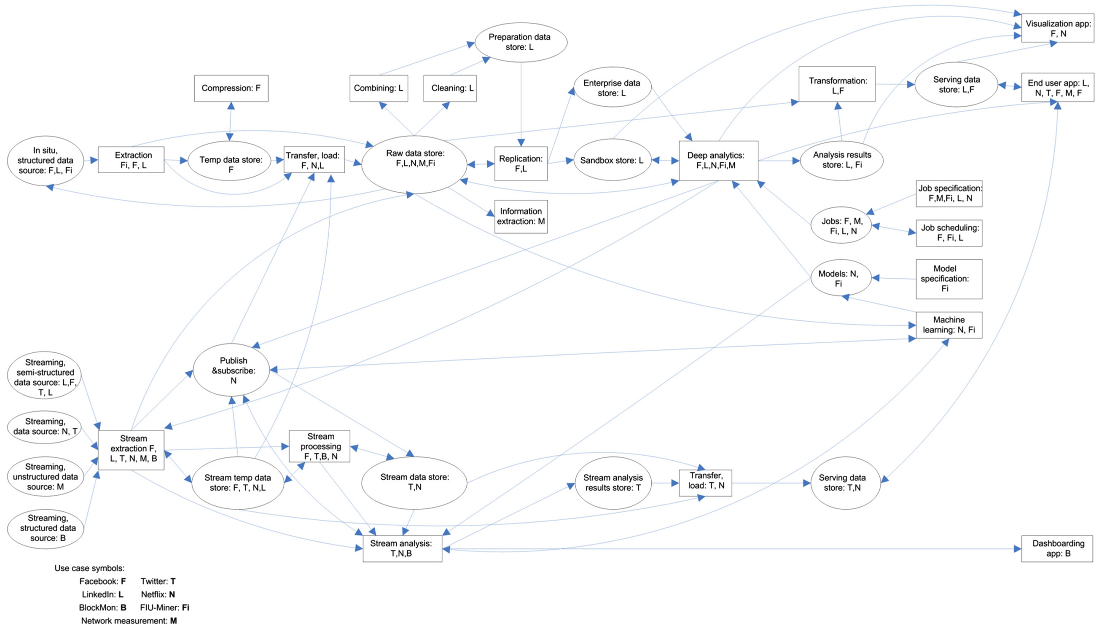


Figure 1: Big Data reference architecture by Oussous et al. (2017).

Looking at the reference architecture in 1, the previously mentioned challenges by Oussous et al. (2017) can also be identified. For example, there are subsections of the architecture concerned with the aggregation, storage and analysis of data. Each of the components of the reference architecture is connected to other ones using arrows that represent the flow of data. Since the general flow of data is the same for all example implementations that were used to derive the reference architecture from, a *Big Data Value Chain* (Bhadani and Jothimani 2016, p.8-11) can be drawn that represents the different steps involved in Big Data implementations to go from data sources to the desired value.

2.1.1 Big Data Value Chain

The *Big Data Value Chain* was conceptualised by Bhadani and Jothimani (2016, p.8-11) and is based upon the general concept of a value chain introduced by Michael Porter in his 1985 book *Competitive Advantage: Creating and Sustaining Superior Performance*. It describes the seven phases all Big Data applications consist of. Each phase or step has a distinct function in the overall system similar to Porter's value chain where value is added to the product or service in each step. (Porter 1985)

The seven steps of the *Big Data Value Chain* according to Bhadani and Jothimani (2016, p.8-11) are the *generation*, *collection*, *transmission*, *pre-processing*, *storage* and *analysis* of data as well as the final step of *decision making*. The value chain can be seen in 2. The next paragraphs discuss each of the steps in more detail and explain what happens inside a Big Data system for the value to be extracted from the data.

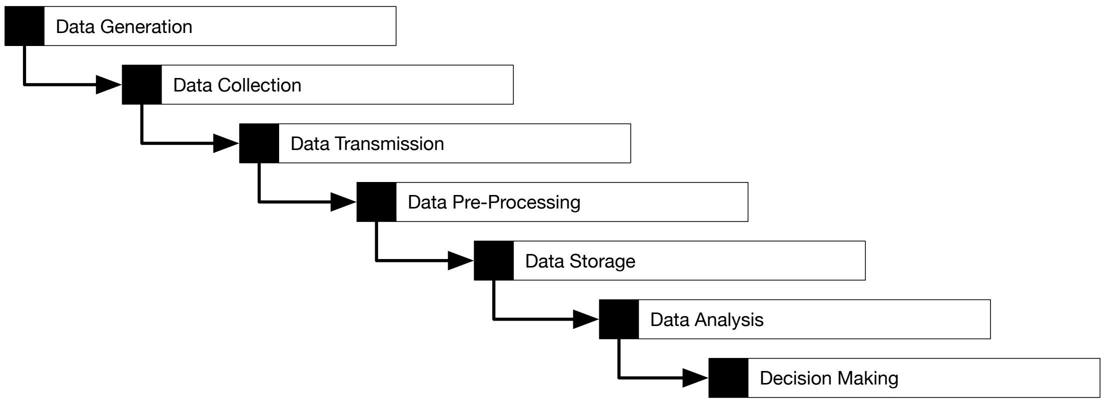


Figure 2: The seven steps of the Big Data Value Chain

Any situation in which Big Data technologies and methods are being applied starts off with the *generation* of the data itself. Without data, there would not be a need for Big Data in the first place. There are many cases in which the most of the data used is being created by the same system whose user experience is being enhanced by the application of Big Data. Popular examples for this are Netflix and Twitter's recommendation systems where on the one side the user behaviour is recorded and used as data for the system while also being influenced by the results of applying Big Data techniques to said data in order to produce recommendations.

Second, the generated data needs to be *collected*. There two categories for Big Data sources that come into play in this step: internal and external sources. Internal sources are ones that reside within the organisation that is developing the Big Data application. This includes user data, sensor data, general business data, log files as well as any other types of information generated by the day to day operations of the organisation. Besides internal data, the collection of external data is important, too. External data may be weather reports, traffic reports or publicly accessible content from social networks. Combining internal and external data can result in insights of higher accuracy and quality. (Oussous et al. 2017, p.4) One of the reasons for this is the context that can be provided by external data sources.

Once the data is collected, it needs to be *transmitted* to its storage location from where any further processing takes place. There are two different types of transmission, depending on where the data source is located. External data sources may need to be handled differently than ones which reside within the internal software infrastructure. This applies to the *transmission* as well as the *collection* steps of the Big Data Value Chain. While internal data sources can be accessed without any limitations in most cases, external data sources may be subject to limitations like the frequency of accessing the data or the amount of data that can be collected or transmitted within a period of time.

Before the data is stored long-term and can be analysed, it almost always requires some form of *pre-processing*. Bhadani and Jothimani (2016, p.9-10) list three distinct sub-steps of data pre-processing. In the *integration* sub-step, the data of the different available sources is combined. This may include transforming different types of data to a standardised format before importing the data into the long-term storage. Another sub-step is the *cleaning* of the data. Even high quality data sources may include noise and errors in the data. Cleaning may delete or modify the data to raise the overall data quality. Common problems are inaccuracies, missing data points or inconsistencies. The last step before storing the data permanently is to remove any redundant data. Redundancies in the data are not only costly in terms of storage space and processing time, but they can also lead to further inconsistencies.

Another additional challenge mentioned by Chen, Mao, and Liu (2014, p.176) is *data confidentiality*. Data sets often contain sensitive and/or personalised information. For reasons of security and privacy, storage of those types of information may be unwanted. In some cases like medical studies regulations exist that prohibit the storage of such data in an unaltered way that allows the identification of individuals. Therefore, a vital part of pre-processing the data may be to encode it in a way that relations are still identifiable to keep the data's usefulness without compromising the anonymity of the individual data source. In an ideal scenario only pre-encoded data would be able to be collected, but often times that is not the case.

The pre-processed data is then stored long-term. Big Data storage systems need to make vast amounts of data accessible while providing reliability. This is only possible through distributed storage systems that spread out the load to multiple servers. Another factor is the type of data. As Bhadani and Jothimani (2016, p.7) explain, there is structured, unstructured and semi-structured data. Structured data has a predefined format that does not change. Data stored in traditional relational databases is part of that category. On the other hand, files like texts, images, audio and video are categorised under unstructured data. The in-between category of semi-structured data is constrained to a general structure that can vary between single instances of data. This means one instance can include data points that are missing from another instance. A good example for this are the various sensors used in Internet of Things applications. While they normally produce data of the same format, the actual data included in each reading may differ from sensor type to sensor type. Depending on the circumstances, the storage level of a Big Data application has to incorporate different storage systems designed around these issues.

The final step of the Big Data Value Chain that directly handles the data is the *analysis*. There are two parts to the analysis of Big Data. The first one is to use a feature set relevant to the use case in combination with the appropriate technologies and algorithms in order to extract new insights from the data. A feature is some kind of data point derived from the data set. Some data points can be used directly, others have to be modified in some way. For example, different algorithms exclusively work with numerical data. A relevant feature of the data set containing textual data needs to be encoded in order to be usable by the algorithm. The other part of the analysis is to visualise the results in some way for the final step of the Big Data Value Chain. Visualisations can be anything from a simple tabular representation of the resulting information to complex graphs and even specially developed features insight end user software systems.

At last, there is the *decision making* phase. In this final step the resulting insights of the Big Data Value Chain are taken into consideration by the parties involved in making decisions. These may be people in charge of projects, departments, companies or even countries but also just the simple user of an internet service. It all depends on the specific situation and use case Big Data is utilised for. Just as with the *data generation* phase, the *decision making* phase is not directly part of the Big Data software infrastructure responsible for extracting the insights out of the data. In an exemplary Big Data application, like Netflix's recommendation system, the parties generating the data and making decisions are one and the same. In general, the insights are used to enhance a product in any way possible. Netflix enhances the user experience through sensible recommendations to its users, other companies may use Big Data to predict the demand for their product in order to produce enough units or to increase margins by cutting costs.

All in all the Big Data Value Chain can be seen as a more abstract alternative to the reference architecture. It provides a more general view on what makes up a Big Data application in the real world. Therefore, the Big Data Value Chain was used in the following section of the thesis to categorise the different available technologies and methods for

Big Data. Using the phases of the value chain as categories simplified the decision making process during the conception of the guide and its accompanying software infrastructure.

2.2 Technologies and Methods

This section introduces some of the most common Big Data technologies and methods that are being used in real world applications. Though some restrictions were made concerning the tools that could be included in the list. For one, only freely available and usable tools are listed. Commercial software would not allow for an easy distribution and use of the guide's accompanying software infrastructure without proper licenses. This decision is based on one of the goals mentioned in the introduction of the thesis.

Second, the listed technologies all reside within the Apache Hadoop ecosystem. Apache Hadoop was initiated in 2011 and is has been widely used in Big Data applications (Chen, Mao, and Liu 2014, p.178). Besides that, it is also supported by a large and active community that not only maintains the project but continuously extends its capabilities with tools that are built on top it (Oussous et al. 2017, p.7).

Third and last, the list is restricted to projects that are part of the Apache Software Foundation like Apache Hadoop itself. While there are technologies being developed by private organisations under the open source banner, it cannot be ruled out that business models change and such software is re-licensed. Depending only on projects that are part of the Apache Software Foundation ensures to a certain degree that the software will stay open source and freely usable for personal and commercial uses.

All technologies and methods listed in this section can be found in at least one of the sources used for this thesis. The main sources for this list are (ordered descending by year of publication) Oussous et al. (2017), Bhadani and Jothimani (2016), Acharjya and Kauser Ahmed (2016), Pääkkönen and Pakkala (2015), Emani, Cullot, and Nicolle (2015) and Chen, Mao, and Liu (2014).

2.2.1 Apache Hadoop

Apache Hadoop is a software framework designed around the challenges of Big Data. Every aspect of it is focused on providing the means to store and process exceptionally large data sets through horizontal scalability. This means that a cluster of computers running Hadoop can be extended with more machines to increase the system's performance. This is cheaper and more feasible for Big Data than relying on improved hardware performance in terms of storage capability and computational power. There are three main modules that allow Hadoop to process large amounts of data across multiple computers.

HDFS or *Highly Distributed File System* is the storage engine of Hadoop. It allows for the data to be distributed across multiple machines and focuses on high throughput. To allow ease of use when interacting with a Hadoop cluster, the data can be accessed as if it was stored on a single drive. Internally though, HDFS distributes parts of a bigger data set to all of the cluster's nodes.

The *MapReduce* module is a programming model designed for parallel execution. Each MapReduce program consists of multiple steps that can be executed simultaneously on all of the machines in the Hadoop Cluster. In the first mapping step, each machine reads the part of the data that is available locally and derives key value pairs from it. In an intermediate shuffling step, the resulting lists of key value pairs are redistributed across the machines so that all pairs with the same key can be found on a single machine. Afterwards, the reduction step will process all the key value pairs with the same key at once to create a single result per key. This results in a list of key value pairs with unique keys which is written to the HDFS for further usage.

The last main module of Hadoop is *YARN* or *Yet another resource negotiator*. It's purpose is to manage the cluster's resources and schedule jobs. Whenever a data is added to the HDFS or a MapReduce program is executed, YARN makes sure that enough resources are available and that they are used properly to their full extent. (Oussous et al. 2017, p.7)

2.2.2 The Hadoop Ecosystem

The Hadoop ecosystem consists of numerous software packages that handle one or more of the Big Data Value Chain steps. Every Big Data application utilises a distinct compilation of tools that work together in forming an implementation of the value chain. The following paragraphs introduce some of the most common and prominent projects and categorise each one in terms of the value chain's steps.

