FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Frameworks for Distributed Big Data Processing: A Comparison in the Domain of Predictive Maintenance

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur/in

im Rahmen des Studiums

Business Informatics

eingereicht von

Rudolf Plettenberg
Matrikelnummer 01229086

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer: Assistant Prof. Mag.rer.soc.oec. Dr.rer.soc.oec. Manuel Wimmer
Mitwirkung: Projektass. Dipl.-Ing. Mag.rer.soc.oec. Dr.techn. Alexandra Mazak

Wien, 16.04.2018

(Unterschrift Verfasser/in) (Unterschrift Betreuer/in)

Erklärung zur Verfassung der Arbeit

Rudolf Plettenberg, Stubenring 2, 1010 Wien

„Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.“

Ort, Datum	Unterschrift

Abstract

Predictive maintenance is a novel approach for making maintenance decisions, lowering maintenance costs, increasing a plants capacity and production volume, and positively affecting environmental and employee safety. In predictive maintenance, condition data of machines is constantly collected and analysed to predict future machine failures. Due to the high volume, velocity, and variety of gathered data, Big Data analytic frameworks are necessary to provide the desired results. The performance of these frameworks highly influences the overall performance of a predictive maintenance system, raising the need for tools to measure it.

Benchmarks present such tools by defining general workloads for a system to measure its performance. Due to the wide popularity of Big Data analytics across industries, benchmarks for Big Data analytic frameworks are defined specifically for each domain. While there are currently many benchmarks available for other domains such as retail, social network, or search engines, there are none available for Big Data analytic frameworks in the application area of predictive maintenance.

This thesis introduces the predictive maintenance benchmark (PMB). The PMB is a benchmark aimed at measuring the performance of Big Data analytic frameworks in the field of predictive maintenance. The data model and workload of the PMB represent typical tasks encountered by a predictive maintenance system. The PMB is implemented in the two most popular Big Data analytic ecosystems Hadoop and Spark and show Spark outperforming Hadoop in almost every task. For evaluation, findings gathered during implementation and execution of the PMB are analysed. Furthermore, the PMB results are validated against other studies comparing Hadoop and Spark.

Kurzfassung

Predictive Maintenance (vorausschauende Wartung) ist ein neuer Ansatz für das Fällen von Wartungsentscheidungen und ermöglicht eine Senkung von Wartungskosten, eine Steigerung der Produktion, sowie eine Erhöhung der Sicherheit und des Umweltbewusstseins. Bei Predictive Maintenance werden permanent daten über den Zustand einer Maschine gesammelt und verwendet, um Vorhersagen über künftige Ausfälle zu treffen. Auf Grund des Volumens, der Geschwindigkeit, und der Vielfalt der gesammelten Daten werden für deren Analyse spezielle Software Frameworks aus dem Bereich der Big Data Analyse benötigt. Die Leistung dieser Frameworks ist maßgeblich für die Leistung des gesamten Predictive Maintenance Systems.

Benchmarks erlauben es die Leistung von Frameworks zu messen und dienen daher gleichzeitig als Basis dafür, diese zu vergleichen. Durch den branchenweiten Einsatz von Big Data Analyse, und die daraus resultierenden unterschiedlichen Einsatzgebiete, ist es wichtig die Frameworks innerhalb eines bestimmten Aufgabengebietes zu vergleichen. Zurzeit existieren solche Big Data Benchmarks für die Bereiche des Handels, der Sozialen Netzwerke, der Web Suche, sowie der Bioinformatik. Es gibt allerdings derzeit keinen Benchmark, der den Tätigkeitsumfeld von Predictive Maintenance abdeckt.

Die vorliegende Diplomarbeit stellt daher den Predictive Maintenance Benchmark (PMB) vor. Der PMB setzt sich zum Ziel, die Leistung von Big Data Analyse Frameworks an Hand von Aufgaben aus dem Bereich Predictive Maintenance zu testen. Das Datenmodell und das Arbeitsvolumen von PMB repräsentieren hierbei typische Aufgaben eines Predictive Maintenance Systems. Nach der Entwicklung des PMBs, wird er auf den zwei populären Big Data Frameworks Hadoop und Spark implementiert. Die Resultate der jeweiligen Implementationen dienen als Basis für den Leistungsvergleich zwischen Hadoop und Spark. Schlussendlich wird der PMB durch Erkenntnisse, die während der Planung, Implementierung und Analyse der Resultate gewonnen wurden evaluiert. Zusätzlich werden die Resultate des PMBs noch mit anderen Studien, die die Leistung von Hadoop und Spark vergleichen, validiert.

Table of Content

List of figures

List of Tables

1 Introduction

1.1 Motivation

The importance of maintenance has grown with the increasing complexity of production systems [1]. Efficient maintenance lowers costs, increases a plants capacity and production volume, and positively affects environmental and employee safety [2]. *Predictive maintenance* is a novel approach for making decisions on timing and substance of maintenance work. The basic concept behind predictive maintenance is to predict future failures based on past and current condition data of a machine [3]. An increasing number of machines, affordable sensors, and intensive research have led to an immense volume of available data, often known as Big Data [4].

