Independent degree project – second cycle

Master’s thesis, 30 higher credits

Computer Engineering

Big Data analytics for the forest industry
A proof-of-concept built on cloud technologies

David Sellén
MID SWEDEN UNIVERSITY
Department of Information and Communication Systems
Examiner: Tingting Zhang, tingting.zhang@miun.se
Internal supervisor: Mehrzad Lavassani, mehrzad.lavassani@miun.se
External supervisor: Johan Deckmar, johan.deckmar@dewire.com
Author: David Sellén, dase1001@student.miun.se
Degree programme: Master of Science in Computer Engineering, 120 credits
Main field of study: Computer Engineering
Semester, year: Spring, 2016
Abstract

Large amounts of data in various forms are generated at a fast pace in today’s society. This is commonly referred to as “Big Data”. Making use of Big Data has been increasingly important for both business and in research. The forest industry is generating big amounts of data during the different processes of forest harvesting. In Sweden, forest information is sent to SDC, the information hub for the Swedish forest industry. In 2014, SDC received reports on 75.5 million m³fub from harvester and forwarder machines. These machines use a global standard called StanForD 2010 for communication and to create reports about harvested stems. The arrival of scalable cloud technologies that combines Big Data with machine learning makes it interesting to develop an application to analyze the large amounts of data produced by the forest industry. In this study, a proof-of-concept has been implemented to be able to analyze harvest production reports from the StanForD 2010 standard. The system consist of a back-end and front-end application and is built using cloud technologies such as Apache Spark and Hadoop. System tests have proven that the concept is able to successfully handle storage, processing and machine learning on gigabytes of HPR files. It is capable of extracting information from raw HPR data into datasets and support a machine learning pipeline with pre-processing and K-Means clustering. The proof-of-concept has provided a code base for further development of a system that could be used to find valuable knowledge for the forest industry.

Keywords: Big Data analytics, Apache Spark, StanForD 2010, forest industry, harvest production report.
Table of Content

List of Figures ................................................................................................................. vii
List of Tables ...................................................................................................................... viii
List of Listings .................................................................................................................... viii
Terminology ....................................................................................................................... ix

1 Introduction ..................................................................................................................... 1
  1.1 Problem motivation ........................................................................................................ 2
  1.2 Overall aim .................................................................................................................... 2
  1.3 Scope ............................................................................................................................ 2
  1.4 Concrete and verifiable goals ......................................................................................... 3
  1.5 Outline .......................................................................................................................... 3

2 Background and related work ......................................................................................... 4
  2.1 Big Data ........................................................................................................................ 4
  2.2 Data mining ................................................................................................................... 4
    2.2.1 Machine learning ..................................................................................................... 4
    2.2.2 Machine learning pipeline ....................................................................................... 5
    2.2.3 K-Means ................................................................................................................ 6
  2.3 Apache Hadoop ............................................................................................................. 8
    2.3.1 Distributed file system ............................................................................................ 9
    2.3.2 MapReduce .............................................................................................................. 9
  2.4 Apache Spark ............................................................................................................... 10
    2.4.1 Spark cluster ............................................................................................................ 10
    2.4.2 Resilient Distributed Dataset ................................................................................ 11
    2.4.3 Programming model ................................................................................................ 12
  2.5 Spark’s component stack ............................................................................................... 12
    2.5.1 SparkSQL ................................................................................................................. 13
    2.5.2 MLlib ......................................................................................................................... 14
  2.6 Evaluation of performance scalability .......................................................................... 15
  2.7 Standard in forest communication ............................................................................... 16
    2.7.1 Harvest production reports ..................................................................................... 17
  2.8 Related work ................................................................................................................. 19
    2.8.1 Harvest production reports ..................................................................................... 19
    2.8.2 Big data analysis tools for science .......................................................................... 20
# Methodology

3.1 Research method .......................................................... 21  
3.2 Data collection ............................................................... 22  
3.3 Implementation .................................................................. 23  
  3.3.1 The harvest production reports 23  
  3.3.2 K-Means method 23  
  3.3.3 System architecture 24  
  3.3.4 Back-end frameworks 25  
  3.3.5 Front-end frameworks 27  
3.4 System tests and evaluation .................................................. 29  
  3.4.1 Test Environment 29  
  3.4.2 Test data 30  
  3.4.3 Test cases and gathering of primary data 31  
  3.4.4 Data analysis and system evaluation 33  
  3.5 Validity and reliability ....................................................... 33  

# Implementation

4.1 System overview ............................................................... 34  
  4.1.1 Packets and files structure 36  
4.2 Data storage ....................................................................... 37  
4.3 Back-end ........................................................................... 38  
  4.3.1 System boot 38  
  4.3.2 Analysis of HPR data 39  
  4.3.3 REST API 39  
4.4 Front-end .......................................................................... 40  

# Results

5.1 Horizontal scalability test results ......................................... 45  
5.2 Vertical scalability test results ............................................... 47  

# Analysis

6.1 The implementation ........................................................... 51  
  6.1.1 Architecture 51  
  6.1.2 Development frameworks 51  
  6.1.3 System extendibility 53  
6.2 Test results ......................................................................... 54  

# Conclusions

7.1 Ethical considerations ........................................................ 56  
7.2 Suggestions for further research .......................................... 56  

References ............................................................................ 58
Appendix A: Comparison of SparkSQL and Spark .......................... 64
Appendix B: Spark configurations ................................................. 67
Appendix C: System requirements .................................................. 68
Appendix D: Class diagram .......................................................... 70
List of Figures

Figure 1: Data gathering during the harvest process.........................1
Figure 2: Components in a general machine learning pipeline..............5
Figure 3: K-Means clustering on the Iris dataset..............................6
Figure 4: HDFS architecture..................................................9
Figure 5: Spark cluster.........................................................11
Figure 6: Spark’s component stack.........................................13
Figure 7: Simplified illustration of the HPR message structure..........18
Figure 8: Back-end development frameworks..............................25
Figure 9: Front-end development frameworks.............................28
Figure 10: Computer cluster of virtual machines..........................30
Figure 11: System overview....................................................35
Figure 12: Project files and packet structure..............................36
Figure 13: System boot process..............................................38
Figure 14: View of worksites..................................................41
Figure 15: Selecting targeted worksites using the map..................42
Figure 16: UI for loading datasets and apply ML............................43
Figure 17: Clustering results..................................................44
Figure 18: Load of HPR data with different number of nodes..........45
Figure 19: Extract worksites information with different number of nodes..................................................................................46
Figure 20: Create a new dataset with different number of nodes.......46
Figure 21: Save a dataset with different number of nodes................46
Figure 22: Build a K-Means model with different number of nodes.....47
Figure 23: Test the model with different number of nodes................47
Figure 24: Load of HPR data using different numbers of CPU cores....48
Figure 25: Extract worksites information using different numbers of CPU cores........................................................................48
Figure 26: Create new dataset using different numbers of CPU cores..49
Figure 27: Save a dataset using different numbers of CPU cores........49
Figure 28: Build a K-Means model using different numbers of CPU cores..................................................................................49
Figure 29: Test the K-Means model using different numbers of CPU cores..................................................................................50
List of Tables

Table 1: StanForD 2010 messages. ............................................................. 16
Table 2: Data stored in HPR files. ............................................................ 18
Table 3: Hardware specifications of the server machine ......................... 29
Table 4: Hardware specification of the virtual machines ....................... 29
Table 5: Java project's packet structure. ............................................... 37
Table 6: REST API. ............................................................................. 39
Table 7: Horizontal scalability factors. ............................................... 45
Table 8: Vertical scalability factors. ..................................................... 48

List of Listings

Listing 1: K-Means || (k, ℓ), initialization algorithm. ......................... 8
Listing 2: Scala example using Spark for counting error messages. ....... 12
Listing 3: SparkSQL query using SQL. .............................................. 14
Listing 4: SparkSQL query using the DataFrame API. ......................... 14
Listing 5: Steps in the test scenario. ............................................... 31
### Terminology

**Abbreviations and acronyms**

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface.</td>
</tr>
<tr>
<td>CSS</td>
<td>Cascading Style Sheet</td>
</tr>
<tr>
<td>ETL</td>
<td>Extract, Transform and Load</td>
</tr>
<tr>
<td>HDFS</td>
<td>Hadoop Distributed File System.</td>
</tr>
<tr>
<td>HPR</td>
<td>Harvest Production Report.</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>JVM</td>
<td>Java Virtual Machine</td>
</tr>
<tr>
<td>ML</td>
<td>Machine Learning</td>
</tr>
<tr>
<td>PMML</td>
<td>Predictive Model Markup Language</td>
</tr>
<tr>
<td>RDD</td>
<td>Resilient Distributed Dataset</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>SDC</td>
<td>The Swedish forest industries IT-company.</td>
</tr>
<tr>
<td>SPA</td>
<td>Single-Page Application.</td>
</tr>
<tr>
<td>StanForD</td>
<td>Standard for Forest machine Data and Communication.</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language.</td>
</tr>
<tr>
<td>UDF</td>
<td>User Defined Function</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Identifier</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
**Mathematical notation**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m^{3fub}$</td>
<td>Cubic metres solid wood exclusive bark and tops.</td>
</tr>
<tr>
<td>$d(p, q)$</td>
<td>Distance between point p and q</td>
</tr>
<tr>
<td>$k$</td>
<td>The number of clusters used in K-Means clustering.</td>
</tr>
</tbody>
</table>
1 Introduction

Large amounts of data in various forms are generated at a fast pace in today’s society. This is commonly referred to as “Big Data”. Making use of Big Data has been increasingly important for both business and research. All this data can be analyzed to reveal hidden patterns and to find new information. Data mining is a field within computer science, mixing computer processing, databases, statistics and machine learning methods to extract valuable patterns in large data sets [1]. The results from data mining can lead to increased efficiency, profit or in other ways enhance our lives or businesses.

The forest industry generates large amounts of data during the different processes of forest harvesting. The four main processes are shown in Figure 1; harvesting of trees, forwarding timber to the road and transportation from the road to a measuring station for analysis of wood quality. All this information is sent to SDC, the information hub for the Swedish forest industry [2]. SDC is an association owned by about 50 different actors in the Swedish forest industry; it is responsible for the management of forest, timber and transportation information in Sweden.

![Figure 1: Data gathering during the harvest process.](image)

In 2014, SDC administrated information on 152.3 million cubic meters of solid wood excluding bark and tops (m³fub) and had a turnover of about 180 million Swedish kronor (SEK). During the same year, SDC received reports on 75.5 million m³fub from harvester and forwarder machines. These machines use a global standard called StanForD 2010, Standard for Forest machine Data and Communication, to communicate machine and harvest reports [3]. The StanForD 2010 reports are stored at SDC together with information on wood transportation and wood quality measurements.
1.1 Problem motivation

The forest industry should be able to take advantage of the large amounts of data at their disposal. For example, in StanForD 2010 there are a lot of information that could be used for analysis such as stem dimensions, coordinates, and wood species. This information could later be combined with other data such as weather, transportation and wood quality. Combining different data sources and applying machine learning could unveil previously unknown relationships and patterns. For example, analysis could show how different parameter affects wood quality, predict the best time to harvest or help to increase forest production. Currently there is no system for analyzing large amounts of forest related data, built especially for research in the forest industry. The development of a system as such could greatly enhance the forest industry.