Apache HBase is a non relational database built on top of HDFS that inherits its scalability features and distributed storage of data. The number of rows and columns it can handle is in the millions to billions. This is doable, because in contrast to traditional relational databases HBase is designed as a column store. That means multiple columns are grouped together as families and stored in that way as well, allowing the data to be split among multiple servers and sparse data to be stored efficiently. If a row does not have data for a column group, there simply is no need to create a record for that group with empty fields. The data for each row can be accessed through a key. In traditional relational databases that are row oriented, the data of all columns of a row always stays at the same place and a *null* value is stored for for empty fields. This column focus also allows for more efficient schema changes. In terms of the Big Data Value Chain, HBase is a clear example of the storage phase. (Oussous et al. 2017, p.7)

With HDFS and HBase as distributed storage systems in a Hadoop cluster, there needs to be a way to actually import data. One tool for the job is *Apache Sqoop*. Its purpose is the bulk transfer of data between different types of data stores with support for unstructured, semi-structured and structured data. Looking at the value chain the software is used for the collection and transmission of data. (Oussous et al. 2017, p.9)

Apache Flume and *Apache Chukwa* are projects with a similar goal to Sqoop, but instead of the focus on bulk data, the focus lies on data streams. They also include rudimentary capabilities for the pre-processing of the data. While the main application for both tools is with log data, Flume also allows the user to create extensions that make it possible to work with different types of event based data. The relevant phases of the Big Data Value Chain for both are the collection, transmission and pre-processing of data. The combination of those three phases is sometimes also called data ingestion. (Oussous et al. 2017, p.9-10)

Apache Pig provides a high-level programming language called *Pig Latin* for the creation of simple data processing and analysis programs. These Pig Latin programs are compiled to a series of MapReduce programs by Pig to allow their parallel execution using Hadoop. Pig Latin shares similarities with procedural programming languages and can be extended with custom functions by the user. With the support for different data formats and storage types, Pig is able to be utilised for extract-transform-load workflows. Such workflows extract data out of different sources, transform the data to the specific requirements of the use case and load the transformed data into the target storage system. In terms of the Big Data Value Chain, this means that the pre-processing and analysis phases are covered by Apache Pig. (Bhadani and Jothimani 2016, p.16)

Apache Hive is a project with a very similar goal to Apache Pig, but with a different approach. Instead of providing a procedural high-level programming language like Pig Latin, Hive offers the usage of a SQL-like language called HiveQL to access Big Data in the HDFS and HBase storage systems. Similar to HBase, Hive's data model represents the data as tables that can be queried using HiveQL. Just as with Pig, Hive allows the user to extend its capabilities with so called *User Defined Functions* and provides the necessary tools for extract-transform-load workflows. This also means that the same value chain phases are covered by Apache Hive as are covered by Apache Pig. Choosing between both of them mainly relies on personal preference towards and knowledge of procedural programming respectively SQL like query languages. One disadvantage of Hive in comparison with Pig is that all data needs a table schema projected onto it in order for Hive to be able to query it. In contrast Apache Pig can work with data directly, choosing to use a schema only in cases where it is beneficial. (Oussous et al. 2017, p.9)

The *Apache Storm* project represents an alternative distributed processing system to Hadoop. In contrast to Hadoop, whose strength lies in batch data processing, Storm focuses solely on the processing of streaming data. Another difference to Hadoop's MapReduce jobs is that Storm jobs will run indefinitely until the user stops them. A MapReduce job will terminate after all data has been processed. The software can be used with any programming language and also integrates with a lot of tools in the Hadoop ecosystem, for example to store processing results. Storm is a clear representative of the analysis phase of the Big Data Value Chain. (Acharjya and Kauser Ahmed 2016, p.516)

Another alternative to Hadoop's MapReduce module is *Apache Spark*. It utilises in-memory computation to achieve a vastly improved processing performance compared to Hadoop. Compared with Apache Storm, Spark can be integrated more tightly with an existing Hadoop deployment as it can use Hadoop's YARN module for the management and scheduling of tasks and HDFS for its storage capabilities. Spark can be used with Java, Python, Scala as well as R and also offers SQL-like functionality through its SparkSQL module. One differentiating factor of Spark compared to Hadoop and Storm is its support for both, batch data processing and streaming data processing. The streaming data can originate from a number of systems like Apache Flume. The software also includes various machine learning and graph processing algorithms for advanced data analysis, which is the main value chain phase covered by it besides the pre-processing of data. (Acharjya and Kauser Ahmed 2016, p.515-516)

Apache Mahout is another project aimed at data analysis. It includes a number of already implemented algorithms used for machine learning and data mining. It can be used with other Big Data tools like MapReduce and Spark to extend their analysis capabilities. (Acharjya and Kauser Ahmed 2016, p.515)

2.2.3 Machine Learning

Besides traditional statistical analysis, Big Data allows for machine learning to be utilised to a great extent. Machine learning is a wide category of analytical methods to infer unknown insights from data and in some cases even make decisions based on those insights. (Oussous et al. 2017, p.5). The following paragraphs introduce some of the most common machine learning techniques that can also be found in tools like Apache Spark and Apache Mahout.

Generally speaking, machine learning is divided into three subcategories of learning. First, there is *supervised learning*, which requires training data where the inputs as well as the outputs are known beforehand to train the algorithm. The better an algorithm is trained, the better results can be derived by it from the following provided data. Second, *unsupervised learning* techniques are able to analyse the data without prior training of the algorithm. This allows for the fast and easy analysis of new data sets and types of data. Last, *reinforcement learning* refers to techniques that learn through the feedback of an external entity. A good example for reinforcement learning are artificial intelligence programs for games. Reinforcement learning algorithms take decisions and adapt their decision making process based on the positive or negative feedback received as a result of the decision. (Qiu et al. 2016, p.2)

Common supervised learning methods implemented in Apache Spark and Apache Mahout include classification and regression algorithms. In classification tasks the algorithm is trained to assign one of a finite number of available classifiers to the input data. Each set of input data points is also called a data instance with each unique instance receiving its own classification by the algorithm. (Alpaydin 2014, p.5-9) An example of this could be to classify songs into genres based on a song's properties like beats per minute or most common occurring frequencies.

In contrast to the predefined outcomes of a classification task, regression algorithms work with a numeric output value. Given a set of input parameters, a data instance's output value can be calculated. Training the algorithm will result in a mathematical function that is fitted to the training data. That fitted function is what allows the algorithm to calculate the output of a new data instance. This also means that the result is never completely accurate but more of an estimation based on past information. (Alpaydin 2014, p.9-11) As with classification, the technique can be used to predict based on past events. For example the algorithm could be trained to predict the time it takes to run a marathon. The training data would use genuinely recorded times as the output value and different circumstantial data points like the altitude of the location or the average ambient temperature throughout the run as input parameters. The fitted function is then able to predict a time given a set of input parameters.

The second category of machine learning techniques are unsupervised learning algorithms. The most common technique used in this category is the *clustering* of data. The goal of a clustering algorithm is to group the all instances of an input data set by similarity. (Alpaydin 2014, p.11-13) Staying with the previously used music example, clustering could be used to group songs together that are similar in some way into genres. The difference compared to the previous classification example is that we do not know any of the labels beforehand. The algorithm alone decides which songs belong together based on the set of input parameters. Apache Spark provides different clustering algorithms for processing batch data and also for processing streaming data, where the resulting statistical model is continuously updated as new data is being processed. (*Apache Spark Documentation* n.d.)

The last of the three main categories of machine learning techniques is reinforcement learning. There are actually no general purpose algorithms implemented for it in either Apache Spark or Apache Mahout. The general concept of reinforcement learning is about an algorithm that makes a sequence of decisions based on input data that is changed by each decision. Between decisions the algorithm adapts its decision making process in accordance with some sort of feedback received as a result of the previous decision. After thorough training, the algorithm has created a decision making policy that is optimised for the best outcome. A common example for reinforcement learning can be found in artificial intelligence programs designed to play games. The rules of a game can be easy to understand and represent as a program while the actual playing of the game stays complex. For example, the game of chess has a relatively small number of rules but basically infinite possible outcomes for each game. A reinforcement learning algorithm designed to play chess would adapt its behaviour based on how past moves affected its probability of winning the game. (Alpaydin 2014, p.13-14) The increasing processing power of modern computing in combination with the possibility of such algorithms to play against themselves can create a program that is trained on billions of played games within days. One of the most interesting aspects of this is that such algorithms start off without any understanding of what a good strategy may be. The only way they learn to play the game well is by playing themselves.

There are also machine learning techniques that do not fit properly in only one of those categories. *Collaborative filtering* is a popular example for this case. It is a common technique for recommendation systems and based on a model of a two-dimensional matrix. In case of a movie recommendation system one axis would be the users and the other the movies. Each user and movie is described by a number of hidden features. Given a user and a movie, the algorithm gathers similar users and movies from the overall data set and tries to fill the gaps of missing pieces. If the original user has not rated a movie, its rating can be estimated by the ratings of the other users with similar taste. Based on this estimation, the movie is recommended to the original user. In this case the algorithm works on the basis of explicit feedback in form of movie ratings provided by the users. The more common case is the one of implicit feedback, in which the data inside the matrix has to be derived from other information like clicks, likes and shares. The algorithms provided by Apache Spark can work with both feedback variants and are also able to extract the hidden features out of a data set on their own. This combination makes collaborative filtering part of supervised learning as well as unsupervised learning. (*Apache Spark Documentation* n.d.)

There are numerous more different techniques in the field of machine learning like *dimensionality reduction* and *frequent pattern mining*, but the ones mentioned most often are the ones listed above. All in all, machine learning is a powerful tool to make sense out of large data sets and infer new knowledge that would have not been found with traditional statistical analysis. Another factor is the possibility to combine these techniques. As the *Apache Spark Documentation* (n.d.) mentions, the results of an unsupervised clustering algorithm are often used as input in other supervised learning techniques to form a data analysis pipeline.

2.2.4 Meta Tools

The tools and analytical methods listed above all directly work with the data and are part of the Big Data Value Chain. But there are also other tools that are part of the Big Data software infrastructure, that only concern themselves with the infrastructure itself and not the data. This means managing and deploying other tools, scheduling jobs or offering an easy way to interact with the available tools for handling the data. These are subsequently called *meta tools* as they work a level above the other tools. The following examples are describe a few of these tools and what their purpose is within the overall software infrastructure of a Big Data application.

The most common management software for Big Data is called *Apache Zookeeper* and aims to coordinate the different parts of the infrastructure. It is able to maintain configurations, take care of naming and provide different types of synchronisation. Centralising these parts of the infrastructure allows for easier maintenance as well as scaling of the involved software packages. (Oussous et al. 2017, p.12)

Another part of maintaining a Big Data infrastructure is the deployment of software packages. Doing this by hand is not only tedious but also prone to errors. *Apache Amabari's* goal is to provide automated mechanism for the deployment. First, all of the machines available to the cluster are added to Ambari which starts its agent software on each of them. Afterwards Ambari can be used to configure which tools should run on which machines resulting in Ambari automatically installing and starting them. The configuration can still be changed afterwards. This makes it easy to reconfigure the cluster to different workloads or assign new roles to machines in case the cluster is scaled up or down. (Oussous et al. 2017, p.12)

Depending on the usage of the Big Data infrastructure, certain jobs created in Hadoop MapReduce, Apache Pig, Apache Spark or any of the other tools need to be executed whenever a previous processing step is done, new data is available or even in fixed time based intervals. Scheduling these jobs can be done using *Apache Oozie*. The software is not only integrated with a number of tools in the Hadoop ecosystem but can also work with custom programs and shell scripts. (Oussous et al. 2017, p.12)

The last example for a Big Data meta tool is *Apache Zeppelin*. It offers the user a web interface to interact with the deployed Hadoop tools using so called notebooks. Notebooks consist of paragraphs that can execute small scripts developed for any of the supported software packages. In this way Zeppelin allows for fast and easy Big Data analysis. Notebooks can also be shared to allow for collaboration between different users. Apache Zeppelin was not found in one the sources used by this thesis. Instead one of its competing projects named *Hue* was mentioned in Oussous et al. (2017, p.13) that servers a nearly identical purpose as Zeppelin. Since one of the limitations concerning technologies and methods was to use only software managed by the Apache Software Foundation, Zeppelin was chosen to be included in this list over Hue.

These meta tools are not required for a working Big Data application, but they can simplify the maintenance of the software infrastructure and offer an easy to use way of working with the available data.

2.3 Communication of Information

The last section of this chapter talks about the different ways information can be communicated. The focus was set to the most effective form of communication for the recipient to comprehend the information. This thesis focused on two main forms of communication: written explanations and ones in audio-visual form. Predominately aural explanations were excluded as the guide may include snippets of code or commands that have to be issued to a shell. In that case exact syntactical reproduction of the code or commands would be necessary and audio-only content would likely produce errors.

As Käfer, Kulesz, and Wagner (2016) write, only few studies comparing written explanations of software tools with ones in a video format have been done. The ones available offer mostly inconclusive results regarding which form of communication is preferred by recipients and which one yields the best results when it comes to actually understanding the content.

In their own study, Käfer, Kulesz, and Wagner (2016) come to a similar conclusion based on their results. They acknowledge that most people prefer videos in the beginning, but also that a written text allows the reader itself better control over the speed of the communication. All in all they conclude that the content of the explanation is a lot more important than the form of its communication when it comes to effectively understanding the conveyed information.

Another study of self-learning tools, like the “Practical Guide to Big Data” aims to be, concludes that even the best content is dependent on how it is utilised by the person consuming it. Each person differs in their “motivation, discipline, self-regulation and time-management skills”. All of stated factors can interfere with the effectiveness of a self-learning tool. (DeVore, Marshman, and Singh 2017) Their conclusion is very similar to the one by Käfer, Kulesz, and Wagner (2016). Both acknowledge that the most important factor in the effectiveness of communication is the recipient followed by the actual information that is conveyed. The form of communication only plays a subordinate role.

This means there is no best way to communicate information in order to effectively explain a software tool or method to the person consuming the information. The only best way would be to provide all of the different forms of communication for the consumer to chose from, which is simply not feasible. Therefore the decision on which form of communication to use should be based on other factors influencing the creation and distribution of the guide that will be discussed in more detail in the chapter regarding the conception and implementation of the guide.