Big Data is characterised by its huge *volume* of data, speed of creation (*velocity*), and *variety* in data formats [5]. This leads to the following significant challenges [6]:

- Big Data often contains inaccuracies, duplicates, or missing values. Data *cleaning* detects and removes these errors in order to raise overall data quality [7].
- Information comes from many different sources in various data formats and must be *transformed* to be analysed [6].
- The high volume of Big Data exceeds the capacity of common *storage* solutions and requires new approaches such as distributed databases, where data is spread among multiple devices [6].
- Volume and variety of Big Data present significant challenges during *analysis.* Fast and accurate processing of petabytes of data requires specialised soft- and hardware [6].

Thus, to cope with these challenges specialised software frameworks have been developed, utilizing not only one but a cluster of multiple computers to store and process Big Data [12]. Processing Big Data generally requires multiple frameworks to work together, in a so-called Big Data analytic ecosystem [13].

The Big Data analytic ecosystem plays a central role in a predictive maintenance system. Its performance affects how much information can be processed, effectively limiting the amount of monitored machines as well as the speed of the analysis. It depends on the performance of each individual framework as well as how well they cooperate. Thus, it is important to analyse and compare such frameworks to discover their strengths and weaknesses in the application area of predictive maintenance.

1.2 Problem Statement

Due to volume, variety, and velocity of Big Data, common data processing technologies cannot handle its analytics satisfactory [8]. Only advanced data mining and storage techniques enable storage, management, and analysis of Big Data [8]. This has led to the development of specialised software frameworks. In many cases, such as predictive maintenance, one framework is not enough to provide all desired functionalities [9]. Before assembling a Big Data analytic ecosystem, a decision on which frameworks should be used has to be made. There are many possible candidates with more than 70 contestants in the open-source world alone [10]. Due to their differences in technology and functionality, direct comparison is difficult and special tools such as benchmarks are necessary. Benchmarks define general workloads and measurement metrics, which are used to compare different software with each other [11]. Due to the various application areas of Big Data analytics (i.e. search engines, e-commerce, social networks, or predictive maintenance) Big Data benchmarks aim to generate application-specific workloads [12, 13]. While benchmarks for other areas exist [11, 14, 15], there are currently none available for predictive maintenance. This lack of benchmarks for Big Data frameworks in the field of predictive maintenance prevents focused comparisons in this application area.

1.3 Aim of the Work

Benchmarks are tools to compare different software frameworks with each other. In the area of Big Data, they generally define application specific tasks due to the different possibilities for Big Data analytics. Currently, there are no benchmarks available for predictive maintenance. This thesis introduces such a benchmark: The predictive maintenance benchmark (PMB). The PMB enables the comparison of Big Data analytic ecosystems in the area of predictive maintenance. This thesis answers following research question:

1. What are the requirements for a Big Data analytic ecosystem to perform predictive maintenance?
2. What are the requirements of a benchmark comparing Big Data analytic frameworks in the field of predictive maintenance?

3. Are there any benchmarks for testing the performance of Big Data analytic frameworks in relation to predictive maintenance?
4. Is it possible to test Big Data analytic frameworks using the developed PMB?
5. Using the PMB, how do the most popular Big Data analytic frameworks compare to each other?

1.4 Methodological Approach

1.4.1 Literature Research

A systematic literature search is performed to determine the theoretical background, current state of the art and the landscape of current Big Data analytic frameworks. It follows the methodology defined by Kitchenham at al. [16]: After identifying the research questions, search queries are formulated. These queries are then being used in search engines for data collection. Selecting relevant information is done through an iterative process: At first the titles are scanned for relevant topics, then the abstracts, and at last the full text is read. Each step filters out irrelevant literature. This process ensures a broad and efficient way for literature research [16].

1.4.2 Benchmark Development

The development of the PMB follows the methodology on designing Big Data benchmarks defined by Han and Lu [12]. Major technology-agnostic Big Data benchmarks such as BigBench [14] and BigDataBench [15] followed a similar approach. As shown in Figure 1, designing a benchmark can be divided into the 5 steps of planning, generating data, generating tests, execution, and analysis and evaluation [12]. During the *planning* step, the benchmark object, application domain, and evaluation metric are defined. The next two steps specify the data and workload of the benchmark. Both of them depend on the application domain and should represent common, real-life tasks. For *execution*, the benchmark is implemented and executed. As described below, the implementation of the benchmark follows the methodology of Hevner at al. [17]. During the last step of the methodology, the benchmarking result is analysed and evaluated.