1.2 Overall aim

The overall aim of this study is to provide a technical solution for an interactive analytical system based on the latest cloud technologies. The purpose of the system is to discover new and valuable knowledge for the forest industry by applying data mining on large amounts of forest related data.

1.3 Scope

The focus of this study has been on building a proof-of-concept system that implements a complete analytical pipeline. This includes: visualization of raw forest data, the process of extracting information from raw forest data into datasets, applying machine learning on created datasets and display the results. The proof-of-concept has been thought of as a platform for further development. Hence, the study has covered system architecture and implementation of scalable technologies such that the system can be extended. Furthermore, the scalability and the performance of the system have been investigated to identify the resource requirements to perform large-scale analysis.

The proof-of-concept has been limited to the analysis of harvest production reports (HPRs) from StanForD 2010. The HPR data was targeted as the forest data to show the possibilities and opportunities of this appli-
cation. The implementation of machine learning algorithms was limited to support the unsupervised clustering algorithm K-means.

Important aspects that are not addressed in this study are data ingestion, information security, user interaction and graphical design.

1.4 Concrete and verifiable goals

The following questions will be answered through the development of a proof-of-concept analysis system:

- How can cloud technologies be used to develop an analysis system that can handle gigabytes of HPR data?
- How can the system be designed for modularity and extendibility regarding features and data sources?
- What are the most time consuming features of the system?
- How does the system perform and scale regarding computation time depending on the computer clusters hardware resource and computer nodes?

1.5 Outline

Chapter 2 describes the theoretical background and related research. In Chapter 3 the methods are described and motivated, followed by Chapter 4 which contains a detailed description of the implemented system. The results from the tests are presented in Chapter 5 and analyzed in Chapter 6. In Chapter 7, the conclusions and suggestions for further research are presented.
2 Background and related work

In this chapter background knowledge on Big Data, machine learning and frameworks for development are explained as well as previous research related to forestry and other scientific Big Data analytics tools.

2.1 Big Data

We are currently living in a time where the amount of digital data is growing exponentially and finding a way to use all data have become essential for both business and research. The term “Big Data” is often described by three V’s; high velocity, high variety and high volume [4]. In general, this means that the data is too large and fast growing for a single computer to manage. Cisco [5] anticipates that the global IP traffic will surpass one Zettabyte ($10^{21}$ bytes) by the end of 2016 and reach two Zettabytes by 2019. Frost & Sullivan predicts [6] that the global data traffic will be more than 100 Zettabytes by the year 2025. Today’s large volumes of data have resulted in a need for scalable technologies, for both storage and processing. Big Data storages, which store data in its native format, are often referred to as data lakes.

2.2 Data mining

Manual analysis methods are no longer practical with the rapidly growing data of today. A whole new discipline known as data mining has emerged to solve this problem [7]. Data mining consists of methods for automatic or semiautomatic processing to extract valuable knowledge in large datasets [1]. The area of data mining is a combination of different disciplines such as databases, statistics, visualization and machine learning.

2.2.1 Machine learning

One approach to better understand big amounts of data is machine learning. Science and business as well as engineering have found different use cases to apply machine learning [8]. With different machine learning methods we can create decision structures, classify unlabeled objects, predict outcomes, group together similar data and find associated features [1]. The machine learning methods are divided into two types, unsupervised and supervised methods.
Supervised learning is the name of method that makes use of a training set with labeled data to build a model that can be used for labeling unlabeled data. When the data does not have a label that describes its class or when exploring data for hidden structures is of interest, then unsupervised learning methods are applied. The most recognized unsupervised learning methods are clustering algorithms, for example the K-means method. [1]

### 2.2.2 Machine learning pipeline

A system for machine learning generally consists of the components shown in Figure 2 [9]. These components are data ingestion and storage, data cleansing and transformation, building and testing a machine learning model and finally deploying the model.

![Machine learning pipeline diagram](image)

**Figure 2: Components in a general machine learning pipeline.**

The first process is to collected data from the world, for example sensor data, and put in storage [9]. This is the process of data ingestion. The stored data is then cleansed and transformed; this is commonly referred to as extract, transform and load (ETL). ETL is performed to prepare the raw data for analysis and load it into a more convenient storage [1]. This process includes attribute selection, discretizing, cleansing, sampling and aggregation. There are also lots of practical challenges when aggregating data from different sources such as different formats, different primary keys and errors.

When the data have been prepared it is time to apply machine learning to the data. A ML algorithm constructs a model based on a set of data. The model can then be tested and evaluated by applying test data to show how well the model predicts, classifies or cluster data. This process is often performed in iterations to find the best fit algorithm and parameters. As a final step, the model is deployed in live operation and
if update is needed, the whole process can be repeated to build a new and up-to-date model. [9]

2.2.3 K-Means

K-Means is a machine learning algorithm that groups data into different clusters. The algorithm takes a set of data consisting of rows where each row represents a point of equal dimensions. The points are used to produce a given number of clusters of which the data will be divided between [1]. Figure 3 show a K-Means clustering with three clusters of the popular Iris dataset [10].

![Figure 3: K-Means clustering on the Iris dataset.](image)

In the original K-Means algorithm the centroids are initialized at random points taken from the data. The algorithm will then run in iterations containing two steps. In the first step each instance is assigned to its closest centroid. Second, the centroids’ positions are recalculated based on the center of all instances assigned to each cluster. The iterations occur until a given number of iterations is reached, or when the centroids no longer changes after iteration. [1]

K-Means uses a distance measurement to calculate distance between instances and centroids. The default distance measurement for K-Means is Euclidian distance [1]. The Euclidian distance between two points, $d(p, q)$, of $D$ dimensions is calculated as follows:

$$d(p, q) = \sqrt{\sum_{i=1}^{D} (q_i - p_i)^2}$$  \hspace{1cm} (1)
There are some drawbacks in the original K-means algorithm. The worst case time complexity is exponential and it does not guarantee good clusters since the cluster centers are computed locally. Hence the distances between centroids are not taken in consideration. An extended version of K-Means was developed to remove the drawbacks, called K-Means++. The difference is that K-Means++ only initializes the first centroid at random, while all other centroids are calculated sequentially. The centroids will be calculated one by one, where each centroid is biased on the previously selected centroids to give them a better spread. [1]

The time complexity of K-Means++ is $O(nkd)$ of $n$ instances, $k$ clusters and $d$ dimensions. The original k-means algorithm has a time complexity of $O(nkdi)$ where $i$ is the number of iterations needed for convergence. K-Means can easily be parallelized since each initial centroid is generated at random and every other centroid can be computed individually. K-Means++ on the other hand uses sequential calculations which makes it impossible to parallelize. [1]

An additional version of K-Means have been developed by Bahmani et al. [11] to get the benefits of K-Means++ and at the same time be able to scale and take advantage of parallelism. This version is called K-Means|| and was designed to be parallelized. The algorithm works as follows. Let $X=\{x_1,\ldots, x_n\}$ be a set of $n$ points of $d$ dimensions and let $C=\{c_1, \ldots, c_k\}$ be a set of $k$ cluster centers, also known as centroids. A centroid $c_i$ defines a clustering of $X$ such that the points in cluster $c_i$ are closer to $c_i$ than $c_j$ where $j \neq i$. The centroid of subset $Y$, $Y \subseteq X$, is defined as the mean of all points in $Y$.

$$\text{Centroid}(Y) = \frac{1}{|Y|} \sum_{y \in Y} y$$  \hspace{1cm} (2)

Big Data analytics for the forest industry - A proof-of-concept built on cloud technologies
David Sellén

Background and related work

2016-07-14

Listing 1: K-Means||\((k, \ell)\), initialization algorithm.

\[
C \leftarrow \text{Initial center chosen at uniform random from } X \\
\psi \leftarrow \phi x(C) \\
\text{for } \log(\psi) \text{ iterations do} \\
\quad C' \leftarrow \text{sample each point } x \in X \text{ with probability } p_x = \frac{\ell \cdot d^2(x, C)}{\phi x(C)} \\
\quad C \leftarrow C \cup C' \\
\text{end for} \\
\text{for } x \in C \text{ do} \\
\quad \omega_x \leftarrow \text{The weight of } x \text{ calculated from the number of points in } X \text{ closer to } x \text{ than any other point in } C \\
\text{end for} \\
\text{Recluster the weighted points in } C \text{ into } k \text{ clusters}
\]

- \(k\), the number of clusters
- \(\ell\), oversampling factor
- \(C\), a set of centroids
- \(\psi\), the initial cost of the clustering
- \(X\), set of points
- \(\omega_x\), weight of a point \(x\)

K-Means|| uses an oversampling factor \(\ell\) and calculates an initial computation cost \(\psi\) of clustering the initial centroid. Points of \(X\) are sampled individually and put in \(C\) with a certain probability \(p_x\). This means that the selected \(C\) points can be more than \(k\) and therefore needs to reduce to the size of \(k\). The number of centroids is reduced by setting weights to the points in \(C\) and then \(C\) is re-clustered with the weighted points into \(k\) clusters. [11]

2.3 Apache Hadoop

A technology that has been developed to meet the demands of Big Data is Apache Hadoop [12]. Yahoo! [13] and Facebook [14], among others, have successfully used Hadoop to handle several Petabytes (10^{18} bytes) of data. Hadoop is an open-source development framework for building scalable applications running on computer clusters with commodity hardware [13]. The framework consists of two parts, a distributed file system and a distributed computation module. The file system is called the Hadoop Distributed File System (HDFS) and is an open-source implementation inspired by Google File System [15]. The computation framework is known as Hadoop MapReduce and is an open-source implementation based on Google MapReduce [16]. Computer clusters also need a resource manager that manages hardware resources and
schedules jobs inside the cluster. Yet Another Resource Negotiator (YARN) is the resource manager that comes with the Hadoop Framework. A system running on Hadoop can scale linear up to thousands computers where each computer provides storage, bandwidth and computation capacity to the cluster [13].

2.3.1 Distributed file system

HDFS uses a hierarchical namespace of directories and files that is similar to common local file systems. When a file is written to a cluster running HDFS it will be handled by two types of nodes, NameNode and DataNode. Figure 4 shows the architecture of HDFS [13]. Each cluster has a dedicated NameNode responsible for managing file metadata and multiple DataNodes for storing client data as blocks. A NameNode stores metadata related to files such as file permission and file modification time. It is also responsible for knowing where files are stored in the DataNodes. DataNodes store application data as blocks, a block is replicated to multiple DataNodes to provide availability and reliability. [13]

![Figure 4: HDFS architecture.](image)

When a client wants to read from or write to HDFS storage, it will first contact the NameNode [13]. The NameNode will reply with a list of DataNodes, which stores the data block or which the data should be written to.