3 Evaluation Criteria

The last step before the actual conception and implementation of the guide was the creation of evaluation criteria. Being able to evaluate the resulting guide was not only important to check if the defined goals of the thesis have been achieved, but the evaluation criteria itself were helpful during the conception of the guide to steer it towards the right direction.

In the introduction of this thesis, the goals towards the guide have already been defined. These goals served as a foundation for the evaluation criteria of the guide. Creating the evaluation criteria based on these goals ensured that they were not only relevant, but also that the positive fulfilment of the evaluation criteria inherently indicates that the thesis' goals have been achieved successfully.

The following list of criteria uses the terms *have to* and *should*. The former is used in cases were full achievement of the respective criterion was mandatory to also ensure achievement of the goals. In the case of *should*, the criterion was to be fulfilled to the maximum degree possible with reasonable effort.

1. All technologies used and mentioned in the guide should be relevant in real world applications of Big Data.

   This criterion ensures that the guide conveys information that can be directly applied and reused in other circumstances.

2. All technologies used and mentioned in the guide have to be free to use in personal and commercial applications.

   This criterion ensures that the guide itself is allowed to use the technologies in question and minimises potential entry barriers to introducing the information conveyed by the guide in other circumstances.

3. All technologies used in the guide should be provided pre-installed and pre-configured so that they can be used productively without any additional effort by the reader.

   This criterion ensures that the guide focuses on the application of the technologies in question. The need to install and configure the technologies would distract from the guide's purpose to be a practically oriented introduction into the field of Big Data.

4. All data sets used in the guide have to be free to use in personal and commercial applications and be accessible in a reliable manner.

   This criterion ensures that the guide is allowed to use the data sets in question and that the guide is usable for the foreseeable future without problems.

5. The medium used for the guide should offer the most effective way of communication information while allowing it to be distributed easily and updated in an efficient way should the need to do so arise.

   This criterion ensures that the guide can be accessed without any unnecessary restrictions, that the information contained in it stays correct and that the knowledge taught by is being received clearly and comprehensibly.

All in all the fulfilment of the evaluation criteria secured the guide against any problems concerning the licensing of software and that its information was accessible to as many people as possible without any unnecessary restrictions. The criteria also made sure that any skills acquired by working through the guide could be reused without limitations in other projects as the tools involved have no personal or commercial usage restrictions.

4 Conception and Implementation

The conception and implementation of the guide took place in six separate sequential steps. Each step built upon the previous step, so that in the end the complete guide was created. The process started with choosing the overall structure. From there, constraints were defined that the guide had to abide to in order to work. Based on the constraints, the specific software packages were chosen that were needed to implement the guide and that should be introduced to the reader. Then, the practical examples were created that the guide was based around. The final step for the conception was to define the content of the guide in detail to create a coherent text that moves from example to example and includes all the necessary information for the understanding of Big Data and the examples. Finally the guide was implemented based on the concept. The following sections of this chapter explain each of the steps in more detail.

4.1 Structure

The most basic decision to be made regarding the structure of the guide was the choice of medium. As chapter two explained, multiple studies comparing written and video content came to no conclusive result which medium is better suited to effectively convey information. According to the studies, it depends mostly on the person consuming the information and the quality of the content itself. Therefore, other factors were taken into account for the decision.

First, the effort involved in creating a written guide and one in video format was estimated. Video production is not only more complex than writing, it is also more costly too achieve a certain degree of quality. Specialised equipment like cameras, microphones and editing software is needed to produce video content, not to mention the additional time it takes to shoot and cut the footage. Additionally, creating the guide in written form allows easier updates and extensions of the content in the future.

The second factor reflected a similar image as the first one. It was concerned with the distribution of the guide after its creation. While platforms like YouTube have made it easier in recent years to make video content accessible from anywhere, written content is still more versatile and easier to distribute.

Another factor was the reader's better ability to control the speed of consumption with a written guide. Especially in the case of complex topics, being able to slow down or repeat reading specific passages is a valuable ability. Video content depends on the playback mechanism to allow adjusting the speed and even if the feature is included, slowed down video produces distorted audio.

Additionally, written content in a digital format allows the reader to copy-and-paste passages of text for exact syntactical reproduction. This is especially helpful with passages of code and shell commands that depend on a correct syntax. At the time of the decision, it was clear that the examples would need to include such passages.

In the end, the clear choice was made to produce the guide in written form. While video is an interesting medium, and the referenced studies even showed that most people prefer it in the beginning, the combination of all relevant factors heavily favoured written content.

Subsequently, the way the guide was to be distributed was made. This decision also included the specific text format, the guide would be written in. There were a number of factors influencing the choice that all pointed towards distribution as a website, which can be centrally hosted on the internet and accessed from anywhere as long as the reader has also access to an internet connection. Additionally, the centralised hosting of websites make it that any future update or extension of the text would be directly accessible by the reader. Versatility regarding the consumption is another advantage of websites. They can be read on various types of devices and modern browsers also provide the ability to save any website as a PDF file or print it out on paper. In consequence, the reader retains more freedom of choice in how the guide is consumed.

The last decision to be made that was concerned with the overall structure of the guide concerned itself with the way the guide's accompanying Big Data software infrastructure would be provided. The infrastructure includes all the tools required in order to work through the examples in the guide. Similarly to the guide itself, the infrastructure needed to be distributed in some way.

One of the predefined goals and evaluation criteria established that the reader should have to do as little as possible to setup the software infrastructure. The best way to achieve this is to provide it already pre-installed and pre-configured. The only way of doing that is to provide all necessary software inside a virtual machine. A virtual machine is a simulated and/or emulated computer system that is running on a host computer system. All software required by the guide as well as the data required by the examples is provided from within the virtual machine.

Managing and running such a virtual machine is done by specialised software and a lot easier than managing and running each Big Data software package by itself. Additionally, removing the infrastructure is simplified as well. Removing the virtual machine and uninstalling the virtual machine host software is all that is needed to clean up after completion of the guide.

4.2 Constraints

Based on the structure of the data and the information gathered from surveying Big Data applications and tools, it was clear that certain limitations existed regarding the reproduction of a realistic Big Data scenario.

There are two different types of data that are being utilised in Big Data applications: batch data and stream data. Batch data refers to a finite set of data instances that is being processed and analyses, while steam data refers to data that is continuously generated and imported into the Big Data application for continuous processing and analysis. This means that batch data can be provided together with the infrastructure itself while stream data depends on a working data source. Such a data source would need to produce sensible data that also reflects the real world in some way which excludes to possibility to simulate the data source in the virtual machine.

Therefore, any example using stream data would have dependent on an external data source that is not controllable. Any change to the data source, for example to the URL under which it can be reached or to the format of the returned data would have resulted in the need to update the guide. Without an update the relevant example would be rendered useless. Besides that, external data sources also have the constant possibility of being shut down. In that case not even an update to the guide could save the example. In consequence, the decision was made to completely exclude any external dependency from the guide which also meant that only batch data processing was to be included in the guide.

Batch data does not have the problem that stream data has. The finite set of data can be provided together with the Big Data software infrastructure as long as the license under which the data was published allows for redistribution. This circumvents all of the problems with stream data as the data is accessible as long as the guide itself is accessible and the format of the data cannot change in any way.

Another limitation regarded the scope of the guide and was derived from the technical limitations of running a virtual machine for the software infrastructure. Each included software package would have increased the required resources that needed to be allocated by the host system in order for the virtual machine to function without problems. At some point, the system requirements would have been so high that most personal computer systems would have problems with running the virtual machine. In that case, the guide would be useless for a large percentage of interested readers.

Based on that, the decision was made to only include the most relevant phases of the Big Data Value Chain in the guide. As shown in chapter two, each introduced software package is designed around one or more of the value chain phases. Of all the phases, the generation of data and the making of decision are excluded because they are not directly involved with the software infrastructure of a Big Data application. In contrast the storage phase is existential. Without a storage system, no data could be stored that is to be processed and analyses. The remaining phases are the collection and transmission of data and the pre-processing and analysis of data.

In the end the pre-processing and analysis phases were chosen to be represented in the guide over the collection and transmission of data. They constitute the phases that produce the most tangible results for the reader. The tools for the other phases basically only move data from one system to the next. Another factor that influenced the choice was again about the dependency on external data sources as the collected data would need to originate somewhere. Avoiding external was deemed to be vital to ensure the guide's longevity.

4.3 Apache Hadoop Ecosystem

The Apache Hadoop ecosystem is the foundation for most real world Big Data implementations. It was already clear that Hadoop was to be used for the guide during the survey of available technologies and methods, which is why software outside of the ecosystem was not considered or researched. Consequently different tools built on top of Apache Hadoop were chosen that represent the storage, pre-processing and analysis phases of the Big Data Value Chain.

Additionally Apache Zeppelin was chosen to be used as the central access to the infrastructure which the reader uses to interact with the different available tools. Zeppelin allows its user to execute different types of scripts that interact with the tools and displays the results. It supports nearly all tools available in the Hadoop ecosystem. Furthermore, scripts and results are stored permanently and the software provides a selection of basic built-in visualisations for the results.

The storage system used for the guide was kept simple. HDFS is part of every Apache Hadoop deployment and can store any type of data in files. This allowed for easy usage with the provided data sets as batch data is generally stored in files or databases, which in most cases can also be represented in the form of structured data files. Another advantage of HDFS is that nearly all of the tools listed in chapter two can directly work with HDFS to load and store data.

The first pre-processing and analysis tool to be introduced was chosen to be MapReduce itself. Just as with HDFS, MapReduce in included with every Hadoop deployment. The module and its identically named programming model build the foundation for the highly distributed parallel execution of analysis tasks that Big Data and Hadoop are known for. Therefore understanding MapReduce itself was deemed to be important to understand what happens in background when more high-level tools are used to analyse the data.

Apache Pig was chosen to be the first tool that provides a level of abstraction from MapReduce through its high-level programming language Pig Latin. Subsequently, Pig Latin scripts can be compared in a good way to the Java program code of MapReduce programs, as both represent a form of programming. Pig's closest competitor was Apache Hive. Instead of a programming language, Hive provides a SQL-like query language called HiveQL. Pig's main advantage in the decision was its ability to directly work with files without requiring a data model or schema to be projected onto the data. In contrast Hive only works with a table schema that is projected onto the data before the data is read. This makes working with Pig easier which is beneficial for an introductory guide.

The other tool chosen to be used for the processing and analysis of data was Apache Spark. It can work with batch data as well as streaming data and provides a programming interface to the Python, Java, Scala and R programming languages as well as to its own SparkSQL module for querying data. Spark additionally includes various machine learning and graph processing algorithms for advanced analysis that goes beyond what Apache Pig and Apache Hive can do.

In terms of the programming language to use with Apache Spark, the decision was made to use Python. Python it not only tightly integrated with Spark but also with Apache Zeppelin. The latter allows Python to be used to create custom visualisations through its *Matplotlib* module. Matplotlib can create plots of any type and form. As Zeppelin's built in visualisations are rather basic, introducing the possibility for custom visualisations makes a powerful tool for data analysis available to the reader.

4.4 Examples and Data Sets

To achieve the overall goal for a practically oriented introduction to Big Data, it was decided that the guide should focus on examples showcasing more and more complex data analysis using a different tools available in the Apache Hadoop ecosystem. The examples should show the reader how to use the tools and be supplied with data sets that are pre-loaded into the virtual machine during provisioning.

4.4.1 MapReduce

Based on the technologies selected for the guide, the first example chosen has the intention of explaining the MapReduce programming model. It is based upon the example included in the Apache Hadoop documentation. The official example is a basic MapReduce program that calculates the number of times each distinct word occurs in a given text. The problem with the example was that the text being analysed is quite short. Therefore the book *Alice's Adventures in Wonderland* by Lewis Carroll was chosen to be analysed. The book is old enough that it is part of the public domain. This means the text can be redistributed alongside the guide without any problems concerning licensing.

The problem with counting the words in the book was that there were too many. The MapReduce program resulted in a list of hundreds of word counts. Therefore, the decision was made to slightly alter the official example. Instead of counting the words directly, the example would count how often the length of each word occurred in the text. This reduced the result set drastically to a number that could be displayed without any problems in Apache Zeppelin.

Another decision was made to provide the MapReduce program in pre-compiled form. The user's only task was to execute the program. While an explanation of the program is important to understand MapReduce, its compilation is mainly tedious. Additionally, the creation of custom MapReduce programs is not prevalent with the advent of tools like Pig and Spark that abstract the process to a certain degree.

4.4.2 Apache Pig

The purpose that was defined for the second example was to bring out the differences in working with MapReduce directly and using an abstraction like Apache Pig. Therefore, the previous example of counting the lengths of words occurring in a text was chosen to be recreated using a simple Pig Latin script. The idea was to highlight the advantages of Pig and how it differs from the Java code of the MapReduce program.

Additionally the built-in visualisations of Apache Zeppelin were to be introduced. Instead of only displaying the results in a tabular form, in this case the results should be displayed as a simple bar graph with the year as the x-axis and the number of accidents as the y-axis. This also meant introducing the way the desired visualisation is picked and configured inside Apache Zeppelin's user interface.

The last part of the Apache Pig example was designed to save the results as a CSV file and store that file in HDFS. The ability to not only load data but also store it permanently after processing and analysing allows for more complex analysis tasks as multiple tools and algorithms can be combined.

4.4.3 Apache Spark

The next software chosen to be introduced in the guide was Apache Spark. As Spark in itself is a lot more powerful than the other tools, because of the included module for machine learning tasks, it was decided to use the software throughout the rest of the guide and explain its usage with more and more complex examples. The guide should also mention that Apache Pig and Apache Spark may seem to work in a similar fashion, but that they are designed for different circumstances. As Spark processing is done in memory it is a lot faster, but very large data sets can cause memory allocation problems. A disadvantage that does not exist with Apache Pig as it uses the standard Hadoop MapReduce paradigm in the background which does not load the complete data into memory.