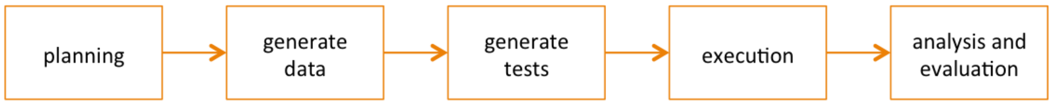


Figure 1: Benchmarking process as defined by Han and Lu [12].

1.4.3 Benchmark Implementation

For the individual implementations of the PMB on the most popular Big Data analytic frameworks, the thesis follows the methodology proposed by Hevner at al. [17]. Table 1 displays the seven guidelines for design-science research and their description. The bold text below specifies how the guideline will be followed during this master thesis.

Table 1: Design-science research guidelines as proposed by Hevner et al. [17, p. 83].

Guideline	***Description* / specification**
Design as an Artefact	*Design science research must produce a viable artefact in the form of a construct, a model, a method, or an instantiation.*
	The PMB is implemented on the most popular Big Data analytic frameworks and executed on a cluster of five Raspberry Pi's [1]. For each tested ecosystem, an artefact is created consisting of the code implementing the benchmark as well as the cluster-specific configuration of the frameworks within the ecosystem.
Problem Relevance	*The objective of design science research is to develop technology based solutions to important and relevant business problems*
	Implementing and executing a benchmark is important to evaluate its design. Observations made during implementation as well as analysing the benchmark results provide feedback on the validity of the benchmark.
Design Evaluation	*The utility, quality, and efficiency of a design artefact must be rigorously demonstrated via well-executed evaluation methods.*
	The implementation of the PMB is evaluated by comparing the measurement results to findings of similar benchmarks from other domains.
Research Contributions	*Effective design science research must provide clear and verifiable contributions in the areas of the design artefact, design functions, an/or design methodologies.*
	The PMB serves as a measurement tool to compare Big Data analytic frameworks in the domain of predictive maintenance. The implementation of the PMB serves during its evaluation.
Research Rigor	*Design science research relies upon the application of the rigorous methods in both the construction and evaluation of the design artefact.*
	Each step of the process is rigorously documented and follows before defined guidelines.
Design as a Search Process	*The research for an effective artefact requires utilizing available means to reach desired ends while satisfying laws in the problem environment.*
	The implementation of the PMB follows an iterative process. After each iteration, the current state is tested and evaluated.
Communicate Research	*Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences*
	The results of the research are presented within the proposed master thesis. The thesis provides a theoretical background, the specifications of the PMB, and the results of its implementation.

[1] https://www.raspberrypi.org/

1.5 Structure of the Work

Following the introduction, the second chapter covers the related work. This thesis proposes PMB, a new Big Data benchmark in the field of predictive maintenance. The related work therefore covers other benchmarks in the field of Big Data, separating them into technology-bound and technology-agnostic works.

The third chapter gives an overview of Big Data analytics and Big Data analytic ecosystems. The different elements of such ecosystems are explained in detail based on the six pillars model proposed by Khalifa et al. [18]. Building upon this, a general architecture of a Big Data analytic ecosystem is presented, which will be used to develop test-setups for the benchmark implementation.

In the fourth chapter a theoretical background to the field of predictive maintenance is offered. After an introduction in different maintenance techniques, all necessary steps from collecting data to making maintenance decisions are explained.

The fifth chapter presents the predictive maintenance Benchmark (PMB) following the methodology introduced in chapter 1.4.2. After determining the requirements and goals of the PMB, the data model and workloads are specified.

In the sixth chapter the evaluation of the PMB is shown. The benchmark is implemented on the currently most popular frameworks, which are portrayed in detail. The performance of the frameworks is measured and compared to each other. The results are used to evaluate if the PMB is viable.

Finally, conclusions as well as limitations of the work are discussed. Furthermore, possibilities for future work are presented.

2 Related Work

This thesis introduces the PMB, a new Big Data benchmark in the field of predictive maintenance. Related work therefore covers relevant research in the area of Big Data benchmarks. The increase in popularity of Big Data analytics led to the rapid development of new benchmarks by both academia and industry [11]. Existing Big Data benchmarks can be separated in technology-bound and technology-agnostic benchmarks [11].

2.1 Technology-bound Benchmarks

Technology-bound benchmarks are tied to specific Big Data analytic frameworks. They measure the performance of this framework across multiple hardware systems. Technology-bound benchmarks pursue different goals than PMB. While the PMB compares Big Data analytic frameworks with each other, technology-bound benchmarks compare the performance of one framework on different hardware setups. Below, the main contributions to technology-bound benchmarks are listed. They are grouped by the framework they are bound to.