2.3.2 MapReduce

Hadoop MapReduce is a parallel data processing model that allows distributed processing of large data. Data stored in a Hadoop computer cluster is partitioned and distributed among the machines. This enables
parallel MapReduce jobs to be executed close to the data. A MapReduce process consists of a map function that takes the distributed partitions of data and creates an intermediate set of <key, value>-pairs that is passed to the reduce function, all values with the same key are merged together such that each key will possibly have a smaller number of values. An example could be to count the occurrence of all words in a number of documents where the result would be pairs of words as keys and the occurrence of each word the value <word, occurrence>. [13]

A disadvantage of Hadoop MapReduce is that it was not developed for iterative algorithms [17] such as machine learning algorithms which are important in this study. Hadoop MapReduce has also been criticized for not being very development friendly and for having high latency for simple queries. These disadvantages has led to the development of in-memory processing platforms for building applications to run on computer clusters for fast and iterative computations on Big Data such as Apache Spark [17], Apache Flink [18] or Microsoft Prajna [19].

2.4 Apache Spark

Apache Spark is a framework for processing large datasets on computer clusters. It has been open-sourced since 2010 and is maintained by the Apache Software foundation [20]. It was developed by Berkley University of California as they found the MapReduce framework insufficient for interactive analysis of data and iterative machine learning algorithms [17]. Spark offers a general-purpose application programming interface (API) to develop applications that take advantage of the distributed memory in a computer cluster. By storing computed results in memory, Spark can process data in iterations much faster than MapReduce which would store results between iterations to disk [21].

2.4.1 Spark cluster

When developing an application with Spark, a driver program is created by a developer. The driver program contains a SparkContext which is used to access the Spark execution environment. The SparkContext connects to a Spark Master which is also known as a resource manager. The master node prepares worker nodes with executors which executes tasks. An illustration of a Spark cluster can be seen in Figure 5. When performing an operation on a dataset the master node prepare executors on the worker nodes and then the driver program sends tasks to the
worker nodes. The workers will read input data from stable storage or its memory cache depending on whether or not the data has been persisted to memory. [22]

Spark can also run in a local mode with CPU cores on the local machine, acting as a Spark executor and the local file system is used instead of a distributed storage [22].

2.4.2 Resilient Distributed Dataset
Spark relies on its data interface called resilient distributed datasets (RDD) which are fault-tolerant and distributed data types. RDDs enable development of applications to perform fast in-memory computations and data reuse in computer clusters. Either data from stable storage (i.e. Hadoop) or from other existing RDDs can be used to create a new RDD. An RDD is read-only hence a new RDD must be created through transformations to perform changes. The content of an RDD is distributed partitioned records that can be transformed into a new RDD using transformation such as map, reduce, join and filter. In addition, RDDs provide efficient fault-tolerance by not requiring data replications over the network; instead lost data can be reconstructed based on logs of previous data transformations. Different operations on RDDs can run in parallel, such as foreach, collect and reduce. The foreach-operator makes each record in the RDD pass through a user defined
function (UDF), collect retrieves all data from a distributed RDD and reduce will merge records together and retrieve a result. [21]

2.4.3 Programming model
Spark runs on the Java Virtual Machine (JVM) and offers an application programming interface (API) for Java, Scala and Python. Spark itself is written in Scala [21]. Listing 2 shows a code example written in Scala on how to use Spark to count the number of error messages in log files.

**Listing 2:** Scala example using Spark for counting error messages.

```scala
val conf = new SparkConf().setAppName("Application")
val sc = new SparkContext(conf)
val lines = sc.textFile("hdfs://…")
val errors = lines.filter(_.startsWith("ERROR"))
errors.persist()
errors.count()
```

In the example above, a Spark context is initialized and used to load text files containing logged data. The data is loaded from HDFS storage into an RDD called lines. Each line of all log files are filtered into a new RDD called errors if the line begins with the text “ERROR”. This new RDD is persisted into the computer cluster’s memory and then the number of lines is counted and the result is sent to the driver.

2.5 Spark’s component stack
Apache Spark comes with its own resource manager (Standalone scheduler) and a set of modules that run on top of the Spark Core, see Figure 6. Spark is compatible with different resource managers such as YARN and Mesos but they are not required with Spark’s own but simpler standalone scheduler [22]. Lots of different data storages are supported, from local storage to distributed storages such as HDFS, HBase and S3. The different modules for Spark are SparkSQL, Spark Streaming, MLlib, GraphX and Spark R. The modules used in this study are SparkSQL and MLlib which are further described in the following chapters.
2.5.1 SparkSQL

SparkSQL is a module that enables ETL and makes it possible to write SQL queries on semistructured data. The framework aims to provide a simpler and higher-level interface than MapReduce by combining relational and procedural approaches. SparkSQL comes with a relational query optimizer called Catalyst which automatically optimizes queries for increased performance. With Catalyst, SparkSQL can be extended to support new data sources such as raw JSON files or XML. In addition to Catalyst, SparkSQL offers an API called DataFrame. With DataFrames, developers can perform relational operations like `select`, `groupby` and `where` while also supporting procedural operations such as `map` and `foreach`. A DataFrame can be compared to a table in a relational database and has a schema describing its structure of items and their data type. Supported data types are for example string, Boolean, double date, and timestamp. DataFrames are interoperable with Spark’s RDDs. Operations performed on a DataFrame are not computed until an output operation is called, such as `collect` or `count`. [23]

SparkSQL has native support for reading directly from JSON-files and can be extended through Catalyst to support other formats like CSV (comma separated value) or XML (extendable markup language) files. There are also supports for Java Database Connectivity (JDBC) to connect to different databases such as MySQL or PostgreSQL. Furthermore, SparkSQL can be queried from different distributed querying tools such as Apache Hive. [23]
Queries can be written in two different ways, either as SQL queries in plain text or by using the DataFrame interface. Listing 3 contains an example written in Scala using SQL to query JSON data containing name and age of people. A SparkSQL context is used to read a JSON-file from HDFS into a DataFrame. The DataFrame is then registered as a table called people. The table person is queried to get the name of persons older than 30 and finally the name of these people are shown. No operations are performed on the data until the show-command is called. When this command is called the query is optimized trough Catalyst and then the optimized query is executed.

Listing 3: SparkSQL query using SQL.

```scala
def df = sqlContext.read("hdfs://data/people.json")
df.registerTempTable("people")
val results = sqlContext.sql("SELECT name FROM " + "people WHERE age > 30")
people.show()
```

The same query could also be written using the DataFrame API, shown in Listing 4 below.

Listing 4: SparkSQL query usign the DataFrame API.

```scala
def df = sqlContext.read.json("hdfs://data/people.json")
val people = df.select("name").filter(df("age") > 30)
people.show()
```

The results from a DataFrame can be stored directly into a distributed database such as HBase and HDFS or to the local file system [23]. The DataFrame API has native support for writing into the parquet file format. The parquet file format [24] is developed to work with the Hadoop ecosystem and is a column storage file type based on Google Dremel [25]. It is designed to be efficiently compressed while at the same time applying a scheme to the data.

2.5.2 MLlib

In 2012, Spark’s machine learning library called MLlib was developed at the UC Berkley AMPLab and was open sources by the end of 2013. MLlib is built on top of Spark and takes advantages of Spark’s ability to
scale and perform fast iterative computations. The library lets developers build machine learning pipelines together with Spark and SparkSQL. It makes use of the DataFrame API and comes with a set of scalable data processing and machine learning algorithms. Supported machine learning algorithms includes decision trees, linear, clustering algorithms, association mining and ensemble learning. One of the clustering algorithms implemented in MLlib is K-Means|| described in Chapter 2.2.3. MLlib also supports an ML model standard called Predictive Model Markup language (PMML). This standard makes it possible to train and share ML models between different analytical applications. [26]

2.6 Evaluation of performance scalability

System scalability can be measured in different ways. This chapter describes how hardware resources affect the scalability of distributed and parallel computation systems.

A system is assumed to be scalable if the performance of the system is improved in proportion to an increase of hardware resources. For systems using computer clusters there are mainly two ways to add hardware resource for increased performance. This is known as horizontal and vertical scaling. Horizontal scaling or scale-up refers to how the performance of a system is changed when adding additional nodes to a computer cluster. In vertical scaling (scale-out), the hardware resources on the existing machines in the computer cluster are increased. Better CPU, more RAM or additional hard drives may be added to the machines. [27]

In parallel computing, performance can be evaluated by measuring a systems speedup strong and weak scalability which depends on the number of CPU cores. When measuring a system’s strong scalability the number of CPU cores are increased while the size of the problem stays the same. Strong scalability measures a systems speedup. In weak scaling the problem increases along with the number of CPU cores. The idea of weak scaling is to evaluate if the system can solve larger problems in the same time. [28]
2.7 **Standard in forest communication**

The StanForD 2010 standard [3] arrived in 2011 and is a successor of the earlier version of the original StanForD standard from the late 1980’s. It is the de-facto standard used globally by cut-to-length (CTL) harvest machines. CTL is a harvesting method where harvesting and forwarding machines are working together to harvest a forest. The harvest machines perform a chain of events where they fell, delimb and cut a stem into logs of specified lengths by the stump area [29]. First the machine takes its gripping arm around the tree while using a chain saw cutting tool to fell the tree. After the tree has been felled it is turned horizontally and pushed through a set of knives that cuts the branches from the tree. When the tree has been cleaned from branches to a specified length the machine uses its chain-saw head to cut a log from the stem. After a stem has been cut into logs the top of the tree is cut off.

StanForD is continuously being developed and administrated by Skogforsk, the Forestry Research Institute of Sweden. It has been created through cooperation between different stakeholders of the forest industry and is supported by the leading machine manufactures such as John Deer and Komatsu together with control system manufacturers such as Dasa Control Systems. StanForD 2010 exists in different versions and contains fourteen types of messages shown in Table 1. The messages are in XML format and have associated XML schemas describing their structure. [3]

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>.pin</td>
<td>Product instruction</td>
<td>Control</td>
</tr>
<tr>
<td>.oin</td>
<td>Object instruction</td>
<td>Control</td>
</tr>
<tr>
<td>.spi</td>
<td>Species group instruction</td>
<td>Control</td>
</tr>
<tr>
<td>.ogi</td>
<td>Object geographical instruction</td>
<td>Control</td>
</tr>
<tr>
<td>.foi</td>
<td>Forwarding instruction</td>
<td>Control</td>
</tr>
<tr>
<td>.fdi</td>
<td>Forwarding delivery instruction</td>
<td>Control</td>
</tr>
<tr>
<td>.udi</td>
<td>User-defined data instruction</td>
<td>Control</td>
</tr>
<tr>
<td>.hpr</td>
<td>Harvest production</td>
<td>Production reporting</td>
</tr>
<tr>
<td>.thp</td>
<td>Total harvest production</td>
<td>Production reporting</td>
</tr>
<tr>
<td>.fpr</td>
<td>Forwarded production</td>
<td>Production reporting</td>
</tr>
<tr>
<td>.ogr</td>
<td>Object geographical report</td>
<td>Production reporting</td>
</tr>
</tbody>
</table>
The messages are divided into four different categories, Control, Production reporting, Quality assurance and Operational monitoring. Control messages contain instructions for how harvesters and forwarders should manage products and provide information about lengths, product identities, prices and so on. Production reporting messages are used to report detailed information about harvested stems and information about forwarded stems. The purpose of quality assurance messages is to ensure high accuracy when harvesters measure stem weight or the length to cut logs of a stem. The operational monitoring messages store information about the utilization of the machine such as time period of harvesting, breaks, repair and planning.