To be able to create the different Apache Spark examples and to reflect a more realistic scenario than the analysis of a single text file, a new data set needed to be selected. The main potential sources searched for appropriate data sets were governmental and non-profit organisations as the data had to comply with a number of requirements and in recent years such entities have started to publish more and more publicly available data. First, the selected data set had to consist of raw data. This means data that is not pre-processed or pre-analysed in any way. Data that is already analysed inadvertently may have lost some of its hidden insights that machine learning is used for to uncover. Second, the data had to consist of a large amount of instances to reflect a real world scenario at least partially. A data set with only a few hundred entries is not ideal when using machine learning algorithms. The more instances are available, the better the result. And finally, the data had to be freely usable and redistributable alongside the guide.

In the end, a data set published by the UK government under the name *Road Safety Data* (n.d.) was chosen that consists of traffic accident data for accidents where one ore more people were injured. As a lot of accidents without human injuries are not reported to the police, the decision was made by the UK government to not publish the data for such accidents to keep the quality and reliability of the data set high. The content of each data instance consists of the information the police record when a traffic accident is reported to them. In that case they have to fill out a STATS19 form. The data set exhibits a number of positive aspects regarding the usage in data, which is why it was chosen for the guide. With over a million recorded traffic accidents in the time from 2009 to 2016, there are enough instances to successfully apply machine learning techniques. Additionally, the data for each traffic accident has a variety regarding the types of information. For example there are data points for locations, weather and road conditions as well as general information regarding the accident.

The following examples were created by exploring the data set using different combinations of data points to find interesting statistics of different complexity. The first and simplest one was to simply count the accidents by their year of occurrence and display the result in the built-in line graph visualisation of Apache Zeppelin. This example is similar to the previous ones and can showcase how Spark is used to the reader without being to complicated.

The second example for Apache Spark had to increase the complexity of the resulting data set. In this case another count was chosen, but instead of only providing a single dimension in the form of the year, this time the accidents were grouped by the time of day and the day of the week of their occurrence. Again, the line graph was chosen to visualise the results. For this example, an additional passage needed to be included that explained how to visualise more than one dimension with Apache Zeppelin. In the case of the line graph, the problem is solved by drawing a line for each weekday and distinguishing the day of the week using distinct colours.

Another important aspect of Big Data and data analysis in general that was to be introduced using this example was the concept of context. As is predictable, most accidents occurred from Monday to Friday during the rush hour periods. According to the results, weekends generally seemed less dangerous, with the exception of the nighttime. The idea was that the guide would explain that these absolute numbers can be misleading without knowing the amount of traffic that occurs during the week and on weekends. If weekdays had ten times the amount traffic than weekends, but only two times the number of accidents, rush hour may not be as dangerous as initially assumed.

The final Big Data example of the guide was designed to introduce a machine learning algorithm for data analysis. While these algorithms can find patterns and similarities in any combination of data points, a basic clustering based on the latitude and longitude of each accident was preferred. Using such a basic example was meant to simplify the understanding of what a clustering algorithm does as the subject of machine learning may be completely unknown to the reader.

In addition, the example was to be supported by a custom visualisation that plotted the cluster centres onto a map of the United Kingdom. Instead of only displaying the latitude and longitude positions of the cluster centres in a table, drawing them onto a map further helps the reader in understanding the algorithm intend. Furthermore introducing the possibility to create custom visualisation opens up new possibilities for the reader if he or she decides to experiment with the tools and data beyond the examples provided by the guide.

4.5 Content

The next step in creating the concept for the guide was to specify the content in more detail. While the technologies and examples were already clear at that point in time, specifying the content meant to tie everything together into a coherent text.

Based on the fact that the term Big Data is used throughout the research field with slightly different interpretations and that the guide was intended to be introductory to the field, it seemed sensible that the guide should start with a general definition of Big Data. Consequently the same definition by De Mauro, Greco, and Grimaldi (2016) was chosen to be conveyed that was already used in the thesis itself.

Afterwards, an important part of the guide was to provide an explanation of how to setup and use the accompanying virtual machine and how to interact with the included tools using Apache Zeppelin's web interface. The choice of tools made the size of this section as small as possible. After installing Vagrant downloading the repository of necessary files, only the single *vagrant up* command was needed to start the automatic setup of the virtual machine. Additionally, the explanation of Apache Zeppelin would suffice for the reader to be able to work through all of the examples.

The first example concerned itself with the Hadoop's MapReduce module and identically named programming model. In consequence of that, this part of the guide needed to start with an explanation of Apache Hadoop and all of its different parts. Since YARN was not chosen to be worked with directly in one of the examples, the focus was put on HDFS and MapReduce itself. The guide should start with a short general description of Hadoop, followed by a more in-depth explanation of what HDFS is and how it can be used. This was especially important as HDFS was to be used as the storage system for every single example.

The actual content of the HDFS explanation was split into two parts. The first one would explain the general concept and what differentiates HDFS from traditional file systems and the second part would focus on a few of the most useful commands to handle data and files with HDFS. Not all of the commands may be used throughout the rest of the guide, but their knowledge allow the reader to ingest his or her own data into the system for further experimentation. To demonstrate the commands, the actual data used in the following MapReduce example was to be loaded into HDFS.

Since it was previously decided that the MapReduce program was to provided along-side the data for easier usage, the section concerning MapReduce had to focus on what the MapReduce programming model is and how it allows for the analysis of such large amounts of data using multiple computers in parallel. Besides that, the actual code for the mapping and reduction functions was to be explained step by step. One benefit of providing the program pre-compiled was that the code surrounding the two functions could be hidden. It was mainly boilerplate code that would only distract from the important aspect of the example, which was how the mapping and reduction functions work in detail to implement the theoretical programming model of MapReduce. The last step of the example was to execute the program and display the results in Apache Zeppelin's interface.

The overall content for the examples of Apache Pig and Apache Spark was designed in a straightforward way. First the respective software package was to be introduced with a few sentences regarding its common usage and included features. Afterwards, the specific examples were to be explained step by step. Additionally the tight integration of the tools with Apache Zeppelin's visualisation system needed some clarifications as some visualisations had the need to be configured before displaying the desired result.

Finally a conclusion for the guide was designed. Its purpose was defined to give the reader various ideas of what to do next in order to gain knowledge in Big Data. He or she could read up on the introduced tools and experiment with the provided software infrastructure and data, and additional reading material of subsequent topics should be taken into consideration. Big Data is a wide field and due to previously described limitations, only a portion of it was able to be included in the guide. It should also be mentioned that a good way to utilise the software available to the reader would be to ingest new data into it. For this, different data ingestion tools should be listed that serve different purposes depending on the specific data. Subsequently, a short explanation of the differences between batch data and streaming data as well as structured, semi-structured and unstructured data was to be given to conclude the guide.

4.6 Implementation

The detailed concept of was then implemented, starting off with the realisation of the virtual machine including the required Big Data software infrastructure and data sets. That was followed by the creation of the text as well as additional image assets showcasing parts of the user interface and expected results for better clarification. Finally, the finished product was thoroughly tested in order to ensure that every aspect of the guide worked as intended.

The first step was to download Vagrant and select an appropriate box. Using the system requirements and supported operating systems listed in the documentations of all required software packages, a basic installation of 64-bit Ubuntu 14.04 was chosen to be the foundation of the virtual machine. In the end, the virtual machine was given 4 gigabytes of memory and a local IP address of *192.168.33.10* that was to be used to access the virtual machine from the host. Additionally a shell script was added that was designed execute the commands required to provision the system.

The first software package to be added to the provisioning was Apache Hadoop as it builds the foundation for the Big Data software infrastructure. A number of additional steps were needed in order for Hadoop to work properly, but these steps could be reused for the other software packages as well. Hadoop needed a Java Development Kit 8 installation as well as a few customised configuration files to be available on the system. The latter were stored in a sub-folder containing all required configuration files together with the Vagrantfile and provisioning script. An additional provisioning step was added to the Vagrantfile whose purpose was to load the entire folder of configuration files into the virtual machine. From there, the provisioning script copied the files to the correct locations for Hadoop to utilise them. The final step for Hadoop and all the other tools was to its various background services.

After everything was configured and scripted, the virtual machine was recreated and Hadoop's proper functionality was verified. Vagrant allows to access any of its hosted virtual machines to be accessed via SSH, which helped in verifying that Hadoop worked properly but was not particularly user friendly. Consequently, Apache Zeppelin was the next software to be installed as it would allow to interact with Hadoop and the other tools from a web interface that would be a lot more comfortable than the command line.

The basic installation steps for Apache Zeppelin were very similar to the ones of Hadoop, but without the need for additional dependencies to be installed as well. The first step was always to download the software package, decompress it and move it to its proper location. This was followed by overwriting the relevant configuration files and starting the software's various background services. After the installation process for Apache Zeppelin was established, the virtual machine was recreated again to ensure that everything would work out of the box for the reader. With Zeppelin running, its user interface could be accessed via the configured IP address and port 8080. Any further verification of installed software packages was done using Apache Zeppelin.

An additional benefit of Apache Zeppelin was that it came pre-packaged with installations for Apache Pig as well as Apache Spark. This meant that no additional provisioning steps were needed for those two software packages. To ensure their proper function, first, an additional file provisioning step was added to the Vagrantfile that was instructed to load the exemplary data sets into the virtual machine. Afterwards, the virtual machine was recreated again and Apache Zeppelin was used to load the data onto HDFS and access it via Apache Pig and Apache Spark.

In order for Python and its Matplotlib module to work properly, a number of additional dependencies needed to installed on the system. After its installation Apache Zeppelin contains a number of different example notebooks with scripts, including one dedicated to working with Python and Matplotlib that only worked after all of the required dependencies and Matplotlib itself were installed properly. The most complicated part of this was the installation of Matplotlib's *Basemap* extension. It is used to create the UK map visualisation of the last Apache Spark example and required a number of additional dependencies that had to be manually compiled from source as no binary packages were available.

After all of the necessary steps were added to the provisioning of the virtual machine, the actual text of the guide was written. During the writing process, a freshly provisioned virtual machine was used to verify that the examples worked properly. Additionally, multiple image assets were created and included in the guide so that readers could compare their results with the expected ones. Though, in case of the last example, the resulting map of the UK is distinct for each time the algorithm is run. This is a consequence of the utilised *k-means clustering* algorithm, which randomises the starting positions of the cluster centres.

4.6.1 Problems

A number of problems emerged during the implementation phase that complicated the creation of the guide in one or more ways. It was not helpful that solving a problem meant to let Vagrant recreate the virtual machine from scratch to test if the provisioning worked correctly. The more software was installed during the provisioning, the longer it took for the process to finish. In the end, the process took up to fifteen minutes from start to finish, slowing down the implementation significantly. Additionally, as some of the required software can only be downloaded as source code and has to be compiled during the provisioning, that time can increase further if the virtual machine is created on less powerful hardware.

Another problem that occurred as a result of constantly re-downloading the software was that some servers would start to throttle the connection and slow down the process even further. With around three to four gigabytes of data being downloaded each time the virtual machine was recreated, a great deal of network traffic was caused by the provisioning. The throttling could circumvented by switching servers from time to time, but this also meant that the provisioning needed to be constantly monitored in order to detect a throttled connection.

The virtual machine itself was running into problems as well once Apache Spark was tested out. While all of the tools that work directly with Apache Hadoop and MapReduce are not limited by the amount of memory available, as the data is processed directly from disk, Apache Spark's approach of in-memory processing caused the virtual machine's memory to fill up from time to time and refuse the execution of scripts. In the end, the memory allocated to the virtual machine by the host system was increased from two gigabytes to four gigabytes. This meant higher system requirements for the reader's computer, but the problem could not be solved in a different way. These requirements as well as a hint on the large amount of traffic caused by initialising the virtual machine were added to the introduction of the guide.

The combination of different software systems running on a single virtual machine also caused additional problems for the configuration of the software systems. At one point, distinct subsystems tried to allocate the same network port for communication which caused one of the subsystems to fail. While running multiple virtual machines would have simplified the realisation of the software infrastructure, it would also have increased the system requirements for the host machine even further. In fact, it could feasibly have increased the configuration complexity even more as some of the Big Data tools use SSH to access different subsystems which would have required the SSH keys to be distributed across all of the virtual machines.

A few other minor issues emerged during the testing of the virtual machine. For one, the example counting the word lengths of a text resulted in different counts depending on if the MapReduce program or the Pig Latin script was used. This was caused by the varying string tokenizer implementations included with Java and Apache Pig that split each line of the text into words that are separated by whitespace. As the overall result only changed slightly, the issue was not solved as it would have involved creating an identical custom implementation for both systems. Another minor issue appeared once the virtual machine was restarted. Some of the Big Data tools would not restart properly and stop functioning, even though they were initially started as self-managing background services. This issue was solved by adding a warning to the guide to not shut down the virtual machine, but to suspend it instead.

All in all, the constraint to only include a subset of the steps involved in the Big Data Value Chain was a very good choice. Any more required software packages would have increased the effort to implement the virtual machine exponentially as the provisioning would have taken even longer, more configuration issues could have emerged and the allocated resources for the virtual machine would have been exhausted even quicker. With all of the listed issues saved, the virtual machine was functioning properly and was able to accurately reproduce the examples included in the guide.

5 Evaluation

The evaluation of the guide was done using the previously defined evaluation criteria of chapter three. Each criterion was examined in detail in its fulfilment proven through specific examples of consideration that were taken into account during the conception and implementation of the guide.

All technologies used and mentioned in the guide should be relevant in real world applications of Big Data.

Generally speaking all of the technologies used and mentioned in the guide were extracted from scientific publications. Many of which based their findings on the analysis of actual Big Data applications. The guide uses Apache Hadoop as its foundation and Apache Pig as well as Apache Spark on top of that for more in depth processing and analysis tasks. The survey of technologies has shown that these are tools that are commonly used various Big Data implementations. Additionally, the guide mentions Apache Flume, Apache Sqoop and Apache Chukwa for the ingestion of data, which where also discovered in the survey of scientific publications. In the case of Apache Pig, the initial developer of the tool was Yahoo before the project ownership and management moved to the Apache Software Foundations.