HiBench, introduced by Huang et al. [19], and the MapReduce Benchmark Suite (MRBS) developed by Sangroya et al. [20] offer a variety of benchmarks for the Hadoop [2] MapReduce environment. The tests performed by HiBench include micro-benchmarks such as WordCount and Terasort as well as complex use cases from the domains of web search (page rank, nutch indexing), machine learning (Bayesian classification, K-Means clustering) and analytical queries (Hive [3] joins and aggregations). Similar to HiBench, PMBs workloads also include machine learning (classification) and analytical queries. However, they are neither bound to Hadoops MapReduce or Hive, nor are they utilized in the context of predictive maintenance. MRBS defines workloads such as a movie recommender, database queries, DNA sequencing, text processing and classification using the Naïve Bayes algorithm. In contrast to PMB, MRBS is tied to Hadoop and does not cover tasks in the area of predictive maintenance.

Li et al. [21] present SparkBench, a comprehensive benchmarks suite for the Spark ecosystem [4]. It offers workloads in four different areas: machine learning, graph computation, SQL queries and streaming applications. The workloads include logistic

[2] http://hadoop.apache.org/
[3] http://hive.apache.org/
[4] http://spark.apache.org/

regression, support vector machines, matrix factorization, pagerank, triangle count, hive SQL queries, twitter tags and page views as well as K-Means, linear regression, decision tree, and shortest path calculations. SparkBench measures execution time and data process rate (data size/execution time) as performance indicators. In contrast to PMB, Sparkbench is limited to the Spark ecosystem. Additionally, the workload of SparkBench covers a set of multiple generic tasks rather than focusing on a specific application area as PMB does.

Apart from the scientific community, manufacturers of Big Data frameworks provide benchmarks for their respective products. The Apache Software Foundation introduces GridMix[5], a benchmark for their Hadoop MapReduce environment. GridMix emulates different users sharing the same cluster resources submitting synthetic MapReduce jobs into the system. Different to other benchmarks, the workload is not predefined but modelled after an existing Hadoop system by analysing its job history. GridMix enables analysis on how an existing Hadoop system would perform on different hard- and software settings. In contrast to PMB, GridMix does not specify a fixed set of workloads but uses the workloads of an existing Hadoop cluster. Thus, GridMix can be used for any application area, provided a Hadoop cluster already exists. Many Big Data analytic frameworks offer examples that can be considered as micro benchmarks. Hadoop, for example, includes examples such as WordCount, Pi, Terasort and Grep[6]. WordCount calculates the amount of words in a provided text file. Pi estimates the digits of the mathematical constant Pi using a quasi-Monte Carlo method. Terasort sorts one terabyte of data and Grep counts the matches to a regex expression in an input file. Spark[7], Storm[8] and Flink[9] also include such examples. These examples have already been used in scientific papers as micro benchmarks to test Big Data analytic frameworks [22–26]. Micro benchmarks are fundamentally different to PMB. They are bound to a specific framework and only consider small and simple tasks. The PMB is framework independent and defines complex workloads in the domain of predictive maintenance.

[5] http://hadoop.apache.org/docs/r1.2.1/gridmix.html
[6] http://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html
[7] http://spark.apache.org/examples.html
[8] http://github.com/apache/storm/tree/v1.2.1/examples/storm-starter
[9] http://ci.apache.org/projects/flink/flink-docs-release-1.4/examples/

2.2 Technology-agnostic Benchmarks

Technology-agnostic benchmarks define general workloads without tying them to a specific framework. Their goal is to compare similar software rather than the same software across multiple hardware settings [11]. This section discusses the scientific effort in the development of Big Data benchmarks. The benchmarks are grouped by similar approaches.

The first group of benchmarks focuses on specific parts of a Big Data analytic ecosystem such as data storage or data processing. This focus presents the main difference to the PMB, which tests an entire Big Data analytic framework. Pavlo et al. [27] introduce a benchmark to compare the capabilities of different distributed databases, therefore focusing on data storage frameworks. The benchmark defines a series of 23 SQL-queries on a given data set and measures their execution time as performance indicator. The university of Berkley implemented a variation of Pavlov's benchmark, the AMPLab Benchmark[10]. The AMPLab benchmark builds upon the queries defined by Pavlo et al. [27] but uses a different data set. The university of Berkley implemented the benchmark for Redshift[11], Hive[12], Impala[13] and Stinger[14]. Ferrarons et al. [28] present a benchmark called Primeball, which focuses on data processing. They propose a fictitious news site hosted in the cloud to serve as a benchmark. A data set, queries and several use cases typical for news sites are defined to be implemented and measured. In addition to execution time, Primeball also takes costs of cloud services into account and integrates them into their performance matrix. In addition to the focus on single processing frameworks, Primeball differs from PMB in its application area. While Primeball consists of tasks typical to an online news site, PMB defines tasks based on predictive maintenance.