2.7.1 Harvest production reports

During the process of felling and cutting trees, harvesters measure the width, length and volume of every stem and log. These details are stored in the HPR message [30]. A simplified illustration of the HPR messages structure is shown in Figure 7.
The root element is called *HarvestProduction* and contains a *HarvestProductionHeader* with message creation information [30]. The root element also contains *Machine* objects, which in turn stores all the specific harvest production information such as tree species group definition and stems. The forest harvesters are equipped with a tool on its arm that cuts the trees; this tool can sometimes cut a bunch of trees in one cut, this is known as multi-tree handled stems. They can be logged as either a single processed stem or a multi-tree processed stem. Either way, the element stores information about each log that has been cut of a specific stem. In short, the data that can be found in the HPR files are presented in Table 2.

<table>
<thead>
<tr>
<th><strong>Table 2: Data stored in HPR files.</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data for harvested stems</strong></td>
</tr>
<tr>
<td>Stem ID</td>
</tr>
<tr>
<td>Wood species</td>
</tr>
<tr>
<td>GPS-position</td>
</tr>
</tbody>
</table>
Big Data analytics for the forest industry -
A proof-of-concept built on cloud technologies
David Sellén

2.8 Related work

2.8.1 Harvest production reports

In a study [31] done in Australia in 2015 the HPR data were used to produce productivity models for forest harvesting. The result was a model with equivalent accuracy as the harvester productivity models produced by manual time and motion studies. This study relied on three HPR files with about 200 stems in each. The HPR data was filtered manually using Microsoft Excel.

Another study related to the HPR data [32] was conducted by researchers at Skogforsk where they tested different machines running the StanForD 2010 standard before the practical implementation of the standard in the autumn of 2013. The goal of the study was to evaluate the implementation of StanForD 2010 in different harvest machines and especially the harvest production report (HPR). All machines did not use all features in the StanForD 2010 standard but all machine supported the HPR. When the harvest machines calculated the log volumes the differences between the machine and the manually measured volume differed by 0.07 to 0.18 percentage, which was considered satisfying. Furthermore, the study shows that the harvesting machines can generate highly accurate data. A limitation is that they only used machines that are correctly calibrated in their tests.

A system that heavily relies on the HPR data is hprCM, a harvest production calculation module described in [33]. The hprCM uses the HPR files and it requires that the harvesters have GPS-trackers for position of each individual stem (which is the machines position during harvest). The system has been used to calculate harvested bio fuel (i.e. tops, branches, slash and stumps). It also allows reliable predictions of how much bio fuel will be provided from a final felling. The system is extended in [34] with the possibility of generating information about the remaining stocks after thinning and final felling and to automatically transmit harvest production and GPS data to forest registers and plan-
ning systems. The HPR data is used to reconstruct information on whole trees. This is done by combining the dimensions of logs cut from a stem with the use of the algorithms described in [35]. The reconstruction of complete tree data is done to be able to reassemble information about the harvested forest to enable further analysis. The system also implements different filters to sort out erroneous data and trees that do not fit certain given preferences. Other features of the system are the ability to calculate the slash and stumps in an area, visualize the GPS position of harvested stems on a map, and visualize and estimate the harvested hectare.

2.8.2 Big data analysis tools for science

Machine learning has shown usefulness in different research areas and can help researchers in their scientific process, for example to generate hypotheses from large data volumes [8]. Landset et al. [36] mention that the traditional machine learning tools such as WEKA and R were not designed to handle data of multiple terabytes very efficiently. This has led to the development of technologies for analytical processing of Big Data which have been used in different research fields, for example biology [37] and astronomy [38]. Nothaft et al. [39] point out that traditional distributed storage and processing frameworks for scientific use have not been optimized for the application of machine learning techniques or user-defined functions on semistructured data. Trying to meet these needs has led to a large amount of different open source solutions for Big Data analytics, for example Apache Spark.

Spark has previously been used in scientific applications to process large amounts of data [38] and for analytics with Spark’s native modules, and SparkSQL MLlib [40], [41]. Another analysis tool called VariantSpark [42] is an application created with Spark for clustering genomic information with K-Means.1
3 Methodology

This chapter presents an overview of the research method, motivations of selected methods and development tools. It also describes how the results were evaluated and includes a discussion of ethical aspects related to this study.

3.1 Research method

The research methodology known as Design Science (DS) has been applied in this study. DS is used when developing something to serve a human purpose [43]. In comparison, traditional natural science is used when trying to understand reality. DS contains both a qualitative and quantitative parts, from defining the objectives to evaluating and measuring the results. Ken Peffers et al. have developed a specific process model for the application of DS in information systems (IS) research [44]. Their model consists of six activities: problem identification, definition of objectives, design and development, demonstration, evaluation and communication of the results. The implementations of these activities are described below in the order which they were carried out.

1. Problem identification and motivation: The research problem was identified and motivated based on the collected secondary data. The collection of the secondary data is described in the following chapter, Chapter 3.2. The results from this activity are presented in this report as a part of the introduction, theory and related work.

2. Definition of objectives: The objectives were derived from the idea of making the proof-of-concept a foundation of a fully working analysis system for the forest industry. The objectives are based on the results from activity 1 and have been defined as a set of research questions to be answered. These questions together with the scope of this study are defined in Chapter 1.

3. Design and development: Based on the research questions which were a result of the second activity, an artifact was created. An artifact is a term used within DS for the solution that will meet the objectives [44]. In this study the artifact is the proof-of-concept of an analysis system for
analyzing HPR data. The requirements specification for the system was created based on the results from previous activities. The design and implementation of the system was based on the requirement specification. The specification was allowed to be altered to make the development process agile for unpredicted barriers that could appear. Hence requirements could be added or removed during the implementation phase.

4. Demonstration: The results from the design and implementation activity are described in Chapter 4 and will serve as a demonstration of the system.

5. Evaluation: The horizontal and vertical scalability of the system were evaluated by running the system with different computer resources. The tests are described in Chapter 3.4. A month of real HPR data is used during the evaluation; this dataset is described in more detail in Chapter 3.4.2.

6. Communication: The results from this study are communicated to stakeholders and published online. The software code is open-source and uploaded to an online repository on GitHub. Publishing the code makes it possible for the public to contribute to the development of the system.

3.2 Data collection

Both primary and secondary data have been collected. Primary data have been gathered from performance and scalability tests. Secondary data were collected from academic journal articles, conference papers, books, white papers, work reports as well as digital reports and other material from the Internet. The search for scientific papers was conducted through Google Scholar and Primo, Mid Sweden University’s digital library that is connected to a wide range of publishers of scientific papers. Other scientific papers were found using specific publisher’s digital libraries, such as IEEE Xplore or ACM Digital Library. Solutions for Big Data analytics applied in other research areas were studied to find a successful architecture design. Previous forestry research and data processing on HPR data has been reviewed for information on how it could be used.
3.3 Implementation

The following sections describe the methods and tools used in the development the proof-of-concept for the analysis of HPR data.

3.3.1 The harvest production reports

The HPR data was selected as the forest data to be analyzed to show how data from the StanForD 2010 standard could be implemented into the system. Support of HPR data was considered to provide a good foundation for forest analysis. The system stores the raw HPR in its native XML format since some data that might not be useful today may prove useful in the future and could enable more analyzes. A drawback of storing the data in its native format is that it will not be storage efficient. The HPR data could also be connected with other information, some examples are:

- StanForD2010 FPR files with the forwarding machines data
- Transportation data from trucks transporting the timber
- Wood quality from SDC’s measuring stations
- Weather data from the same coordinates as the stem that has been harvested, forwarded or transported.

Future possible analysis and additional data sources that could be beneficial are further discussed in Chapter 7.2.

3.3.2 K-Means method

Clustering algorithms can be used to cluster data into groups and then the data in each group can be further analysed to see why the data were grouped into these clusters. K-Means is a widely used clustering algorithm and has been listed as one of the top 10 data mining algorithms [45]. This clustering algorithm was selected as the ML algorithm to implement as an example into this proof-of-concept. Some of the reasons for the selection of K-Means as the algorithm to implement are:

- K-Means works well with large datasets which is required by the system.
- The HPR data contains a high degree of numerical information.
- It can be used to explore data by creating different clusters and analyzing the differences in the data of these clusters.
3.3.3 System architecture

The design selected for the system is a client-server architecture consisting of a back-end server written in Java 8 and a front-end web application. Using Java, the application can run on all popular operating systems making the system platform independent. By creating a web application to act as the client user interface the system can be used from any type of electrical device that can run a web browser. The system has been built on open-source frameworks to make it possible for anyone to use or continue to develop without any software costs.

All computations and data processing is controlled by the back-end (server) whereas the front-end (client) works as the user interface of the system. The back-end performs all the heavy computations and can take advantage of the use of a computer cluster for Big Data storage and heavy computations. If running the system with a computer cluster it can either be connected to a self-hosted data cluster or a cloud service. This design makes the system very adjustable for specific needs and available resources. Clients will not have to worry about upgrading their hardware since all the heavy work is done at the server or computer cluster. A single computer cluster is used for both storage and computation which reduces complexity, cost and maintenance.

The architecture style Representational State Transfer (REST) was selected since it makes the system easily extendable with new functionality and can be integrated with different kinds of front-ends. For example, the system can communicate with a website as well as a native desktop application. This makes the back- and front-end loosely coupled which makes it possible to create different front-ends that can communicate with the same back-end system.

The front-end is implemented with HTML, CSS and JavaScript as a single-page web application (SPA). The SPA shows how a modern client application could be designed to interact with the back-end system. A SPA does not require the whole web page to be completely loaded when moving around subpages. Instead when a user interacts with the web page this will send a POST or GET request to the back-end and only updating parts of the page when receiving a response.
3.3.4 Back-end frameworks

The first step in developing this proof-of-concept application was to select what open source frameworks to use for storage and processing of Big Data. Important factors when choosing the frameworks were that they had been widely adopted, offer extendibility to the system and have big community support. To keep track of all framework dependencies in the Java project the build automation tool Maven has been used. The frameworks used in building the back-end are listed below and an illustration of how they are connected is shown in Figure 8.

- Apache Hadoop v. 2.7.2
- Apache Spark v. 1.6.1
- SparkSQL v. 1.6.1
- Spark-XML v. 0.3.3
- MLlib v. 1.6.1
- Spring Boot v. 1.2.0 with embedded Tomcat v. 7.0.52

The back-end is developed with HDFS and Spark. By using Spark and Hadoop it becomes possible to choose how the system should get its resources. The system could either run altogether on a single machine or with its own computer cluster. It is also possible to connect the system to rented computer resources from a cloud service such as Amazon AWS or Google Cloud Platform. This makes it possible to select an option depending on user requirements, which could be both economic and power related.