The only software package included without direct sources was Apache Zeppelin. In this case, other tools with the same goal were discovered in the papers, but Apache Zeppelin was favoured because it is part of the Apache Software Foundation, too, while competing tools like Hue are being developed by private organisations. Matplotlib was included because of its tight integration with Apache Zeppelin. Most Big Data software concerned with visualisation is commercial software and was therefore discarded for this guide. Furthermore, most of the other use cases for visualisations were found to be in situations where custom programs were written in order to integrate the results with an existing software solution. Therefore, Matplotlib and its possibility for customisation's appeared to be a sensible choice.

All in all, this evaluation criterion was fulfilled as much as was possible within the constraints of the guide.

All technologies used and mentioned in the guide have to be free to use in personal and commercial applications.

This evaluation criterion was directly implemented in the survey of viable technologies itself. Since the survey was restricted to only include Big Data technologies that are part of the Apache Software Foundation, only freely available tools that can be used for personal and commercial usages were considered. Additionally, all software projects owned and managed by the Apache Software Foundation are open source. This means that the can be maintained, updated and extended by any interested person or organisation.

The additional software tools used and mentioned in the guide included Python, Matplotlib, Baseman as well as Vagrant. Each and everyone of these tools is open source and personally as well as commercially usable. This meant that the evaluation criterion was successfully achieved.

All technologies used in the guide should be provided pre-installed and pre-configured so that they can be used productively without any additional effort by the reader.

This evaluation criterion was achieved to the fullest extend, with the exception of the virtual machine host software itself. The only software package that needs to be installed in order to work through the guide is Vagrant. All the tools included in the virtual machine are ready to be used once the virtual machine has been successfully initialised and provisioned.

All data sets used in the guide have to be free to use in personal and commercial applications and be accessible in a reliable manner.

There are two different sets of data used within the guide. The text to Lewis Carroll's fantasy novel *Alice's Adventures in Wonderland* and the *Road Safety Data* published by the Department for Transport of the British government. The book was published well over a hundred years ago and has since entered the public domain. This means it is freely usable as well as distributable without any restrictions. In case of the Road Safety Data, the Department for Transport licensed the data set under the *Open Government License* (n.d.). Works published under this license can be used personally and non-commercially as well as distributed as long as the source is acknowledged and the license is included. Consequently, this evaluation criterion has been successfully achieved to its fullest extent.

The medium used for the guide should offer the most effective way of communication information while allowing it to be distributed easily and updated in an efficient way should the need to do so arise.

The fifth and last evaluation criterion was designed to force the guide to use the best possible medium for the communication of information as well as the maintenance of the guide. As described in chapter two of this thesis, all of the studies relevant to this have not found a conclusive result regarding which medium is more effective in conveying information. But, the choice to use a written guide in the form of a website promised easier distribution and maintenance. Therefore, this evaluation criteria has been successfully fulfilled as well.

The complete evaluation of the guide revealed that all of the evaluation criteria had been successfully achieved. Only a small number of compromises need to be made that did not affect the quality of the guide in any meaningful way. As the evaluation criteria were designed around the goals for this thesis, the successful achievement of the criteria meant the successful achievement of the thesis' goals as well.

6 Conclusion

The overall premise of the thesis was the lack of practically oriented information regarding Big Data. This premise was supported by the findings of chapter two, in which scientific publications were surveyed for information regarding the application as well as implementation of Big Data in real world circumstances. The findings have show that most of the research into Big Data has focused on the theoretical and technical aspects.

The creation of the guide has shown that it is possible to include practical aspects into the research and even to make the practical application of Big Data the main focus. The available software tools and algorithms make it possible to create a Big Data infrastructure without any licensing costs.

The result of the thesis is not only the guide but also its accompanying software infrastructure in the form of a virtual machine consisting of various Big Data tools to experiment with. The virtual machine is especially valuable for interested individuals, as it not only allows to test out different technologies but is also extensible as the whole installation and configuration process is outsourced in plain text files that are easily editable.

But, the creation of the guide has also shown the constraints and limitations of such an infrastructure. While the virtual machine is useful for the introduction into the field, more complex processing and analysis tasks will most certainly result in the included software packages reaching the virtual machines limitations in terms of processing power and available memory.

It would be favourable to initiate additional research into the creation of a more robust software infrastructure that may have the possibility to run on multiple computers in order to achieve a certain degree of scalability needed for Big Data to become truly useful and reflective of the real world. Such a software infrastructure could be valuable as a foundation for further practically oriented research into the field of Big Data. Having a useful and easy to deploy software infrastructure for Big Data research would eliminate one of the most time consuming aspects of focusing on the application side of Big Data. As shown by the various problems and issues that emerged during the implementation of the guide, most of the time was spent putting the required software tools into a functioning state.

The resulting guide of this thesis is a valuable resource for interested individuals. With rising applications of Big Data in more and more industries, the demand for educated talent will increase as well. Consequently, it is important to continue creating educational tools for people to start off in the field of Big Data and gain more in-depth knowledge of the different aspects of it.

Appendix A

A Practical Guide to Big Data

This guide is designed to introduce the reader into the field of Big Data by experimenting with different technologies. It provides all the necessary tools in form of a virtual machine. With the Big Data software infrastructure already in place, the guide is able to focus solely on using the tools to extract new insights from a set of raw data.

Prerequisites

The following software and hardware is needed to work through the practical examples of the guide:

- Vagrant 2.0.0+ (`https://www.vagrantup.com/`) to manage and automatically provision the virtual machine.
- 4 GB of free memory for the virtual machine to work.
- A modern browser.

Be aware that the guide will consume more than 3 GB of internet traffic. Besides the folder of the guide itself, that provides the data set, the virtual machine will also install a number of different software tools upon starting it for the first time.

Introduction

Big Data has become a driving force of change in nearly every sector of the economy and government imaginable. But while the technologies and methods become better and more widely used, there is a rising demand for talent in the field as well. Interested individuals are always faced with a harsh learning curve in order to get started with Big Data. Be it that the available information is mainly technical or theoretical, or that it takes a considerate amount of time to even get a basic infrastructure working.

This guide was designed to circumvent those problems. It focuses on applying different freely available technologies to a set of example data. In order to achieve that focus, the software infrastrucure necessary is provided via a virtual machine. The reader just needs to start the VM and can begin experimenting with the data through a user-friendly web interface. All the tools used in the guide are open source and freely available to make Big Data as accessible as possible to the reader.

After working through the guide, the reader will be able to utilize the provided software infrastructure on his or her own sets of data. The introduced technologies and methods allow for a basic exploratory study of data sets that result in the creation of new analytical insights into the data as well as the creation of supporting visualizations.

In the end, this guide tries to serve as a basic starting point for interested individuals. With the knowledge gained from working thorugh it, the reader should be able to further acquire more in-depth knowledge of Big Data and its different parts and apply it in different circumstances.

What is Big Data?

Before getting started, this section will briefly talk about what the term Big Data actually means. The guide is based on the following definition of Big Data by De Mauro, Greco and Grimaldi from their 2016 paper *A formal definition of Big Data based on its essential features*:

> Big Data is the Information asset characterized by such a High Volume, Velocity and Variety to require specific Technology and Analytical Methods for its transformation into Value. The data itself is defined at the beginning by three different aspects. Volume is the size of the data set. Velocity is the speed at which new data becomes available. Variety are the differences in types and structure of the data. In combination, the datasets for any Big Data application become highly complex to handle.

The second part of the definition mentions technologies and algorithms. While there are traditional tools available to process data, those are mostly not applicable in the case of Big Data due to a number of reasons. For example the data set may be simply to big to be processed by them, or they can't handle its complexity in terms of variety in structure. Big Data technologies and algorithms are made especially to tackle those problems by not relying on pre-defined structures and being able to process vast amounts of data using multiple computers in parallel.

As with any type of analysis, some kind of value is being extracted from the input data, which is mentioned in the last part of the definition. In the case of Big Data, that value often comes in the form of insights that have been previously unknown. A good example for this are the recommendation systems, like the ones used by streaming services and online retailers. Machine learning algorithms are used to find new correlations and patterns in the data to gain a competitive advantage over the competition.

In summary, Big Data tries to utilise the vast amounts of data that are being generated and whose rate of generation keeps increasing. That utilisation comes in the form of new technologies and analytical methods that help in gaining insights into the data.

Starting the Virtual Machine

The virtual machine is provided in form of a *Vagrantfile* together with a number of provisioning scripts and configuration files that can be found in the GitHub repository (`https://github.com/dkaisers/A-Practical-Guide-to-Big-Data`) accompanying this guide. This site itself as well as all of the exemplary data is also present in the repository and will be uploaded to the virtual machine upon starting it for the first time.

In order to start the virtual machine, download the repository and navigate into the `virtual-machine` folder in the command line of your choosing. After that, simply issue the `vagrant up` command. Starting the virtual machine for the first time will take a few minutes, as the script needs to download all of the necessary software packages and install them.

To pause the virtual machine, only use the `vagrant suspend` command. This will allow the virtual machine to be used later again by unpausing it with `vagrant up`. Please do not use `vagrant halt` to turn off the virtual machine, as the tools may not work properly after completely restarting the virtual machine.

Once the vagrant up command has finished, the web interface that will be used throughout the guide can be viewed by navigating to `http://192.168.33.10:8080/`. If this IP address is causing issues for your specific network configuration, it can be changed in the Vagrantfile.

Apache Zeppelin

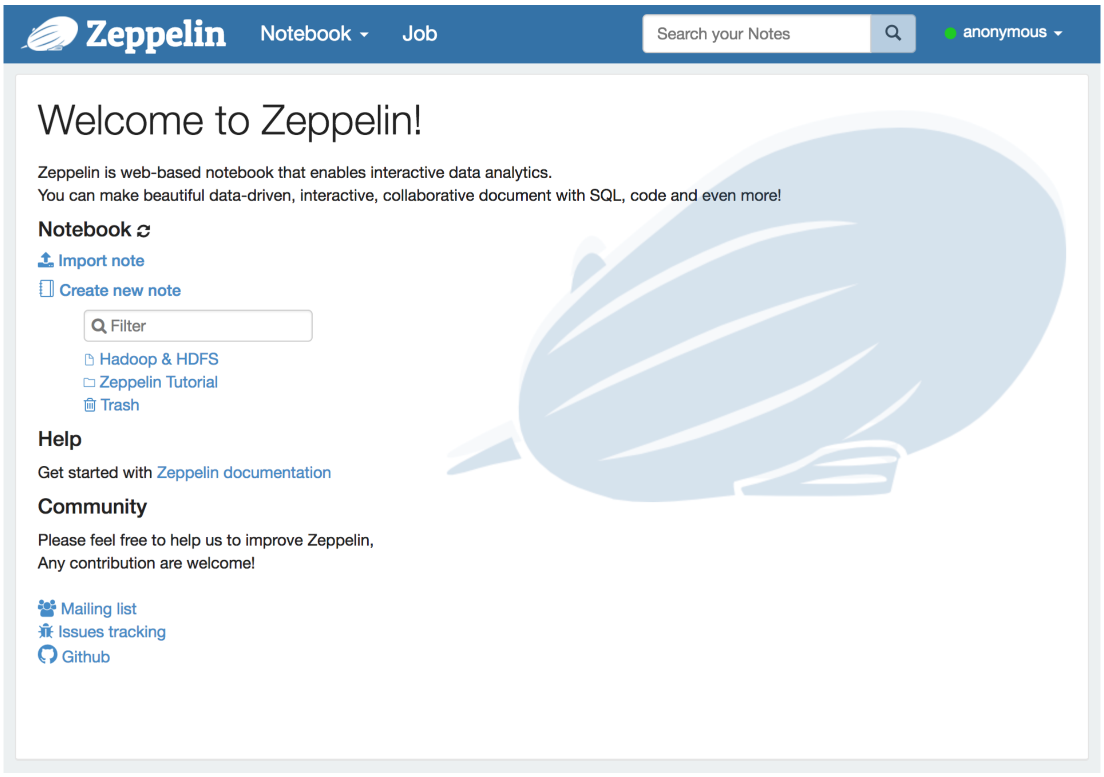
The Apache Zeppelin (`http://zeppelin.apache.org/`) project is a web-based notebook application. It allows to interact with a number of different Big Data tools through so-called notebooks. These can also be shared collaboratively as well as exported and imported on different deployments.

As a Apache project, it is completely free to use and open-source. It is a great tool to analyse and experiment with a data set. There are a number of different alternatives available. It should be said though, that not all user interfaces for Big Data look like this type of tool or are even remotely structured like it. Various tools for different purposes can be found that allow access to a Big Data infrastructure and help with the analysis and visualisation. Most of those tools are commercial products.

On the side of Big Data driven features for end user applications, most user interfaces are implemented by the application itself. Examples for this are the various different recommendation engines in applications like Netflix, Spotify or Amazon. Another prominent example are advertisements like the ones from Google, that use Big Data in combination with machine learning to deploy personalised advertisements to each single user.

Using Zeppelin

As previously mentioned, the web interface for Zeppelin can be reached by navigating to `http://192.168.33.10:8080/` in the browser. On the start page of Zeppelin you will find a list of notes, that currently only contains the included examples. Click on *Create new note*, enter `Hadoop & HDFS` as the name and select sh as the default interpreter. After clicking *Create Note*, the note will be created and you will be redirected to its page.



Welcome to Zeppelin!

Zeppelin is web-based notebook that enables interactive data analytics.
You can make beautiful data-driven, interactive, collaborative document with SQL, code and even more!