Like the PMB, the second group of benchmarks tests entire Big Data analytic ecosystem rather than its parts. However, while PMB focuses on tasks typical for predictive maintenance, the benchmarks below are based on tasks from other application areas. Ghazal et al. [14] presents the end-to-end Big Data benchmark BigBench, which is based on a product retailer model. BigBench covers ten business cases: Cross-selling, customer micro-segmentation, sentiment analysis, analysing user shopping experience, assortment

[10] http://amplab.cs.berkeley.edu/benchmark/
[11] http://aws.amazon.com/
[12] http://hive.apache.org/
[13] http://impala.apache.org/
[14] http://de.hortonworks.com/blog/stinger-next-enterprise-sql-hadoop-scale-apache-hive/

optimization, pricing optimization, store performance analysis, return analysis, inventory management and price comparison. The tests include handling structured and unstructured data as well as different data set sizes. Since its initial proposal, BigBench was continuously improved and in 2017 standardized by Cao et al. [29] of the Transaction Processing Performance Council (TPC) under the name TPCx-BB. Gao et al. [15] introduce the Big Data benchmark BigDataBench, which offers a wide variety of workloads. It does not focus on one business model but takes data sets from various domains and defines related workloads. In particular, BigDataBench consists of 15 data sets and 40 workloads coming from the domains of search engine, social networks, e-commerce, multimedia analytics and bioinformatics.

Many of today's popular Big Data benchmarks do not originate in scientific work but are defined by institutions. Since they are often considered as industry standards, the main non-scientific benchmarks are mentioned below. In contrast to the PMB they focus on testing specific areas of a Big Data analytic ecosystem. The main institutes defining Big Data benchmarks are the Transaction Processing Performance Council (TPC)[15], Standard Performance Evaluation Corporation (SPEC)[16] and the Storage Performance Council (SPC)[17] [11]. The TPC-H benchmark (Ad-hoc decision support benchmark) [30] is aimed at testing data warehouse frameworks and consists of 22 business queries simulating companies involved in managing, selling, and distributing products. The performance of the data warehouse is given by the execution time of the individual queries. The TPC-H benchmark is available for the Big Data querying systems Hive[18] and Pig[19]. The SPEC SFS 2014 benchmark [31] measures the maximum sustainable throughput of file systems. Its workloads consist of creating, reading and removing directories and files of various sizes. The performance of the file system is defined by the execution time of the workloads. The SPC-1 and SPC-2 benchmarks [32, 33] are targeted at comparing the performance of different storage systems. SPC-1 defines various I/O operations for file systems such as read, write, and remove. SPC-2 specifies I/O operations in the three areas of file processing, database-queries, and video on demand.

[15] http://www.tpc.org/
[16] http://www.spec.org/
[17] http://www.storageperformance.org/
[18] http://github.com/rxin/TPC-H-Hive
[19] https://issues.apache.org/jira/browse/PIG-2397

3 Big Data Analytics

In the data-centric world of the 21st century, Big Data analytics has become a major issue and research topic [34]. Cheaper data storage, a growing number of Internet users, connected devices, and sensors contribute to an ever increasing volume of collected data [35]. Data analysis has become a critical corporate asset, is disrupting industries, and enables new markets and opportunities [36].

This chapter offers an introduction into the world of Big Data and Big Data analytics. After defining the term, the main technology drivers are introduced to provide background knowledge for the techniques used in Big Data analytics software. Following this, the *Big Data analytic ecosystem* is introduced, describing a construct of multiple software frameworks working together [37]. In order to structure such an ecosystem, Khalifa et al. developed a 6-pillar model [18], which is presented thereafter. At last, a general architecture for Big Data applications is introduced.

3.1 Big Data

The origin of the term *Big Data* is not known, but assumed to be in the mid-1990s somewhere in Silicon Valley [38]. However, its widespread popularity began as recent as 2011 [39], as indicated by Figure 2, which shows a Google Trends[20] analysis.

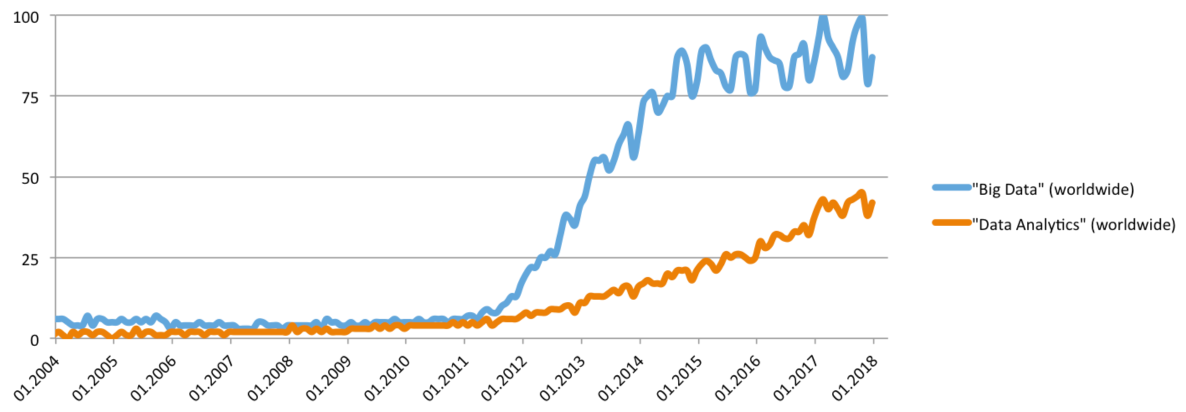


Figure 2: Popularity of "Big Data" and "Data Analytic" according to Google Trends from 2004 – 2017.