Apache Hadoop

The use of Hadoop HDFS is optional; if the system runs on a desktop computer then the local file system could be used for storage instead. However, HDFS is preferred for storage of raw HPR files since it is scalable and new machines could be added to share the workload. The
Hadoop framework is widely used to store Big Data and lots of different clouds provide Hadoop support. Instead of using Hadoop MapReduce the parallel processing framework Spark has been selected for its ability to process machine learning algorithms.

**Spark and modules**

Apache Spark was selected as the computation framework for the implementation because of its performance and large community support. It comes with a complete ecosystem of related technologies to develop machine learning pipelines and to perform interactive Big Data analytics. The performance of Spark is supposed to be 100 times faster than Hadoop MapReduce when running in-memory, and 10 times faster when running from disk [20]. Spark also offers a package for running R programs [46], which is a programming language widely used by researchers. Furthermore, Spark supports ETL on raw data in various formats and from databases and distributed storage like HDFS through SparkSQL.

SparkSQL’s built-in query optimizer Catalyst makes it possible to extend the system to support more than the natively supported data sources. An example is Spark-XML [47], a third-party module for SparkSQL that makes it possible to run SQL queries on XML-files. A small performance test was conducted to identify differences in using SparkSQL with Spark-XML compared to plain Spark and RDDs. The test was to count the number of stems in a set of HPR files. The results from the tests can be found in Appendix A, and they showed only a minor performance difference in favor for plain Spark and RDDs. However, SparkSQL and Spark-XML was chosen since the performance difference insignificant and SparkSQL offers rich data manipulation and extraction features. SparkSQL with Spark-XML can read and automatically identify the data types of elements in XML-files from StanForD 2010. SparkSQL also makes it possible to read and write DataFrames into Parquet files. The Parquet file format was selected to store datasets since it is designed for distributed storages, efficient compressing of the data and makes it fast to search.

Spark supports the machine learning libraries MLlib, Mahout and H20; these are the most comprehensive and best performing Big Data machine learning libraries according to a recent comparison [48]. MLlib
was selected out of these three machine learning libraries since it offers preprocessing features as well as many common algorithms and highest performance [32].

Spark version 1.6.0 was used as the targeted Spark version in the beginning of the implementation phase but this was later changed to version 1.6.1. The main reason for changing the Spark version was that Spark 1.6.0 contained a bug [49] that did not allow NULL values in DataFrame arrays. Furthermore, the release of 1.6.1 occurred after the implementation process had started.

**Spring Boot**

When developing software it is most useful to follow general solutions that have proven to solve commonly occurring problems. Spring is a framework that makes developers follow certain structures and to use well-known design patterns. It resembles Java Enterprise Edition (JavaEE) with enabling annotations and auto-binding or data injections.

Spring Boot [50] is part of the Spring ecosystem and enables production-ready and stand-alone Spring applications. The framework is designed to make it easy for developers to get started with development. It comes with embedded Java application servers such as Tomcat and handles a lot of configurations automatically. Spring also simplifies the implementation of REST interfaces. In the development of the proof-of-concept, Spring Boot was considered to make development simpler and the code cleaner. It would also reduce the effort of later deployment in a production environment since it is designed to run on Java application servers.

### 3.3.5 Front-end frameworks

As the client is implemented as a web front-end this opens up for an endless number of web frameworks that could be used. The frameworks selected for this implementation are all open-source projects written in JavaScript and have been well used in web development projects. A list of the frameworks used in building the front-end is shown below and an illustration of how they are connected is sown in Figure 9.
AngularJS

Google began the development of an open-source web application framework in 2010 called Angular [51]. The framework is created to simplify the implementation of SPAs and has grown into one of the most popular web application frameworks. Angular extends HTML with custom tags, extending HTML with more features. There are different modules that can add additional features to Angular such as Session for web sessions and Route to allow unique URLs to different page views. This framework is used to create the logic for the SPA client. Angular also handles REST calls and is used to enable the client to communicate with the REST API in the back-end.

User interface frameworks

Bootstrap [52] is an open-source User Interface (UI) framework for adding responsive UI design to webpages. Responsive design means that the interface of the web application will change depending on screen size and makes it possible to write a single UI that will work on all kinds of devices.

Another open-source JavaScript library is LeafletJS [53] which makes it easy to add light-weight interactive maps to the web application. The library comes with lots of different features and has been used to visualize the geographical positions of HPR data. The map tiles can be loaded from different providers for example OpenStreetMap [54] which offers free tiles that have been used in this implementation.
A CSS pre-processor called LESS [55] has been used to reduce time spent styling the SPA and to make it simpler to change the CSS of the user interface.

### 3.4 System tests and evaluation

This chapter describes the environment on which tests have been performed as well as the tests themselves and how the system has been evaluated.

#### 3.4.1 Test Environment

A test environment was created with 5 virtual machines (VMs) to emulate a computer cluster for testing and evaluating the implementation. The amount of VMs was limited to 5, so that the hardware on the host machine would be sufficient during the performance and scalability tests. The VMs ran on the same host while both back-end and the client ran on a laptop connected to the same wired local area network. The machines were created using Vagrant [56] and ran on a server using VirtualBox [57]. Vagrant is a software for setting up and managing virtual development environments using machine virtualization software such as VirtualBox. The hardware specifications of the server machine are shown in Table 3.

**Table 3: Hardware specifications of the server machine.**

<table>
<thead>
<tr>
<th>Server hardware specifications</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>2x Intel Xeon E5-2630 v3 2.4GHz, 20M Cache, 8 C / 16 T</td>
</tr>
<tr>
<td>RAM</td>
<td>64 GB, 2133MT/s, RDIMM, 4x Data Width</td>
</tr>
<tr>
<td>OS</td>
<td>CentOS 7, 64-bit</td>
</tr>
</tbody>
</table>

The hardware specifications of the VMs are shown in Table 4. The VMs used a total of 18 CPU cores out of the 32 available on the host through hyper-threading. All cores of the host could not be used since they were reserved for other tasks not related to this project.

**Table 4: Hardware specification of the virtual machines.**

<table>
<thead>
<tr>
<th>Virtual machine hardware specifications</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU cores</td>
<td>4</td>
</tr>
<tr>
<td>RAM</td>
<td>4096 GB</td>
</tr>
<tr>
<td>OS</td>
<td>CentOS 7, 64-bit</td>
</tr>
</tbody>
</table>
The software installed on the VMs were Java Runtime Environment (JRE) 8, Apache Hadoop 2.7.2 and Apache Spark 1.6.1. The selected cluster architecture consisted of a single node serving as both Spark Master and HDFS NameNode while the remaining four nodes were HDFS DataNodes and Spark Slaves. The cluster architecture is illustrated in Figure 10.

![Virtual machines diagram](image)

**Figure 10: Computer cluster of virtual machines.**

The cluster resource manager used was Spark Standalone Scheduler, since it has a small overhead and simplified deployment on the cluster. The system might perform differently in a cluster running another resource manager such as Mesos or YARN. The Spark cluster settings used in the tests can be found in Appendix B.

### 3.4.2 Test data

The data that has been used in the tests are 19.2 GB of StanForD 2010 HPR data of different HPR versions. The data consists of 6388 HPR files with a file size ranging from 5 kB to 113 MB. The files were collected by SDC during December 2015. The data contain null values and the number of stems in a file can range from zero up to thousands of stems. Furthermore, it is possible that not all harvested stems might had stem coordinates, a name for the harvest area or the same name for the same type of wood species.

The naming of the files can contain characters such as blank spaces and dots. This could create problems, for example, if there are dots in the file name, which is not allowed by HDFS. To solve the naming problem the files were simply renamed before being loaded into the HDFS cluster.
3.4.3 Test cases and gathering of primary data

The system performance has been tested in terms of horizontal and vertical scalability. To evaluate the horizontal scalability the system has processed the same amount of data and same calculations using one to four nodes. The vertical scalability was performed in combination with strong scaling by using four nodes but changing the number of execution CPU cores from one to four. The aim of the tests was to show how the system performance scales depending on hardware resources.

All tests ran the same scenario where some of the main features of the system are used together with HPR data. The scenario followed the steps in Listing 5 below. All steps have measured execution time in nanoseconds from the Spark driver application running on the laptop.

**Listing 5: Steps in the test scenario.**

1. Filter and cache all data \( D \) in the worker nodes RAM.
2. Extract and aggregate information about all worksites into \( w \) from the filtered data \( D \). Cache \( w \).
3. Collect the worksites information \( w \) as JSON.
4. Create a dataset \( d \) from \( D \) by extracting stems length, volume as m3sub, BDH and wood species.
5. Train a K-Means|| model \( m \) using the data in \( d \) except for wood species.
6. Test \( m \) by clustering the instances in \( d \).

\( D \), all initial data
\( w \), information of worksites
\( d \), a created dataset
\( m \), K-Means|| clustering model

The first step loads and filters all available HPR data into the Spark workers RAM. In the following step the worksites information is extracted, e.g. the worksite coordinates and the number of stems harvested from each worksite. The worksite information is collected in JSON format from the worker nodes in step 3. In step 4 a new dataset is created where the targeted stem data have been length, m3sub volume, BDH and wood species. The species column was used as a label and was not used when building or testing the model in step 5 and 6. The dataset was not split, meaning the full dataset was used both to train and test the model.
Each test was executed in cold runs, meaning the whole system was rebooted between each new test. The reason for this is that the execution time was shown to be reduced in each of the first iterations. This was assumed to be because of a built-in caching in Spark. An example of this can be seen in Appendix A. By running the tests as cold runs we would get the worst-case execution times. Hence, testing was time consuming. Each step was performed 8 times in every iteration and the mean value of these execution times was used.

To make the tests reproducible and comparable the same settings were used in all tests. When running K-Means|| the parameters was set as follows; the number of $k$ clusters was set to 3, iterations to 20 and an initial random seed set to 12345. For other settings the default values were used as defined in MLlib’s K-Means JavaDoc [58]. The Spark driver memory was set to 4 GB and Spark executors’ memory was set to 3.7 GB each.

**Horizontal scalability test**

Horizontal scalability tests were performed to identify how the system behaves when adding or removing computer nodes in the cluster. By identifying the behaviour, users of the system can get an idea of what hardware resources are required for analyses of different sizes. During this test all nodes were given 2 CPU cores each.

A set of raw data including 1000 HPR files with a size of around 500 MB was used in the tests. The data was loaded into the Hadoop cluster with four DataNodes and one NameNode using a replication factor of 3. The test ran in four iterations where one node was shut down by the end of each iteration. After removing one node, Hadoop’s balance script was executed to rebalance the data on the restoring nodes.