Notebook

Import note
Create new note
Filter
Hadoop & HDFS
Zeppelin Tutorial
Trash

Help

Get started with Zeppelin documentation

Community

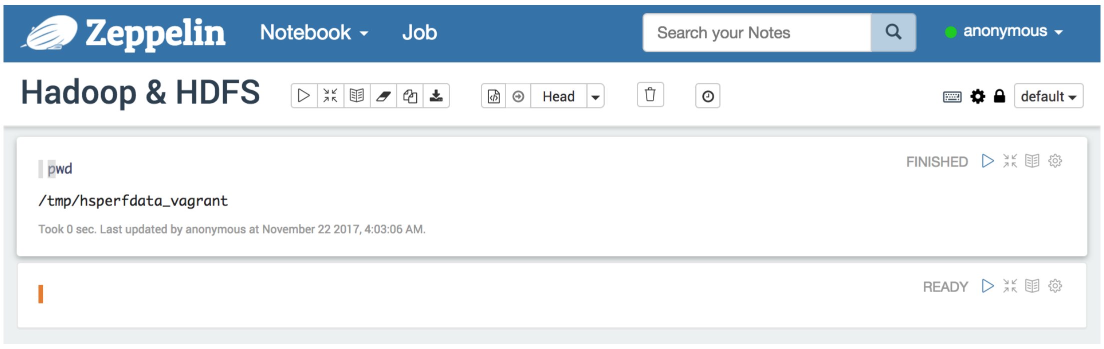
Please feel free to help us to improve Zeppelin,
Any contribution are welcome!

Mailing list
Issues tracking
Github

A note is basically a blank sheet of paper to analyse the data on. Each note consists of one or more paragraphs, with each paragraph being able to simply display some information or execute a set of commands in one of Zeppelin's supported tools. There is also the option to execute the complete note on a schedule by clicking on the small clock icon at the top. This allows the note keep displaying the most current results of your analysis without manually having to re-execute the steps everytime.

You will see an empty paragraph on your note. Click on it to focus it and a blinking cursor should appear. Now type in the bash command pwd and execute the paragraph by either clicking on the play icon in its top right corner, or pressing `Shift + Enter`.

The result should look like the image below, displaying the current folder of the shell. This works because we selected sh as the default interpreter for this note. An interpreter is a part of Zeppelin that is used to execute the commands in the paragraph. In the case of sh, the interpreter is a simple shell like the one you used to start the virtual machine.



```
pwd
/tmp/hsperfdata_vagrant
```

Took 0 sec. Last updated by anonymous at November 22 2017, 4:03:06 AM.

We will be using a number of different interpreters throughout the guide. To select an interpreter for a paragraph that is not the default one of the note, simply put something like `%sh` in the first line of the paragraph's commands. For now, our default shell interpreter is all we need.

Apache Hadoop, HDFS & MapReduce

The Apache Hadoop (`http://hadoop.apache.org/`) software library is the heart and soul of nearly every Big Data application. Besides some common utilities for it's usage, it consists of three different modules that make up the basis for a multitude of other tools that are built on top of Hadoop.

The first module is HDFS, short for *Highly Distributed File System*. It is a file system that can run distributed over multiple servers and focuses on throughput. This means, the data center is not limited by the performance and storage of a single system, but can easily scale up its operation with increasing demand.

The second module is named YARN, short for *Yet Another Resource Negotiator*, and takes care of scheduling the execution of jobs on the Hadoop cluster as well as managing its resources. We won't be directly interacting with it in this guide, but it still works for us.

The third module is called MapReduce and will be explained in more detail in one of the following sections of the guide.

Using HDFS

As a first analysis task, we will be doing a word length count on a text file, which counts how often each length of a word occurs in the text. The file is located on the virtual machine under `/home/vagrant/examples/wordLengthCount/alice.txt`. It is the text to Lewis Carroll's fantasy novel *Alice's Adventures in Wonderland*. But to utilise Hadoop, it first needs to be put onto HDFS.

In order to achieve that, replace the `pwd` in the first paragraph of the still opened up *Hadoop & HDFS* notebook with the following line of code.

```
hdfs dfs -put /home/vagrant/examples/wordLengthCount/alice.txt alice.txt
```

This command will tell HDFS to load the local file onto the distributed file system. In this case it is still on the same computer, as we only work with a single virtual machine, but in a real-world scenario, Hadoop will decide for itself where to put the file exactly.

To check that the file was successfully loaded onto HDFS, execute the `hdfs dfs -ls` command in the paragraph. It should result in a list containing a single item, the `alice.txt` file. There are various other sub-commands that can be substituted for the `ls` part of the last command. The sub-commands are based on the traditional commands used on the command line to navigate and manipulate the file system. For example, `rm` is used to remove files from the HDFS and `mkdir` is used to create new folders.

MapReduce

Now that we have our text in HDFS, let's do the actual word length count. That is where the third Hadoop module, named MapReduce, will come into play. MapReduce is a programming model to process large data sets (with the help of YARN and HDFS) in parallel.

Every MapReduce program consists of two parts, a *mapping function* and a *reduction function*. First the mapping function will create key-value-pairs from the input data, after which the reduction function will process the pairs grouped together by their key.

In the case of the word length count, for each word in the text, the mapping function will create a key-value-pair consisting of the length of the word as the key and a value of 1, which indicates that the respective length was found one time in the text. The reduction function will then get the pairs grouped by the length of words and simply sum all values together to create the result.

In a real Hadoop cluster with multiple servers, each server will execute the map function on its local data on the HDFS, and there is an intermediate *shuffle step* between the mapping and the reduction functions. The shuffle will redistribute the data based on the keys so that each group is located on a single server in order for the reduction to work properly. This allows for the parallel processing of large amounts of data across multiple machines.

There is another file besides `alice.txt` in the example folder of the word length count, which is the already compiled program called `WordLengthCount.jar`. As Hadoop is created in Java, it is also the most common language in which MapReduce programs are created. The next sections will explain each function in more detail and finally execute the program.

Mapping Function

The mapping and reduction functions are actually full-fledged Java classes, extending Hadoop's provided base classes and overwriting the necessary functions for their desired application. The following code snippet is the *Mapper class* of the word length count.

```
public static class Map
    extends Mapper<Object, Text, IntWritable, IntWritable> {

    @Override
    public void map(Object key, Text value, Context context)
        throws IOException, InterruptedException {

        StringTokenizer tokenIterator = new StringTokenizer(value.toString());
        while (tokenIterator.hasMoreTokens()) {
            String word = tokenIterator.nextToken();
            context.write(new IntWritable(word.length()), new IntWritable(1));
        }
    }
}
```

Each Mapper must extend the Hadoop `Mapper<KEYIN, VALUEIN, KEYOUT, VAL-UEOUT>` class and define the types for the input and output values using Java's generics functionality. Hadoop also provides special type classes to be used, and custom ones can be created. This allows a MapReduce program to be able to process virtually any type of data. In our case, `Text` basically represents a `String` and `IntWritable` an `int`.

The `map` function gets the text of the `alice.txt` file as a `Text` value. First, the contained `String` is split apart using the `StringTokenizer` class, which splits the string at each occurence of whitespace. Next, the results of the tokenization are iterated over and for each token or word the `map` function outputs a key-value-pair of the word's length as key and the value of `1`.

Reduction Function

Next up is the reduction function. Similarily to the mapping function, it is its own class that extends Hadoop's `Reducer` base class. And again, the class's input and output types for keys and values can be customised using generics, to allow for the most flexibility in using MapReduce.

```
public static class Reduce
    extends Reducer<IntWritable, IntWritable, IntWritable, IntWritable> {

    @Override
    public void reduce(IntWritable key, Iterable values, Context context)
        throws IOException, InterruptedException {

        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        context.write(key, new IntWritable(sum));
    }
}
```

In the case of the word length count, the reduction function will simply go through all values for a key and sum them up to create a total count for how often each length of word occurs in the text. The reduce function gets the values as an `Iterable`, so this is done simply by iterating over all the values and adding each one to a variable holding the current total. Afterwards, the resulting sum is written out.

Executing the Program

As mentioned before, an already compiled version of the word length count is provided in the same folder as the `alice.txt` file. Besides including the Mapper and Reducer classes, it configures a few additional things for the program to run on Hadoop. More info on this can be found on Hadoop's website. It is skipped in this guide as MapReduce is mainly introduced to explain what happens in the background when using more high-level tools to interact with a Hadoop cluster.

Using the blank paragraph at the bottom of the Zeppelin notebook, execute the following two lines using the shell interpreter. The first one will make sure that any previous results are deleted from HDFS before executing the MapReduce program by calling the Hadoop command line tool with the program file, a name for the job, the input file(s) and the directory on HDFS to store any output specified as arguments. The execution will take a few moments and print out a number of log entries produced by Hadoop.

```
hdfs dfs -rm -f -r wordLengthCounts/output
hadoop jar /home/vagrant/examples/wordLengthCounts/WordLengthCount.jar
    WordLengthCount alice.txt wordLengthCount/output
```

Once the program execution has finished, execute the following line in yet another paragraph with the shell interpreter. It will print out the results produced by the word length count example. Displayed on the left of each output line is the length of a word and on the right the number of times a word of that length occurs in the text.

```
hdfs dfs -cat wordLengthCount/output/*
```

The `*` wildcard operator at the end of the HDFS path assures that any file produced by the program will be printed out. The number of result files depends on the number of Reducers used by the job. This happens for example on a cluster of multiple servers. Each Reducer will produce its own result file. There is also the option to download a merged version of all result files from the HDFS using the `hdfs dfs -getmerge` command line option.

Apache Pig

There is a lot of work involved in creating MapReduce programs, which is why tools exist on top of MapReduce that abstract that process by providing other means of analysing the data at hand. One of those tools is Apache Pig (`http://pig.apache.org/`). It offers the user a high-level language to analyse data, called *Pig Latin*, that is compiled to a sequence of MapReduce-programs by the tool itself.

This allows users to effortlessly experiment with data sets without writing and compiling specific MapReduce-programs for each and every query on the data set. To showcase this, in a first step we will reproduce our simple word length count MapReduce-program with Apache Pig.

Word Length Count Simplified

As a first step, create a new notebook named Pig with pig as the default interpreter. There are actually two different Pig interpreter built into Apache Zeppelin. The default one, `%pig.script`, is to be used as a shell to run Pig Latin scripts that do not need any type of output. An example for this would be the pre-processing of some data that is then stored on HDFS. The other interpreter is `%pig.query` and expects the last line of the Pig script to return data in order for Apache Zeppelin to display the results in one of its built-in visualisations.

As we will be using the latter one of the two Pig interpreters, the first line of our Pig script should read `%pig.query`. Following this statement is the line `lines = LOAD ’alice.txt’;`. This will load the text file `alice.txt` from HDFS and store its contents separated by lines into the `lines` alias, which basically functions as a variable. Pig calls each of those lines tuples. A tuple can consist of one or more fields of data.

The next step is to split the lines of text into their respective words. In order to achieve this, line number three of our script iterates over each line with the `FOREACH` statement, splits the line into its words by using the `TOKENIZE` function and stores the result into the `wordsPerLine` alias. The script now looks like this.

```
%pig.query

lines = LOAD ’alice.txt’;
wordsPerLine = FOREACH lines GENERATE TOKENIZE($0) AS words;
```

The keyword `AS` gives the resulting data points an identifier and the `$0` part of the line accesses the first data point of each line stored in `lines`, which in our case is also the only one: the line of text itself.

The result of the last line of the script is a list of each line of text split into its words. In order to transform this into a single list of words, we will again iterate over the data using `FOREACH`, but now use the `FLATTEN` function to split the list of words per line up.

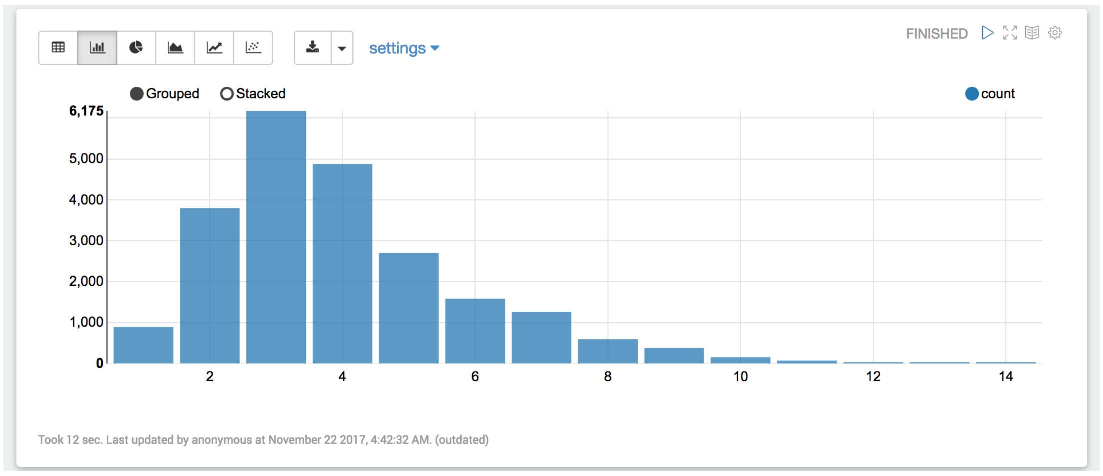
Following the transformation into a list of words is a `FILTER` statement using the `MATCHES` operator with a simple regular expression to only include those tuples that are clearly words of alphanumeric nature. This is an additional step in comparison to the previous MapReduce example, because the Pig `TOKENIZER` function works not as good as Java’s `StringTokenizer` class. After that, another `FOREACH` statement in combination with the `SIZE` function is used to calculate the length for each word.

Finally, the tuples are grouped by the length of word using the `GROUP BY` statement and for each group the count of words of that length is calculated using another `FOREACH` statement with a `COUNT` function. All in all, the Pig Latin script version of our word length count example looks as follows:

```
%pig.query

lines = LOAD ’alice.txt’;
```

```
wordsPerLine = FOREACH lines GENERATE TOKENIZE(\$0) AS words;
words = FOREACH wordsPerLine GENERATE FLATTEN(words) AS word;
words = FILTER words BY word MATCHES ’\\w+’;
wordLengths = FOREACH words GENERATE SIZE(word) as length;
lengthGroups = GROUP wordLengths BY length;
FOREACH lengthGroups GENERATE group AS length, COUNT(wordLengths) AS count;
```

Beware that the last line of the Pig script does not store the tuples into an alias. The `%pig.query` interpreter expects the last line of the script to return some data to be visualised in the Zeppelin paragraph. Executing the script will display a table of data with two columns for the length of a word and the corresponding number of words with that length. Pressing the button for a bar chart will display the result in a nice and clean visualisation that should look like the one below.