Even though the importance of Big Data has been widely recognized, a unified definition has not yet been reached. In the following some of the most popular ones are presented.

[20] https://trends.google.com

McKinsey & Company, one of the world's leading consulting agencies, describes Big Data as data sets, whose size is beyond the ability of typical database software tools to capture, store, manage, and analyse [40]. In this definition, McKinsey uses size as the primary characteristic. McKinsey does not set a fixed value but defines Big Data dynamically by linking it to the capabilities of current database tools. With this approach the definition of Big Data grows with the capabilities of databases, respecting the fact that data sizes too big to process today may become standard in just a few years.

Laney introduced a more diverse approach for the definition of Big Data by characterising it through *3 V's*: *Volume*, *Velocity,* and *Variety* [5]:

- **Volume** characterises size of Big Data. Similar to McKinsey's approach, the definition of volume is dynamical rather than absolute. Instead of being linked to a certain amount of petabytes, the size of Big Data is defined as too much to handle for common storage system [38].
- **Velocity** outlines the speed at which Big Data is generated. The growth in smart devices, sensors and online services has led to an enormous rate of data creation, which is still increasing [38].
- **Variety** describes the diversity of data types within Big Data. In Big Data, traditional tabular information (structured data) is often accompanied by photos, video (unstructured data), or emails (semi-structured data) [41].

While Laneys 3 V's represent the most popular understanding of Big Data, many suggestions were made to extend them. The International Data Corporation (IDC) added *Value* as a fourth V, outlining the importance of utilizing Big Data analytics to gain insights and to support decision making [42]. IBM coined the term *Veracity*, which addresses the uncertainty and unreliability of Big Data. And SAS proposed *Variability*, describing the variations in data flow rate [41].

Throughout the last decade, many other extensions were suggested, resulting in a total of 42 V's as listed by Elder Research [43].

3.2 Technology Drivers for Big Data Analytics

Over the last decade, Big Data analytics has enabled many industries such as retail and manufacturing to increase their margin by lowering operating costs, product development costs, and increasing customer experience [36]. Data centric companies such as Amazon[21] or Ebay[22] significantly transformed their respective market through their recommender systems [44]. Manufacturing companies have the potential of lowering their operating costs by up to 30% by adopting Big Data analytic technologies such as predictive maintenance [44].

The rising popularity of Big Data analytics is driven by an ever increasing capacity of storage and data processing, new and cheap ways of data generation, and the development of new data processing technologies. This chapter presents some of the important technological drivers behind Big Data analytics.

3.2.1 Distributed Computing

The high volume, velocity and variety of Big Data poses big challenges for computer systems such as high demands on storage space and processing power. Storing and processing large amounts of data often exceeds the capabilities of commonly available systems [45].

Distributed systems help in addressing these challenges. Instead of having just one single computer, multiple machines are connected together into a cluster. Data is distributed around the system and stored on different machines. Processing tasks are split into smaller ones, which are then executed in parallel. The system is easily scalable by introducing additional machines into the system.

Figure 3 illustrates the structure of a typical computer cluster in the example of the popular Big Data processing framework Apache Hadoop. For a cluster to work correctly it is important to keep track of all connected machines. Keeping track includes knowing their name, capabilities and how to reach them within the network. Furthermore, their status must be monitored for possible failures or breakdowns. This responsibility of managing the cluster is often separated from normal processing tasks and specifically assigned to one or more machines. By this way it can be separated from the cluster and is less prone for failure. As shown in Figure 3 on a Hadoop cluster the *NameNode* is responsible for overseeing the network. Besides monitoring the resources of a cluster, it is necessary to divide the computational work among it. Therefore, it is necessary to split tasks into multiple smaller ones and distribute them within the network. In the example of Hadoop, this responsibility falls upon the *Resource Manager*. It splits tasks up, monitors

[21] https://www.amazon.com/
[22] https://www.ebay.com/

the current utilization of all connected computing nodes, and distributes the workload accordingly. It also ensures that all tasks are completed successfully and, if necessary, redistributes failed tasks again. The third component of the cluster is the *DataNodes*. Their primary purpose lies in processing tasks and storing data. To prevent loss of data due to component failure, it is replicated multiple times across the cluster [46].