**Vertical scalability test**

The vertical scalability was measured on the performance in relation to the number of CPU cores of each working node. The test ran in four iterations where the number of cores used was reduced from 16 down to 4 cores as a total number of cores or the computer cluster. In this test all available HPR data from one month was used. This means that 19.2 GB of 6388 HPR files were processed in each iteration. Testing the computation time when analyzing a month of HPR data was also part of proving
the feasibility of the system by performing realistic analyzes within a reasonable computation time.

3.4.4 Data analysis and system evaluation

The system will be evaluated on its imposed requirements and the results from the performance and scalability tests. The test results from altering the number of CPU cores and nodes have been analyzed on the system performance scale. The results from the two different tests were compared as to which of the changes in the computer cluster would have the biggest performance boost on the system. The execution times of each event in the test scenario have been analyzed based on what the reason can be behind the differences.

3.5 Validity and reliability

Reliability of the test results is increased through publishing the source code, making it open for anyone to inspect and reproduce the tests. The test cases are well described as well as the test environment. With respect to the sensitivity of the content in the HPR data it cannot be published together with the source code. Instead, the data have been described regarding size, characteristics and the number of files. Since the HPR files are part of the open StanForD 2010 standard the description of the HPR data is published by Skogforsk online. Furthermore, the parameters used for K-Means are shown and the random seed parameters are static to simplify reproduction of the tests. The different tests are performed multiple times to achieve higher confidence in the results.

Validity is increased by clearly describing all parts included in the study. The tests are described as to why they are performed and what they will measure. Furthermore, the data that have been used in development and testing have been provided by SDC and the data are HPR data from real harvests. The HPR data as part of the StanForD 2010 makes the system work as a more general solution that could be used in different countries. The data used was collected in December 2015 just before the start of this study to have the most up-to-date data. Hence, as the data was collected during one month it might not be representative for all months of the year.
4 Implementation

This chapter presents an overview of the system as well as detailed technical descriptions of system components.

4.1 System overview

The proof-of-concept has been designed to show that a set of basic features could be successfully implemented with the selected development frameworks. In short, the targeted features are to load and filter raw HPR data, visualize worksites information, create and save a dataset out of the raw data, load existing datasets, apply ML to datasets and visualize the results.

The implementation of the proof-of-concept has targeted a specific analysis of the application of K-Means to information on stems. The stem data that is possible to extract are the numeric features length, volume, DBH (dimension over breast height) and the nominal feature wood species to use as data label. This example analysis could demonstrate if there is a relation between stem dimensions and species based on all harvested stems. The analysis is a dummy analysis implemented as a concrete example to show the possibilities and feasibility of the system.

The system has been designed as a server-client architecture using a REST API to communicate between clients and the server. The server application, or back-end, is responsible for the system logics and also storage and computation for which it can be connected to a computer cluster to scale. The front-end serve as presentation layer and is an SPA responsible for user-interactions and visualization. A detailed list of requirements on the system components can be found in Appendix C.

Figure 11 illustrates how the components in the system relate to each other. There are two ways the system can be deployed, either by using a computer cluster running Spark and Hadoop or on a single desktop computer. These will hereby be referred to as distributed and local mode, respectively.
The functionalities implemented in the proof-of-concept allow users to:

- View the worksites and the number of harvested trees from that worksite. This is presented both in a list and on a map. Both the map and list can be used to select which worksite data that should be extracted.

- Create a dataset of stem data base on a limited amount of pre-defined attributes: length, m3fub volume, DBH and species. When a dataset is created it is also saved as a Parquet file.

- Load a dataset from a list of saved datasets. When a dataset is loaded it shows the number of instances and dataset features.

- Run K-Means clustering on a loaded dataset. The system takes the following parameters: dataset split ratio of train/test and the split random seed, nr of clusters, iterations and an optional label attribute.
• Presentation of clustering results. Pie charts showing the number and percentage of instances with the same label in each cluster as well as the total number of instances.

4.1.1 Packets and files structure
The code base contains both the back-end and front-end source code. The Java files are kept in the src/Java/edu.miun.firespark packet folder. The Java dependencies are defined in Maven’s Project Object Model stored in the pom.xml file placed in the root of the project folder.

- Java/
  - edu.miun.firespark/
    - etl/
    - machine_learning/
    - messages/
    - rest/
    - utilities/
    - application.java
    - main.java
- resources/
  - public/
    - css/
    - js/
    - index.html
  - application.properties
- pom.xml

Figure 12: Project files and packet structure.

System settings and configurations
The application name, running mode, cluster connection information and data source paths can be changed in a configuration file named application.properties in the resource folder of the Java project. System settings are currently changed by modifying the application.properties file before compiling the application.

Spark configurations are set in the SparkContextBean.java file which represents the Spark context (Spark Driver) as a singleton object. Singleton objects can only have one instance which ensures that there cannot
be more than a single Spark context. If there were more than one this would cause errors.

**Java packets**

The back-ends Java packet domain is called *edu.miun.firespark* and consists of five different packets divided by their responsibilities, see Table 5. A class diagram of all Java classes is found in Appendix D.

**Table 5: Java project’s packet structure.**

<table>
<thead>
<tr>
<th>Packet name</th>
<th>Responsibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>etl</td>
<td>This package is responsible for the ETL process of extracting and combining data from raw data such as HPR files.</td>
</tr>
<tr>
<td>machine_learning</td>
<td>ML algorithms, management of dataset and data preparation for ML.</td>
</tr>
<tr>
<td>messages</td>
<td>Message structure for communication through the REST API.</td>
</tr>
<tr>
<td>rest</td>
<td>The REST API that allows users to interact with the system.</td>
</tr>
<tr>
<td>utilities</td>
<td>Miscellaneous classes such as a Time measurement class and Spark context singleton class.</td>
</tr>
</tbody>
</table>

**Web resources**

The front-ends SPA web resources are divided into two folders, *js* and *css*. The *js*-folder contains the JavaScript dependencies and most importantly the *app.js* file where the logic of the SPA is defined. The *css*-folder stores the CSS files required by the JavaScript frameworks and a LESS file that is used to style the SPA. In the roof of the web resource folder you will find *index.html*, which contains the template of the SPA.

**4.2 Data storage**

Depending on whether the system is running in local or distributed mode, the system will load data either from the local file system or from a computer cluster running HDFS. In both modes, the data have been grouped in the same way consisting of two folders: an HPR-folder for the storage of raw HPR data and a dataset-folder for stored datasets in the Parquet file format. There are also data schemas that, in both different modes, are stored at the same machine running the back-end appli-
cation. These schemas are stored in JSON file format in a folder called `schema`. Using the JSON-formats makes it possible to change the schema manually if needed. Currently the only required schema is for the HPR data.

4.3 Back-end

This chapter describes the boot process of the back-end application and the REST API.

4.3.1 System boot

When the system boots, see Figure 13, the Spar Driver will be created and the embedded Tomcat application server will start and host the web front-end. The front-end can be reached from port 8080 on the IP address of the machine running the back-end. The system will then send the Spark application settings and JAR-dependencies to the Spark cluster. The JAR-dependencies are Spark-XML and also the application itself. Next, the system loads all available HPR files from the HPR-folder, filter it and cache the results into the workers memory. The loaded data contains worksite names, single processed stems with coordinates and species group definition information. This cached data is then used to extract information of all available worksites which will also be cached into memory.

![System boot process diagram](image.png)

Figure 13: System boot process.
When loading the data into a DataFrame, Spark will either create a schema that fits the data or run with a user-defined schema. This is used to make it possible to save schemas that are created by Spark. These are stored as a JSON-file that could be manually changed if needed. If a new data source is added to the system, this requires that a new repository class is created and a related schema file. A schema can either be manually provided or automatically created by the system. When the system is started in distributed mode it can either be connected to a self-hosted data cluster or a cloud service. This design makes the system very adjustable to specific user needs and available resources. After the system has booted it is idle until a client connects and calls a method in the REST API.

4.3.2 Analysis of HPR data

When a dataset containing stem species is analysed it is pre-processed as a test of the application of pre-processing to a dataset. This is done without the user’s knowledge for testing purposes. To identify a stems species in the HPR data we need to link each Stem element with a SpeciesGroupDefinition. Each Stem element in the HPR file contains a SpeciesGroupKey, which is related to a SpeciesGroupDefinition element, which in turn contains the SpeciesGroupName. Since the SpeciesGroupName can be different for the same species, for example “Pine” or “Pine 1”, this needs to be pre-processed before applying ML. In this implementation the species names are replaced by a static set of pre-defined names. The predefined wood species names are fir, pine, contorta, deciduous, birch and unknown if the species is not recognized.

4.3.3 REST API

The REST methods that can be called from the API are created using Spring Controller annotations. The API consists of three controller classes presented in Table 6 below. Each method is called through a Uniform Resource Identifier (URI) together with a HTTP method, either POST or GET.

<table>
<thead>
<tr>
<th>WorksiteController</th>
<th>URI</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET /worksites/listAll</td>
<td>Return a list of all worksites with</td>
<td></td>
</tr>
</tbody>
</table>
implementations, this proof-of-concept built on cloud technologies is a pioneering effort in the forest industry.

### Implementation

<table>
<thead>
<tr>
<th>DatasetController</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST /dataset/create/{ETLQuery}</td>
<td>Receive an ETLQuery which is used to create a dataset. If the dataset is successfully created, it is set as the current working dataset for the user session and stored as a Parquet file in the datasets folder.</td>
</tr>
<tr>
<td>POST /dataset/load/{name}</td>
<td>Receive a simple String of the dataset to be loaded. If the dataset exists in the dataset folder, the Parquet file is loaded and cached in the workers memory.</td>
</tr>
<tr>
<td>GET /dataset/list</td>
<td>Return a list of the names of all available datasets.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MachineLearningController</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST /ml/run/{MLQuery}</td>
<td>Receive a MLQuery with a selected algorithm and ML parameters which will be executed on the client sessions current working dataset.</td>
</tr>
</tbody>
</table>

### 4.4 Front-end

The front-end is divided into two pages, create and analyze. The SPA is designed using Bootstrap for header, columns, lists and buttons. All logics and REST calls to the back-end are created using Angular. The create-page shown in Figure 14 displays the worksites information and lets users create datasets. The map is created using Leaflet and controlled through the Angular application.

On client connect the client will send a GET-request to “/worksites/all” which returns an array of JSON objects containing information on all worksites. When the worksites are loaded into the client, the user can create a dataset by selecting the worksites of interest and pressing “Create”. This will open a popup where the user selects how the data
should be preprocessed. These preferences are sent to the backend and the backend will construct a query to send to the computer cluster.