Took 12 sec. Last updated by anonymous at November 22 2017, 4:42:32 AM. (outdated)

CSV Handling

Our results from the word length count example can be easily made reusable using Apache Pig to store the analysis results in the HDFS. In this case we will be creating a simple CSV file to store the results in.

Copy the script from the previous section to a new paragraph in our Pig notebook. Change the interpreter to `%pig.script` and prepend the last line with `results =` to store the results in an alias. The last step is to append `STORE results INTO ’wordLength-`

`Counts.csv' USING PigStorage(';');` to the script. In the end it should look like this:

```
%pig.script

lines = LOAD 'alice.txt';
wordsPerLine = FOREACH lines GENERATE TOKENIZE(\$0) AS words;
words = FOREACH wordsPerLine GENERATE FLATTEN(words) AS word;
words = FILTER words BY word MATCHES '\\w+';
wordLengths = FOREACH words GENERATE SIZE(word) as length;
lengthGroups = GROUP wordLengths BY length;
results = FOREACH lengthGroups GENERATE group AS length, COUNT(wordLengths) AS count;
STORE results INTO 'wordLengthCountsPig' USING PigStorage(';');
```

Executing this altered script will still calculate the word length counts in the `alice.txt` file, but instead of displaying them in Zeppelin, the results will be stored as a CSV file in the HDFS in the folder *wordLenghtCounts*. Loading the results and visualizing them is now as easy as executing the line `LOAD 'wordLenghtCounts' USING PigStorage(';');` using the `%pig.query` interpreter.

This ease of use makes Pig a great tool to experiment with a data set and explore different features of the data using Zeppelin's visualisation capabilities. And if there is some functionality missing from Pig, there is the possibility to create *user defined functions* in another programming language like Java or Python

Spark

The next level up of Apache Pig is Apache Spark (`http://spark.apache.org/`), which works in a similar fashion, but is more powerful due to its higher flexibility in handling the data and integrated abilities for machine learning. It is a framework for data analysis that can be used with the Scala, Java, R and Python programming languages. The latter of which we will be using in this guide. The main difference between Spark and a tool like Pig is, that Spark does not use MapReduce but includes its own parallel execution scheme that processes the data in-memory. This is a lot faster, but also prone to memory limitations when working with large data sets.

Data Set

Before going into more detail with the Spark examples, let's take a look at the data set for this chapter of the guide. Published by the UK government, the Road Safety Data (`https://data.gov.uk/dataset/road-accidents-safety-data`) data set includes detailed information on all British traffic accidents with human injuries from 2009 to 2016. All in all the data set contains **1 176 602** distinct traffic accidents.

The first step is to put the data onto HDFS for further analysis. The `/home/vagrant/examples/accidents/` folder of the repository contains a CSV file for each year of data named `accidents_*.csv`. Create a new folder on the HDFS called `accidents` by supplying the `hdfs dfs -mkdir` accidents command to a Zeppelin paragraph with the shell interpreter. Afterwards upload the data by issuing the `hdfs dfs -put /home/vagrant/examples/accidents/accidents_*.csv accidents/` command and verify that the data was successfully loaded into HDFS using the `-ls` command

With the data on the HDFS, create a new Zeppelin notebook called `Spark`. In the first paragraph, execute the following lines to display a subset of the complete data in Zeppelin's build in table visualisation.

```
%pyspark

accidents = spark.read.option("header", "true").csv("hdfs://localhost:54310/user/
    vagrant/accidents/*.csv")

z.show(accidents)
```

`%pyspark` tells Zeppelin to use the Python interpreter in combination with Spark. Within this environment, the `spark` variable can be used to access the Spark session and `z` to access the Zeppelin context. The Spark session is used to read all CSV files in the `accidents` folder on the HDFS. The `option("header", "true")` part of the line tells Spark, that each file contains a header line to use as column names. Afterwards the data is exposed to the Zeppelin context using the `z.show(accidents)` command.

The data displayed in the table does not contain any textual data. Besides identifying information like a unique index per accident, or numerical values like the latitude and longitude of an accident's location, the dataset only contains coded data. The accompanying *Road-Accident-Safety-Data-Guide.xls* file explains how to interpret the different values that are encoded and can be found in the same folder as the accident data CSVs.

Simple Analytics

As a starting point to experiment with this data, we will calculate and visualise the total number of accidents per year. It's helpful to keep the table of data open at the top of the notebook to be able to check what data is available and how it is structured. So, in the second paragraph of the Spark notebook, start by defining the `%pyspark` interpreter to be used again.

The first line of actual Spark/Python code stays the same as well. Copy the line from the previous paragraph to read the CSV files and store the data in the `accidents` variable. Next, extract the year of each accident from the data. The data is stored in the form of a `DataFrame` object, which provides a number of different methods to access the data similar to Apache Pig and other querying languages like SQL. The year is stored in the Date column as a textual value in the form of `dd/mm/yyyy`. Append the following line to the script.

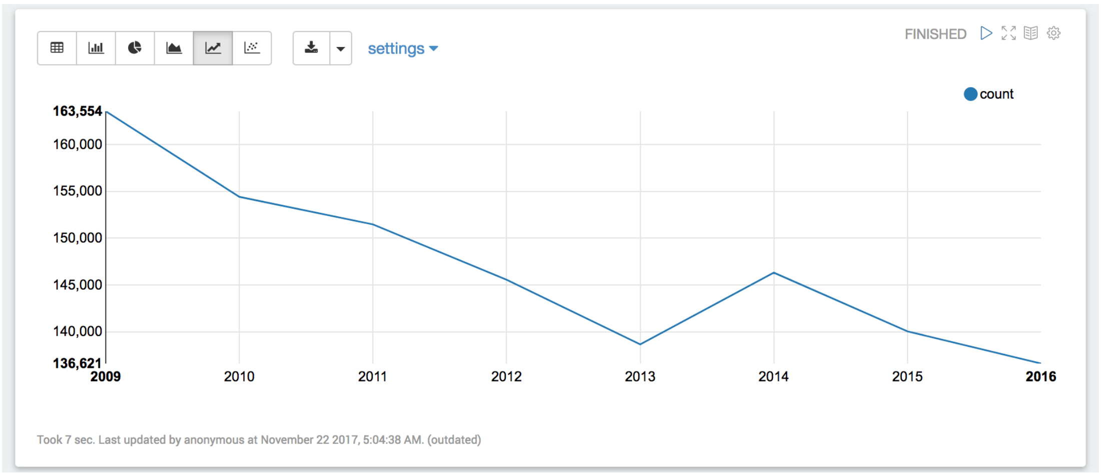
```
years = accidents.select(accidents["Date"].substr(7, 4).alias("year"))
```

The `select` function works exactly as in SQL. It expects a list of columns to be read into the new variable. In this case, we only want to store the year of each accident in the `years` variable. To achieve this, first the `Date` column is selected using `accidents["Date"]`. Secondly, the `substr(7, 4)` function extracts the last four characters of each `Date` value, which contain the year. Finally, the the newly extracted information is said to be stored in the `year` column using the `alias("year")` function.

After extracting the year of each accident, the next step is to calculate the total number of accidents per year. This is done by using the `groupBy` function. It expects a list of columns by which the data should be grouped and the result is a special type of `DataFrame` object that exposes the option of different aggregate functions to be called on the grouped data. In our case, a simple count of the records in each group will do. The final step of the script is to expose the data to the Zeppelin context for visualisation. The complete script looks like this:

```
%pyspark

accidents = spark.read.option("header", "true").csv("hdfs://localhost:54310/user/
    vagrant/accidents/*.csv")
years = accidents.select(accidents["Date"].substr(7, 4).alias("year"))
countsByYear = years.groupBy("year").count()

z.show(countsByYear)
```

Executing the script will result in another data table to be displayed. Selecting the Line Chart from the list of available Zeppelin visualisations will display a nice graph that indicates an overall trend of a decreasing number of accidents each year with a small intermediate increase in 2014.



Took 7 sec. Last updated by anonymous at November 22 2017, 5:04:38 AM. (outdated)

If your graph does not look similar to the one above, check under the visualisation's settings that `year` is selected as *Keys* and `count SUM` is selected as *Values*.

The total number of accidents per year example includes a single feature into the analysis and visualisation. Most of the value that can be extracted from a data set is done so by using multiple features in conjuncture. To demonstrate this, the next example will look into the number of accidents by day of week and time of day.

The script starts exactly the same as before. First, `%pyspark` is set as the interpreter to use and then all accidents are loaded from HDFS into the `accidents` DataFrame. The relevant columns of the dataset are `Day_of_Week` and `Time`. While `Day_of_Week` can be used without any modifications, the `Time` values need to be modified in order to be grouped by the hour. Grouping the data by the `Time` column directly, would result in too many different value groups for Zeppelin to visualise.

Besides that, some accidents do not have a value for the `Time` column and need to be excluded using the `filter` function of a DataFrame. Given the parameter `"Time != ’null’"`, it will only return those records, that have a valid value for the `Time` column. Lastly, the hour values should be converted from textual values to numerical values using the `cast("int")` function. This ensures, that the data will be properly ordered in Zeppelin visualisations. Without this measure, the leading zero of some hour values would cause them to be sorted behind all other values. All in all these steps can be done using the following lines of the script:

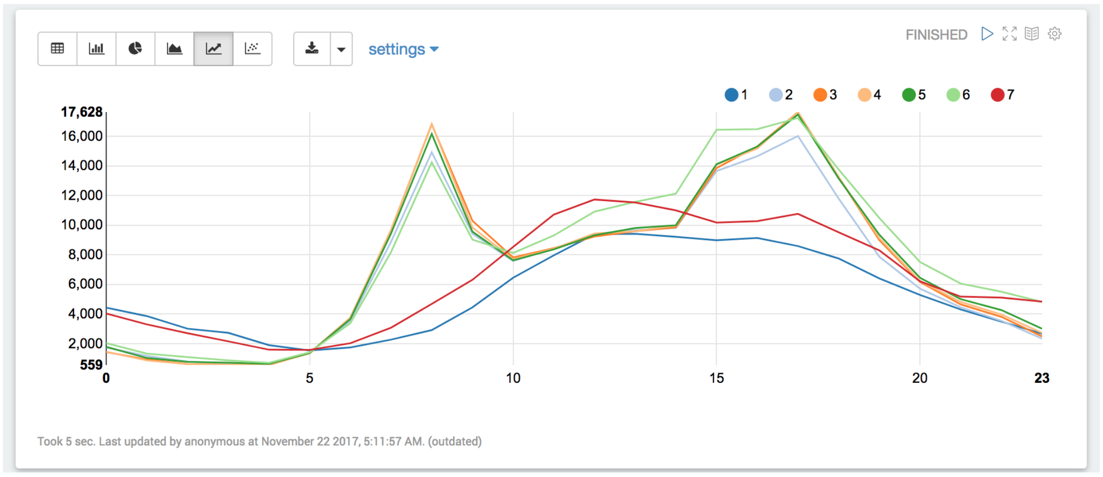
```
accidents = accidents.filter("Time != ’null’")
accidents = accidents.select(accidents["Day_of_Week"], accidents["Time"].substr(0, 2)
    .cast("int").alias("Hour"))
```

Following up by grouping the data by the `Day_of_Week` and `Time` columns in combination with the `count()` function will result in the desired data that can be exposed to the Zeppelin context using `z.show(accidents)`. In the end, the script should look like this:

```
%pyspark

accidents = spark.read.option("header", "true").csv("hdfs://localhost:54310/user/
    vagrant/examples/accidents/accidents_*.csv")
accidents = accidents.filter("Time != ’null’")
accidents = accidents.select(accidents["Day_of_Week"], accidents["Time"].substr(0, 2)
    .cast("int").alias("Hour"))
accidents = accidents.groupBy("Day_of_Week", "Hour").count()

z.show(accidents)
```

Executing the script will display a data table that is not easy to extract any insight from. Switching the visualisation to the Line chart again and configuring the visualisation so that the *Keys* contains `Hour`, *Groups* contains `Day_of_Week` and Values contains `count SUM` will change that. The resulting line chart displays the number of accidents per hour of day with a differently colored line for each day of the week.

The legend for the colours can be found in the top right corner. As it is encoded information, the accompanying file of the data set needs to be consulted. In this case, *1* stands for Sunday with each increasing number going forward one day until *7*, which stands for



Took 5 sec. Last updated by anonymous at November 22 2017, 5:11:57 AM. (outdated)

Saturday. While analysing the chart, keep in mind that this data is out of context. The total number of accidents needs to be seen relative to the actual amount of traffic at any given time of day, since the more traffic exists, the more likely accidents happen.

But even without context, a few interesting observations can be made. For one, rush hours in the morning and afternoon seem like the most dangerous times to drive. Interestingly on Friday afternoons, the spike increases earlier than on the other workdays. On the other hand, weekends are generally safer than workdays with the exception of Saturdays at noon and the night times between 12 AM and 5 AM. Additionally, 5 AM appears to be the overall safest time to drive as it has the lowest number of accidents when looking at the week as a whole.

Machine Learning

One of the most common use cases for Big Data is Machine Learning, where the computer learns about the data it is given without explicitly being told what to look at (as we did in our previous examples). Fortunately, Spark has a few different Machine Learning algorithm built in.

In the following final example of this guide, we'll be using *K-Means clustering* with our traffic data set. This algorithm divides the data into k clusters according to the set of features it is given. It is generally used to identify hidden correlations and or coherencies in the data set as the algorithm will group those data instances together that it sees as similar according to their features.

In our simulated use case, we want to establish a new nationally operating towing service in the UK and therefore want to identify 10 places that will have the best spheres of influence in terms of reachable traffic accidents. Since this use case asks for geographical clustering of the data, using the latitude and longitude of each accident is sufficient as a feature set for K-Means. A more general approach for situations without a specific use case would be to use all available features to cluster the data and sequentially reduce the set of features until all of the irrelevant ones are excluded from the clustering. The clustering then requires further analysis to determine the specific nature of each cluster according to the dominating values for each of the remaining features.