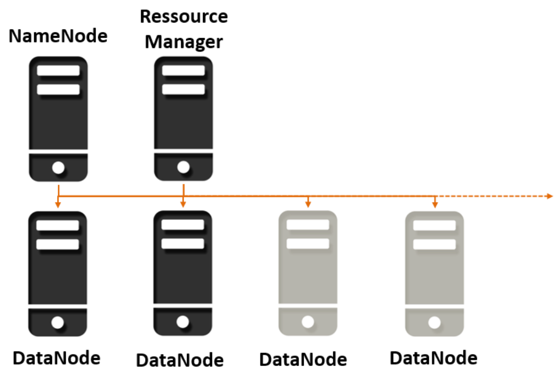


Figure 3: Architecture of a Hadoop 2.x cluster.

Distributed systems offer a lot of advantages over a single machines. They are easier to scale, naturally support running multiple applications, and are more reliable since they do not have one point of failure [47]. However, there is evidence that single-machine systems are sometimes superior due to the costs of task distribution and network traffic: A recent study suggests that in many common data analytic cases with data of up to 100 GB, a single-machine system is sufficient and even outperforms clusters [48]. Single-systems are however strictly limited in their processing capabilities, while distributed systems simply grow by adding components.

3.2.2 In-Memory

Traditionally a computer has two main kinds of storage systems: the hard disk and the random access memory (RAM). The hard disk is responsible for persistent data storage. It has a high storage volume and information stays available even after shutting down the system. The two most common technologies for hard disc storage are ferromagnetic drives (HDD) and solid-state drives (SDD), which use flash memory. RAM is traditionally responsible for volatile information. It offers very fast access speed but is expensive and information is lost if power is removed. Therefore, it is mostly used by software to store frequently used and currently needed information.

In recent years many Big Data analytic frameworks have switched from using disk memory to RAM for data storage [49]. This technique is generally referred to as keeping data *in-memory*. In-memory technology enables over 1.000 times faster access speed than the traditional disk-memory as illustrated in Figure 4. The figure shows the memory hierarchy of a computer with a two-core processor [49]. The hierarchy is defined in terms of access latency and the logical distance to the CPU. In addition to hard disk and RAM, the CPU internal memory caches are listed. Data transfers through the caches into the registry, where the core then processes it. As displayed, latency increases highly down the layers. While RAM is 100 times slower than caches, the latency of disk memory is over a 1.000 times higher than RAM.

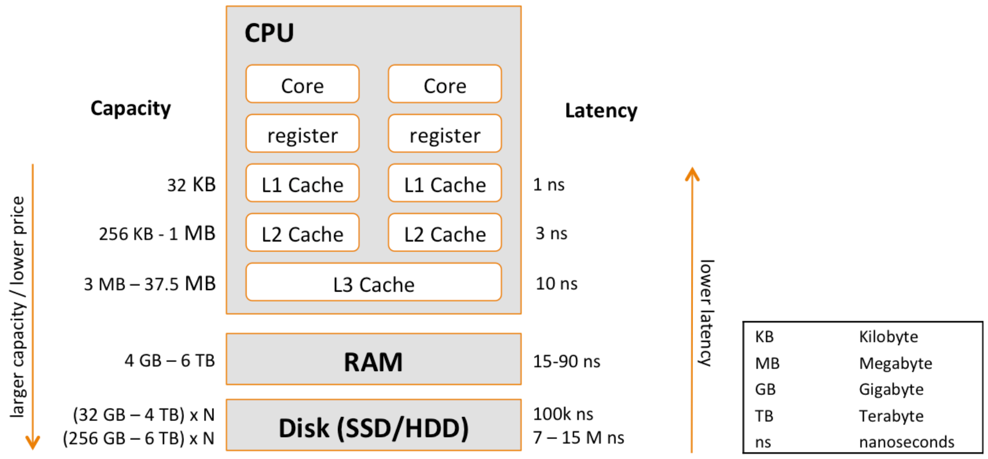


Figure 4: Memory hierarchy and access speed [49].

As mentioned above, the challenge of in-memory technology is preventing information loss if power is removed. Therefore, frameworks based on in-memory technology need rigorous backup mechanisms to ensure data recovery in case of unscheduled system shutdown. Although still very expensive, the highly increased accessing speed of RAM offers great value for companies and is predicted to gain market share with falling prices [50, 51].

3.3 Classification of Big Data Analytic Frameworks

Big Data analytic frameworks are often specialised on specific tasks such as data storage, processing or visualization. Utilizing Big Data analytics in a business context often needs solutions combining many different frameworks. Various tools assembled together form an *ecosystem*. In general, a *software ecosystem* consists of a set of software solutions. Such ecosystems enable, support and automate the activities of associated social and business systems. [52]. For Big Data analytic ecosystems, typical activities are collecting, preparing, transforming, storing and analysing data. To describe the requirements of a Big Data analytic ecosystem, Khalifa et al. [18] developed a model consisting of the six pillars *Storage*, *Processing*, *Orchestration*, *Assistance*, *Interface*, and *Deployment* as shown in Figure 5.