**Figure 14: View of worksites.**

A user can select which harvest worksites that should be included when creating a dataset. This allows the analysis to be limited to certain geographical areas. All dots on a map represent a harvest worksite, when a worksite is selected it turns red otherwise it is green, see Figure 15. The list and map are linked together so that both the list and map can be used when selecting a set of worksites.
In the analysis-page, shown in Figure 16, a user can load a dataset and apply ML algorithms as well as view the ML results. A dataset can be loaded from the leftmost column and when it has been cached at the working nodes the back-end will reply with the dataset information which will be showed in the upper leftmost column. The dataset information that is displayed is its name, number of instances and columns. The middle column contains a form where a user can set algorithm and parameters to apply on the loaded dataset. The input elements in the form are dynamic, i.e. changing depending on which algorithm is selected. Currently the only algorithm that can be selected is K-Means clustering but another option has been added to the list to display how the form changes dynamically when selecting an algorithm.
An example of how results are shown is presented in Figure 17. In the example, each cluster is displayed with the number of instances it contains and a pie chart. The pie chart is created using C3 and shows the percentage of instances with a certain label. The system can easily be extended to show the clustering results in a scatter chart instead, which would make a better visualization of the cluster results. Scatter plot as visualization was not implemented since it would cause the whole dataset with all instances to be sent from the back-end to the client. Hence this was assumed to be too demanding on the clients. How to solve heavy visualizations on the client side should be investigated in further development.
Figure 17: Clustering results.
5 Results

This chapter presents the results from the scalability and performance tests. The execution time is showed in graphs numbered from 1-7 related to the steps in the test scenario showed in Listing 5. Observations on CPU usage of the nodes showed a 100% usage on the working nodes during computations.

5.1 Horizontal scalability test results

The graphs in Figure 18 to Figure 23 shows the results from tests running 1000 HPR files on a computer cluster while altering the number of identical computer nodes from one to four. The scalability factors of each event are shown in Table 7.

Table 7: Horizontal scalability factors.

<table>
<thead>
<tr>
<th>Event</th>
<th>1 node</th>
<th>2 nodes</th>
<th>3 nodes</th>
<th>4 nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>All data</td>
<td>1</td>
<td>2.829726</td>
<td>5.3</td>
<td>7.209559</td>
</tr>
<tr>
<td>Worksites</td>
<td>1</td>
<td>1.263889</td>
<td>1.491803</td>
<td>1.82</td>
</tr>
<tr>
<td>Create</td>
<td>1</td>
<td>1.542105</td>
<td>2.362903</td>
<td>3.367816</td>
</tr>
<tr>
<td>Save</td>
<td>1</td>
<td>2.086538</td>
<td>4.018519</td>
<td>7.75</td>
</tr>
<tr>
<td>Train</td>
<td>1</td>
<td>1.111111</td>
<td>1.428571</td>
<td>1.666667</td>
</tr>
<tr>
<td>Test</td>
<td>1</td>
<td>1.160562</td>
<td>1.47125</td>
<td>1.547416</td>
</tr>
</tbody>
</table>

Figure 18: Load of HPR data with different number of nodes.
Big Data analytics for the forest industry - A proof-of-concept built on cloud technologies

Results
2016-07-14
David Sellén

---

**Extract worksites information**

![Graph](image1.png)

*Figure 19: Extract worksites information with different number of nodes.*

**Create dataset**

![Graph](image2.png)

*Figure 20: Create a new dataset with different number of nodes.*

**Save dataset**

![Graph](image3.png)

*Figure 21: Save a dataset with different number of nodes.*
The results from the horizontal scalability tests indicate that the system performance scales depends on the number of nodes in the computer cluster. Loading and saving data to/from disk (Figure 18 and Figure 21) shows an increased performance of more than 700% from running on a single node up to four nodes. The remaining events had a performance increase of 150–300%.

5.2 Vertical scalability test results

The graphs in Figure 24 to Figure 29 shows the results from tests running one month of HPR data on a computer cluster with four identical nodes while altering the number of CPU cores per node from one to four. The scalability factors of each event are shown in Table 8.
Table 8: Vertical scalability factors.

<table>
<thead>
<tr>
<th>CPUs</th>
<th>1 CPU</th>
<th>2 CPUs</th>
<th>3 CPUs</th>
<th>4 CPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>All data</td>
<td>1.523589707</td>
<td>1.650784581</td>
<td>1.727389344</td>
<td></td>
</tr>
<tr>
<td>Creat</td>
<td>1.196233317</td>
<td>1.305353621</td>
<td>1.322913177</td>
<td></td>
</tr>
<tr>
<td>Save</td>
<td>1.146106973</td>
<td>1.184344096</td>
<td>1.483086204</td>
<td></td>
</tr>
<tr>
<td>Load HPR</td>
<td>1</td>
<td>1.138001288</td>
<td>1.012226154</td>
<td></td>
</tr>
<tr>
<td>Save</td>
<td>1.095172142</td>
<td>1.661486719</td>
<td>2.322507701</td>
<td></td>
</tr>
<tr>
<td>Test</td>
<td>1.609094853</td>
<td>1.679929828</td>
<td>1.819167204</td>
<td></td>
</tr>
</tbody>
</table>

Figure 24: Load of HPR data using different numbers of CPU cores.

Figure 25: Extract worksites information using different numbers of CPU cores.
Results

Figure 26: Create new dataset using different numbers of CPU cores.

Figure 27: Save a dataset using different numbers of CPU cores.

Figure 28: Build a K-Means model using different numbers of CPU cores.
The different events had an increased performance of around 113% to 230%. Saving a dataset shows to perform the highest using two CPU cores and had a reduced performance using more than two cores. Training a K-Means model and testing it on the dataset showed the largest performance increase when adding more CPU cores.
6 Analysis

This chapter contains an analysis of the proof-of-concept including the system architecture, functionality, development frameworks and test results.

6.1 The implementation

The system can handle storage, processing and machine learning on gigabytes of HPR files. It is capable of extracting information from raw HPR data into datasets and support a machine learning pipeline with pre-processing and K-Means clustering.

6.1.1 Architecture

The system architecture has made it possible for the system to scale from running on a single desktop computer to a computer cluster of multiple computers. The three layers, the computer cluster, the back-end and front-end, increases the maintainability. The REST interface from the back-end makes it possible to have multiple user interfaces that could be designed for different purposes. This could for example make it possible to have one monitoring web page with live forest information and a desktop application for advanced analytics connected to the same system. The front-end which in this project has been implemented as an SPA, shows the possibilities of creating lightweight interface for remote interaction with large amounts of data.

The developed system is flexible in terms of how it can be used. It can either run on a single computer or on a server connected to a computer cluster and clients. The back-end also works with any kind of front-end that can use its REST API.

6.1.2 Development frameworks

The proof-of-concept has showed the feasibility of using the selected development frameworks in developing an analysis system for big amounts of forest data. The popularity of the selected tools showed very helpful since issues encountered during development were most often already discussed and solved in various development forums. Hence, there were no issues related to the individual frameworks that could not be solved.
The development frameworks used for the development of the back-end was Spark, SparkSQL with Spark-XML, MLlib Hadoop and Spring Boot. Apache Spark has made it possible to run the system in both local and distributed mode. The cooperativity of Spark and the two modules, SparkSQL and MLlib, has made the source code easily manageable. SparkSQL and MLlib have also made it possible to implement a complete analytical pipeline with data extraction, pre-processing and machine learning. MLlib was successfully used to perform K-Means|| on data directly from DataFrames.

Spark-XML had no simple way of constructing a DataFrame schema based on StanForD 2010’s XML-schemas (XSDs). Instead Spark-XML could be used to generate its own schema based on the HPR files that was loaded into the system. This did not work as well as expected, this is probably because of the different HPR versions of the HPR files that were used. A solution to this problem could be to parse the HPR XML schemas for each HPR version into individual DataFrame schemas. There were no issues related SparkSQL and the Parquet file format. The datasets stored in Parquet was saved and loaded by Spark without the need of a schema. The Parquet files contain descriptions of their contents which was automatically created by spark during the save process.

Hadoop HDFS creates a reliable and available storage for the HPR files and datasets. Adding and removing HDFS nodes during the scalability test showed how easy it was to rebalance the data between nodes and how easily more storage could be achieved by adding new DataNodes. An issue that encountered was that HDFS could not handle the file names of some HPR files since they contained illegal characters. Since data ingestion not was part of the scope of this proof-of-concept the files were simply renamed. But, if a data ingestion layer is added to the system the naming of the HPR files should be handled there. Furthermore, as the NameNode keeps all file metadata in memory a lots of small HPR files can lead to a high memory requirement on the NameNode.

The embedded application server in Spring Boot made it very easy to run the system and deploy the SPA. Spring Boot also made it easy to develop and control the REST API where client queries automatically were consumed as plain old Java objects (POJOs).
The front-end was successfully designed as a SPA resembling a desktop application. The SPA was built using Angular, Bootstrap, LeafLet and C3. Angular controls the logic of the SPA and the contact with the REST API.

6.1.3 **System extendibility**

It should be possible to add multiple data sources such as more StanForD 2010 data files and join these together by harvest order IDs, geographical location or time and date. Other data should be added for more interesting analysis, such as weather data which affects the wood moisture level after harvest. The implementation makes use of Java Interface classes to make it easier to extend the system with new algorithms and other data sources.

The worksites could be visualized through the location of each harvested stem from the HPR data similar to what has been done in [34]. The system can be extended with pre-defined features for processing data. One example is to reconstruct information on whole trees out of the stem data using the formulas from [35].

Spark is compatible with different distributed database systems such as HBase and Cassandra, so that different storages could be combined. The Spark platform also supports streaming data and streaming data mining. SparkSQL could make it possible to create a UI where users could write ad-hoc SQL queries directly on the raw data. Spark could also be extended with the SparkR module to make it possible for users to execute code written in R on the computer cluster. It is possible to add more of the features that come with MLLib for preprocessing, data transformation and different ML algorithms. The system could also be extended using MLLib to support the PMML file format for loading or exporting ML models. If MLLib lacks features, the system can be extended with other ML frameworks such as H2O and Mahout.

REST API in the back-end makes it possible to retrieve results from the system and export to for example MATLAB or to develop other user interfaces, for example mobile applications.
6.2 Test results

The horizontal scalability test results from Chapter 5.1 shows that the system performance scales depending on nodes in the computer cluster. With an increased number of computer nodes, all tested features showed a significant increase of performance. Most notable are the performance boosts on the two steps that perform disk I/O, loading all data from disk and saving a dataset to disk. These two events had an increased performance to more than 700% from running on a single node up to four nodes while the other events had around 150-300%. This shows that disk I/O is a bottleneck of the system which can be reduced making more nodes share the responsibility of writing to disk.

The tests for vertical scalability in Chapter 5.2 showed a slight performance increase overall. The different events had an increased performance of 113% to 230%. The biggest performance boost was achieved when building and testing the K-Means model. This increase was even larger than for the horizontal scalability test. It was not expected that the number of CPU cores would have such a low impact on the other events. One of the tested events that stand out is when saving a dataset. The execution time for saving a dataset is increased when using two CPU cores instead of one, but when the number of cores is further increased to three and four the performance is turning even slower than with a single core. This test was done additional times to ensure that there were no mistakes. One possible reason could be that two threads on a machine cannot write in parallel to the same file hence an increased amount of working threads on the same machine will not make it faster.