The start of the Python script differs slightly from previous examples, as additional dependencies need to be imported: the actual Machine Learning algorithm and the data type used by it. After this initial difference, the script returns to the usual sequence with loading the data set and selecting the relevant and valid records.

```
%pyspark

from numpy import array
from pyspark.mllib.clustering import KMeans, KMeansModel

accidents = spark.read.option("header", "true").csv("hdfs://localhost:54310/user/
    vagrant/accidents/accidents_*.csv")
accidents = accidents.filter("Latitude != 'null' AND Longitude != 'null'")
accidents = accidents.select('Latitude', 'Longitude')
```

The K-Means algorithm of Spark requires the data to be in a specific format that has all the relevant values in an array. Therefore the next line of the script iterates over all records and maps the data into the desired format. Using the reformatted data, the K-Means algorithm is called and returns a `KMeansModel` object holding all the information extracted by K-Means. The parameters of the call determine that 10 clusters should be generated and that the initial positions of the clusters are randomised at the start of the algorithm. This causes the results to slightly differ each time the script is executed as K-Means uses the starting positions and adjusts them more and more using the given data.

```
accidents = accidents.rdd.map(lambda row: array(row.asDict().values()))
clusters = KMeans.train(accidents, 10, initializationMode = "random")
```

The last step of the script is to display the results in Zeppelin the centre coordinates of each cluster that can be found in the `clusters.clusterCenters` attribute. But

as the resulting data is no longer a DataFrame object, the normal `z.show()` call can't be used. Instead we have to output the data by hand. First a `print("%table")` call tells Zeppelin to expect tabular data with each column separated by a *tab-character*. The following `print("Latitude`
`tLongitude")` determines the column headers. Without it, the first line of data would be used. Finally, the script iterates of the cluster centres and prints each one out. All in all, the script should look like this:

```
%pyspark

from numpy import array
from pyspark.mllib.clustering import KMeans, KMeansModel

accidents = spark.read.option("header", "true").csv("hdfs://localhost:54310/user/
    vagrant/accidents/accidents_*.csv")
accidents = accidents.filter("Latitude != 'null' AND Longitude != 'null'")
accidents = accidents.select('Latitude', 'Longitude')
accidents = accidents.rdd.map(lambda row: array(row.asDict().values()))
clusters = KMeans.train(accidents, 10, initializationMode = "random")

print('%table')
print('Latitude\tLongitude')

for center in clusters.clusterCenters:
    print(str(center[0]) + '\t' + str(center[1]))
```

Executing the script will result in a table of 10 rows. Each row contains the latitude and longitude of a cluster center, which represents the 10 geographical positions that have the best access to the most accidents according to the data from 2009 to 2016. These positions can be used in our use case to determine the specific locations of stations for the new national towing service.

Better Visualisation with Matplotlib

Looking at latitude and longitude values does not give a good idea where the cluster centres are actually located and the available Zeppelin visualisations are not equipped for this use case. Therefore the following section will create a custom visualisation using Python and its Matplotlib (`https://matplotlib.org/`) module, which allows for the creation of numerous different graphs and charts. In this specific instance, we'll be using the Basemap (`https://matplotlib.org/basemap/`) extension to display the location of the cluster centres on an actual map of the UK.

First of all we need to alter the cluster generating script by adding two more imports at the top. `from mpl_toolkits.basemap import Basemap` for the Basemap extension and `import matplotlib.pyplot as plt` for the Matplotlib itself. The part of the script responsible for loading the data set and calculating the clusters can stay unmodified, but remove the last the last part of the script that outputs the data to Zeppelin.

The first step for the custom visualisation is to create the Basemap itself and configure it so that only the UK is displayed instead of the whole earth. Besides that, configuring the colours of the map is recommended to produce a visually more pleasing result. The following four lines of code will do exactly that. We will be using a Mercator projection of the earth, as established by the `projection = ’merc’` parameter. The following lines of code define the area to be displayed using latitude and longitude values for the lower left and upper right corner of the map and the `resolution = "i"` parameter defines how detailed the coastlines are being drawn. Afterwards the colours are defined.

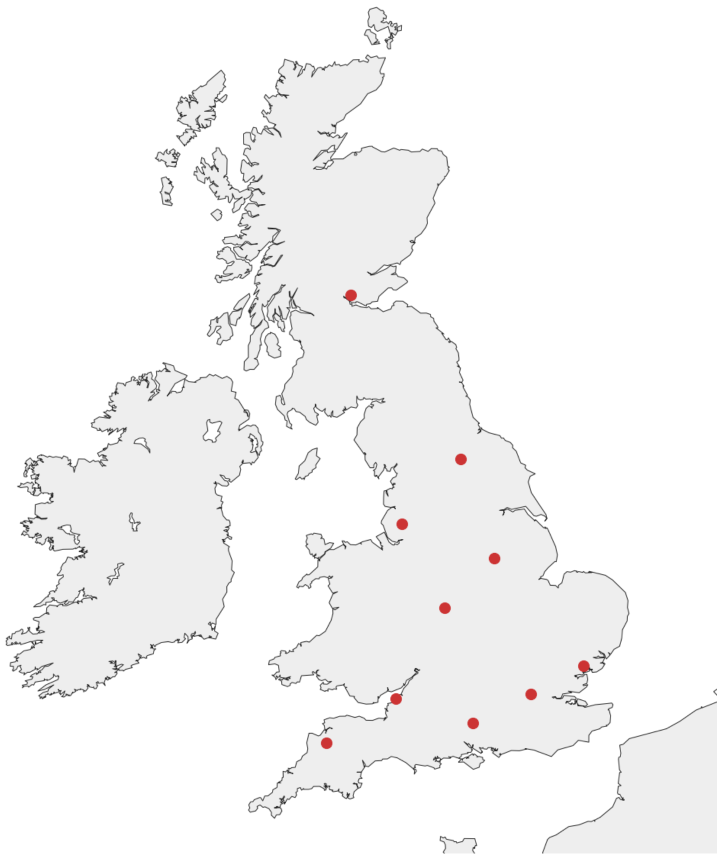
```
basemap = Basemap(projection = "merc", llcrntlat = 49.5, urcrnrlat = 59.5, llcrnrlon
    = -12, urcrnrlon = 3.5, resolution = "i")
basemap.drawmapboundary(linewidth = 0, fill_color = ’#ffffff’)
basemap.fillcontinents(color = ’#eeeeee’)
basemap.drawcoastlines(color = ’#222222’)
```

Drawing the actual cluster centers involves iterating over them again. But this time, a call to the `basemap()` function will calculate the coordinates on the Basemap from the latitude and longitude values. Using the `scatter()` function of the basemap object, each cluster is plotted onto the map. To define the size of the resulting graphic, the line `z.configure_mpl(width=400, height=300, fmt=’svg’)` is added. Finally a call to `plt.show()` will ensure a clean visualisation in Zeppelin without any intermediate outputs. Putting everything together, the script should look like this:

```
%pyspark

from numpy import array
from pyspark.mllib.clustering import KMeans, KMeansModel
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt

accidents = spark.read.option("header", "true").csv("hdfs://localhost:54310/user/
    vagrant/accidents/accidents_*.csv")
accidents = accidents.filter("Latitude != ’null’ AND Longitude != ’null’")
accidents = accidents.select(’Latitude’, ’Longitude’)
accidents = accidents.rdd.map(lambda row: array(row.asDict().values()))
clusters = KMeans.train(accidents, 10, initializationMode = "random")
```

```
# Create the basemap
basemap = Basemap(projection='merc',llcrnrlat=49.5, urcrnrlat=59.5, llcrnrlon=-12,
    urcrnrlon=3.5, resolution='i')
basemap.drawmapboundary(linewidth=0, fill_color='#ffffff')
basemap.fillcontinents(color='#eeeeee')
basemap.drawcoastlines(color='#222222')

# Plot the cluster centers
for center in clusters.clusterCenters:
    x, y = basemap(center[1], center[0])
    basemap.scatter(x, y, 200, c='#cc3333', zorder = 2)

z.configure_mpl(width=800, height=800, fmt='svg')
plt.show()
```

Executing the script above will produce an image similar to the one below. The clusters won't be placed on the exact locations, as the starting positions are randomized. In any case, this result is a lot more intuitive than a table of GPS coordinates. Playing with the different parameters for the Basemap, like the colors, allows for visualizations that can be seemlessly integrated in any other content and design.



Looking Forward

I hope this guide could help in understanding how to use different types of tools in the field of Big Data. There are many more alternatives that can be used and it would be impossible to include them all. Which ones to choose depends entirely on the specific situation. This guide tries to function as a simple starting point from which the reader can gather enough knowledge to be able to recognise situations in which Big Data could be beneficial and how it could be utilised.

As a next step, I would recommend reading up on all the possibilities the different tools used in this guide offer and experimenting with them using the provided virtual machine. It may also be beneficial to look into different data sets and also different types of data sets. Generally speaking data can be categorised in two different ways. For one, there is *batch data*, as used in this guide, which is a finite data set that is analysed and there is *streaming data*, which is a never ending stream of data that is analysed in real time. A good example for the latter would be Twitter and the never ending stream of new tweets generated by its users.

The other way to categorise data is by its structure. In this guide we mainly used *structured* data, which allows for easy analysis as its features are already exposed. Besides that, there is *unstructured* data and *semi-structured* data. Twitter is again a good example for the former. The content of each tweet is short text. A possible feature to extract from this unstructured data would be the hash-tags used or the other users mentioned.

Semi-structured data is a special form of structured data in which each record contains already separated data attributes, but each record could contain a different set of theses attributes. The Internet of Things is a popular field in which semi-structured data widely exists. Collecting a stream of data from a number of different sensors will have structured data for each reading, but a temperature sensor will deliver different information than a humidity sensor.

Handling Data

These different types of data need to be consolidated before being able to make any analysis. Most of the time, it takes a lot of steps to ingest new data into the system. Steps that also need to be repeated on a schedule, or need to be done constantly in the case of streaming data. Therefore a lot of different tools have been created for the job of data ingestion that go way further than just putting data onto the HDFS. They can collect, pre-process and distribute the data using a multitude of different services and formats.

A good example for this is log data. Every software system creates logs for a number of different events and Big Data can help make sense of those logs. To get the data into the system, a tool like Apache Flume (`https://flume.apache.org/`) can be used, that is specifically designed to collect, aggregate and move log data between systems. Other tools like Apache Sqoop (`http://sqoop.apache.org/index.html`) and Apache Chukwa (`https://chukwa.apache.org/`) serve a more general purpose and can not only handle logs. With them data can be easily read, pre-processed and stored for further analysis.

All in all, data ingestion is one of the most important steps of any Big Data application, as without sufficient data, no value can be extracted. The necessary tools highly depend on the situation, as sources and targets vary from use case to use case. But in any case they can help immensely in handling all kinds of data by automating the necessary steps of ingestion.

Combining tools to ingest new data into the system, analysing the data using different methods and displaying the result using visualisations or explicit integrations into existing software systems will result in a complete Big Data infrastructure that is a lot more complex than the simple examples in this guide. Therefore it is detrimental to read up on the relevant topics and experiment with different tools in order to grasp Big Data any further.

Bibliography

Acharjya, Debi Prasanna and P Kauser Ahmed (2016). “A Survey on Big Data Analytics: Challenges, Open Research Issues and Tools”. In: *Article in International Journal of Advanced Computer Science and Applications February*.

Alpaydin, Ethem (2014). *Introduction to machine learning*. MIT press.

Apache Spark Documentation (n.d.). `http://spark.apache.org/docs/latest/`. Accessed: 2017-11-20.

Bhadani, Abhay Kumar and Dhanya Jothimani (2016). “Big Data: Challenges, Opportunities, and Realities”. In: *Effective Big Data Management and Opportunities for Implementation*. IGI Global, pp. 1–24.

Chen, Min, Shiwen Mao, and Yunhao Liu (2014). “Big data: A survey”. In: *Mobile Networks and Applications* 19.2, pp. 171–209.

De Mauro, Andrea, Marco Greco, and Michele Grimaldi (2016). “A formal definition of Big Data based on its essential features”. In: *Library Review* 65.3, pp. 122–135.

DeVore, Seth, Emily Marshman, and Chandralekha Singh (2017). “Challenge of engaging all students via self-paced interactive electronic learning tutorials for introductory physics”. In: *Physical Review Physics Education Research* 13.1, p. 010127.

Emani, Cheikh Kacfah, Nadine Cullot, and Christophe Nicolle (2015). “Understandable big data: A survey”. In: *Computer science review* 17, pp. 70–81.

Käfer, Verena, Daniel Kulesz, and Stefan Wagner (2016). *What is the best way for developers to learn new software tools? A small empirical comparison between a text and a video tutorial*. Tech. rep. PeerJ Preprints.

Open Government License (n.d.). `http://www.nationalarchives.gov.uk/doc/open-government-licence/version/3/`. Accessed: 2017-11-20.

Oussous, Ahmed et al. (2017). “Big Data Technologies: A Survey”. In: *Journal of King Saud University-Computer and Information Sciences*.

Pääkkönen, Pekka and Daniel Pakkala (2015). “Reference architecture and classification of technologies, products and services for big data systems”. In: *Big Data Research* 2.4, pp. 166–186.

Porter, Michael E (1985). *Competitive Advantage: Creating and Sustaining Superior Performance*.

Pospiech, Marco and Carsten Felden (2012). “Big data–a state-of-the-art”. In:

Qiu, Junfei et al. (2016). “A survey of machine learning for big data processing”. In: *EURASIP Journal on Advances in Signal Processing* 2016.1, p. 67.

Road Safety Data (n.d.). `https://data.gov.uk/dataset/road-accidents-safety-data`. Accessed: 2017-11-20.

Wienhofena, LWM, BM Mathisena, and D Romanb (2015). “Empirical Big Data Research: A Systematic Literature Mapping”. In: *arXiv preprint arXiv:1509.03045*.