The overall results from the tests show that the system is reliable for the support of heavier workloads and it can be anticipated to be capable of handling data of very large quantities. To achieve the biggest improvements on performance the system should be given additional computer nodes instead of better hardware.
7 Conclusions

The development tools that were used in the development have successfully been put together into a working proof-of-concept. The system can store, extract and visualises information on harvest worksites from gigabytes of raw HPR files. Datasets with information of stems can be created, pre-processed and used in clustering analysis.

The design of the system makes it possible to run the system on a desktop computer or in a more distributed mode using a server and computer cluster. The computer cluster and front-end loosely connected to the back-end which makes it easy to change into computer cluster or create different front-ends. The system has been implemented using Java interfaces to simplifying extension of new data sources and ML algorithms. It also allows new data, in various formats, to be connected using SQL queries. The REST API can easily be extended with more methods using just a few lines of code.

The most time-consuming features of the system are those who filter and extract data, such as filter and cache all raw HPR data into the system and to create a dataset. These ETL processes do however scale well when adding additional nodes to the computer cluster. Hence, the system scales horizontally rather than vertically regarding CPU cores. The disk I/O is suspected to be the bottleneck; hence more nodes will have a larger impact on the performance.

The study has provided new knowledge to the area of system development for Big Data analytics and forest research through the following:

- Previous research and work related to HPR data and interactive Big Data analysis have been reviewed and presented.
- A proof-of-concept system has been developed that allows data mining on large amounts of HPR data stored in its native format.
- A system architecture and selection of frameworks have been proposed that allows the system to be expanded with a variety of functions that could further enhance analysis of forest data.
• Ideas on how to perform ETL on HPR data and how to visualize HPR data have been discussed.

• Performance and scalability tests on real HPR data have showed the feasibility of the system.

7.1 Ethical considerations
The anonymity of the people and companies whose information can be found in the HPR data has been kept during this study by not publishing any of the actual content from the HPR data.

The HPR files contain information that could be sensitive for some if it was published or leaked. In the HPRs, information on which harvest contractor performed the harvesting and the harvest worksite can be found. This could be used to calculate how much the forest owner earned from a harvest.

A finished version of this proof-of-concept system could be used inappropriately, i.e. not related to science. For example, the system could be used to gather information about a specific forest owner or contractor for information that is not related to science. Hence, limitations or specific access restrictions should be considered in a final version.

The patterns that could be found using this system are supposed to be beneficial to the forest industry. The results could increase the wood quality from harvests and reduce the required harvests and in turn reduce the environmental impacts of forest harvests. However, the patterns found could also increase harvests or make specific geographical areas more attractive for growing and harvesting forest.

7.2 Suggestions for further research
Analyzing big amounts of HPRs might show valuable patterns, but there are other forest related data that could be combined with the HPRs to enable more complex analyzes. Future work should involve investigating what other data sources could be added to the system in order to perform more interesting analyze. Data that could be interesting to combine with the HPR data are for example weather, wood quality, logs temporary storage and log transportation.
This study has provided a working proof-of-concept but with limited functionality. Additional ML algorithms and functionality should be added to the system and preferably developed specifically for forest related data. In addition to clustering algorithms it could be interesting to implement association algorithms to find associated features in forest data. Some additional features that have been implemented in [33] could also be implemented in this system. Such as pre-processing stem data to reconstruct stems into information about complete trees and visualization of individual stems in a map view.

The code has been published online on GitHub as an open-source project for further development.

**Project:** ForestAnalytics  
**Programming language:** Java  
**Dependencies:** Java 8 and Hadoop Common 2.2  
**Licence:** Apache 2.0  
**Code repository:** http://github.com/David-Sellen/ForestAnalytics
References


Big Data analytics for the forest industry - A proof-of-concept built on cloud technologies

David Sellén

References

2016-07-14


References


2015.


[34] Johan J. Möller, John Arlinger, Andreas Barth, Nazmul Bhuiyan, and Björn Hannrup, "A system for calculation and feedback of harvester based information to forestry planning systems," Skogforsk, no. Nr. 756, 2011.


Big Data analytics for the forest industry - A proof-of-concept built on cloud technologies

David Sellén

References


August 2015, pp. 1-8.


Appendix A: Comparison of SparkSQL and Spark

Performance comparison of counting stems

Counting stems in HPR files

Two possible ways of counting stems in the HPR files were tested. The first test was performed using Spark’s MapReduce and in the second test the stems were counted using SparkSQL together with a third-party plugin for querying XML-files called Spark-XML. The tests were conducted using 76 HPR files from the local Windows file system, with a size of 37.5 MB and containing information about 6747 stems.

The average time was calculated as follows:

\[
\text{Average time} = \frac{\text{elapsed time}}{\text{number of runs}}
\]

The elapsed time is the time to perform the counting in 100 concurrent iterations and then dividing the time by 100. The time for each individual count task was taken from Spark’s job history data.

Counting with Spark’s MapReduce

By counting the stems this way, all data is loaded into a RDD and then simply scanned for the number of occurrences’ of the string “<stem>” as it was a simple text file.

```java
JavaRDD<String> lines = 
    sparkContext.getContext().textFile(localDir+"/*");
long nrOfStems = lines.filter( s->
    s.contains("<StemKey>")).count();
```

Data: 37.5 MB
Stems: 6747
Average time from 100 runs: 230 ms

As shown in the diagram, the operation stabilizes after about 20 rounds at 100 ms.
Counting using SparkSQL and Spark-XML plugin

All HPR files are first loaded into a DataFrame called df which is used to create a DataFrame with only the Stems data which are stored into the cluster memory.

```scala
val conf = new SparkConf().setAppName("Application")
DataFrame df = sparkContext.getSQLContext().read()
 .format("com.databricks.spark.xml")
 .option("rowTag", "HarvestedProduction")
 .load(localDir+"/*");
DataFrame stems =df.select(
 Long nrOfStems = stems.count();
```

Data: 37.5 MB  
Stems: 6747  
Average execution time of 100 runs: 299 ms

The loading of all HPR files took 27 seconds and was not included in the measurement. The first count operation took 6 seconds and the graph shows that the operation stabilizes around 200 ms.
Appendix A:
Comparison of SparkSQL and Spark
2016-07-14
### Appendix B: Spark configurations

#### Runtime Information

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java Version</td>
<td>1.8.0_73 (Oracle Corporation)</td>
</tr>
<tr>
<td>Scala Version</td>
<td>version 2.10.5</td>
</tr>
</tbody>
</table>

#### Spark Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>spark.app.id</td>
<td>app-20160527071820-0003</td>
</tr>
<tr>
<td>spark.app.name</td>
<td>ForestAnalytics</td>
</tr>
<tr>
<td>spark.driver.host</td>
<td>192.168.1.232</td>
</tr>
<tr>
<td>spark.driver.maxResultSize</td>
<td>4000m</td>
</tr>
<tr>
<td>spark.driver.port</td>
<td>41417</td>
</tr>
<tr>
<td>spark.executor.id</td>
<td>driver</td>
</tr>
<tr>
<td>spark.executor.memory</td>
<td>3700m</td>
</tr>
<tr>
<td>spark.externalBlockStore.folderName</td>
<td>spark-0dbdf5f-de-0f41-48e0-8d1e-35f47275b4f6</td>
</tr>
<tr>
<td>spark.jars</td>
<td>file:/C:/Users/Dwe/.m2/repository/com/databricks/spark-xml_2.10/0.3.3/spark-xml_2.10-0.3.3.jar,file:/C:/Users/Dwe/IdeaProjects/FireSpark/target/firespark-app-1.0.jar</td>
</tr>
<tr>
<td>spark.local.ip</td>
<td>192.168.1.232</td>
</tr>
<tr>
<td>spark.master</td>
<td>spark://192.168.1.93:7077</td>
</tr>
<tr>
<td>spark.scheduler.mode</td>
<td>FIFO</td>
</tr>
</tbody>
</table>
## Appendix C: System requirements

### The back-end and front-end requirements

There are two types of requirements, functional requirements (F) that describes a feature and non-functional requirements (NF) that describe the behavior of the system. The first table below shows the requirements set on the back-end and the second table for the front-end.

<table>
<thead>
<tr>
<th>Back-end requirements</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Description</td>
</tr>
<tr>
<td>----</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>1</td>
<td>Can take advantage of a computer cluster for storage and computing as well as running stand-alone locally on a single computer.</td>
</tr>
<tr>
<td>2</td>
<td>Performing both storage and computations the same way, a single cluster to reduce the complexity and simplify maintainability.</td>
</tr>
<tr>
<td>3</td>
<td>Store files in its raw format to maintain compatibility with existing tools and keep all data for future use cases.</td>
</tr>
<tr>
<td>4</td>
<td>Extract, transform and load information from raw HPR data into datasets.</td>
</tr>
<tr>
<td>5</td>
<td>Handle missing values and other errors in HPR data.</td>
</tr>
<tr>
<td>6</td>
<td>Increased performance if adding more machines to the computer cluster.</td>
</tr>
<tr>
<td>7</td>
<td>Provide a REST API to enable different types of loosely connected front-ends.</td>
</tr>
<tr>
<td>8</td>
<td>Store datasets in the Parquet format to keep the data structured and at the same time reduce the storage size compared to XML.</td>
</tr>
<tr>
<td>9</td>
<td>Display worksites through its name, the amount of single processed stems and the coordinates of the specific worksite.</td>
</tr>
<tr>
<td>10</td>
<td>Enable a user to select worksites to ETL trees and worksites data into datasets.</td>
</tr>
<tr>
<td>11</td>
<td>Save and load datasets in the Parquet file format.</td>
</tr>
<tr>
<td>12</td>
<td>Allow pre-processing of datasets.</td>
</tr>
<tr>
<td>13</td>
<td>The system should be extendable with more features.</td>
</tr>
<tr>
<td>14</td>
<td>Performance shall scale horizontally.</td>
</tr>
</tbody>
</table>
### Front-end requirements

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>Let users extract, transform and load information from raw HPR data into a dataset.</td>
<td>F</td>
</tr>
<tr>
<td>16</td>
<td>Communicate as a client to the REST API offered by the back-end.</td>
<td>F</td>
</tr>
<tr>
<td>17</td>
<td>Display worksites through its name, the amount of single processed stems and the coordinates of the specific worksite.</td>
<td>F</td>
</tr>
<tr>
<td>18</td>
<td>Enable a user to select worksites to ETL trees and worksites data into datasets.</td>
<td>F</td>
</tr>
<tr>
<td>19</td>
<td>Save and load datasets.</td>
<td>F</td>
</tr>
<tr>
<td>20</td>
<td>Allow clustering with K-Means∥∥ on datasets.</td>
<td>F</td>
</tr>
<tr>
<td>21</td>
<td>A user can split the dataset into a train and test set.</td>
<td>F</td>
</tr>
<tr>
<td>22</td>
<td>A user can assign the number of clusters, the random seed and number of iterations K-Means should run.</td>
<td>F</td>
</tr>
<tr>
<td>23</td>
<td>Present K-Means∥∥ results.</td>
<td>F</td>
</tr>
</tbody>
</table>
Appendix D: Class diagram
The back-end Java class diagram