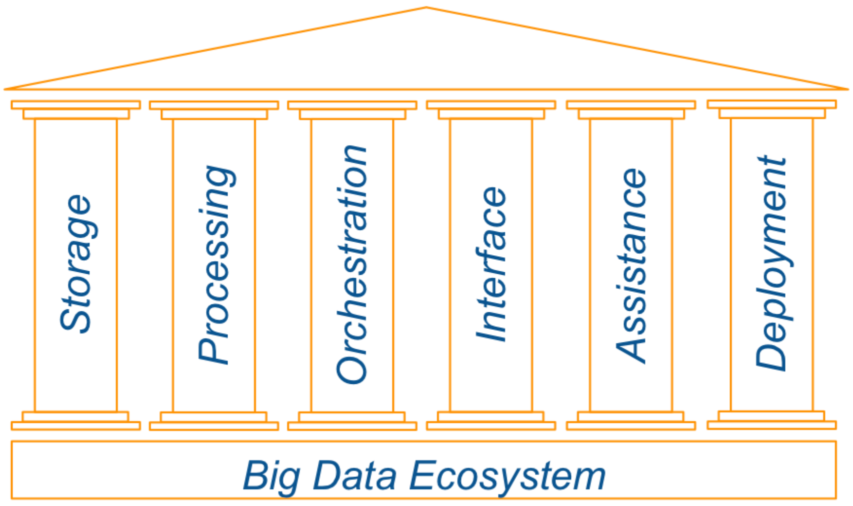


Figure 5: Six pillars for building Big Data analytics ecosystems [18,p. 3].

3.3.1 Storage Pillar

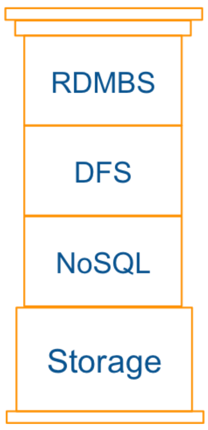


Figure 6: The storage pillar.

Storage is the first pillar of a Big Data ecosystem and describes all functions concerning preserving information [18]. Volume, variety, and velocity of Big Data put high requirements on data storage systems: Big Data storage systems must be able to store an ever growing amount of information in many formats such as tables, text or video [53]. They also must provide high access speed to cope with the high rate new data is entering the system [53]. Traditional data management systems do not satisfy these requirements, which has led to the development of new approaches [54]. To address the challenges of volume, variety, and velocity, Big Data ecosystems often rely on distributed storage systems specifically designed to handle large amounts of diverse information. As shown in Figure 6, storage systems can be divided into three sub categories: relational database management systems (RDBMS), distributed file systems (DFS) and Not-only structured query language systems (NoSQL).

Relational Database Management Systems (RDBMS)

Relational database management systems organize data in one or more tables, following a model first proposed by Codd in 1970 [55]. Tables consist of columns and rows, where each row is identified by a unique key-value. The columns define the schema of the table while the rows hold the information.

Relational databases are designed to support the ACID (Atomicity, Consistency, Isolation, and Durability) properties, first mentioned by Haerder and Reuter in 1984 [56]. To conform to ACID properties, transactions should either succeed or be rolled back (Atomicity), never leave the database inconsistent (Consistency), never interfere with each other (Isolation), and persist even after restart of the database (Durability). A transaction is a short sequence of interactions with the database through which a user can manipulate the data.

According to the information platform DB-Engines[23] RDBMS is by far the most popular database technology with an overwhelming popularity score of 79.6% [57]. This is also supported by the yearly data connectivity report published by Progress [58], which states that only 2% of respondents do not use a relational database. However, it has to be mentioned that both surveys are not focused on Big Data solutions but include the whole infrastructure of a company. While RDBMS are still the most popular choice in Big Data storage, their dominance is not as high as the surveys suggest. Due to the variety of Big Data, other data storage techniques are on the rise [58].

Distributed File Systems (DFS)

Distributed file systems (DFS) offer similar functionalities as file systems implemented in popular operating systems such as Windows or Linux. In contrast to them, files are not stored on a single computer but are distributed across multiple machines.

[23] https://db-engines.com/

Observing the distributed file systems at Google, Ghemawat et al. [59] concluded that machine breakdowns inside a DFS are unavoidable. DFS often consist of hundreds or even thousands of inexpensive machines. Due to the huge size of systems and poor component quality machine breakdowns are rather common. To address this issue, DFSs implement rigorous methods to prevent data loss. Many products such as the Google file system (GFS) [59] or the Hadoop distributed file system (HDFS) [60] replicate data within the system. This replication ensures that loss of hardware components does not result in loss of data.

Not only Structured Query Language Systems (NoSQL)

Not only structured query language (NoSQL) databases are the third and final element of the storage pillar. According to Brewers CAP theorem, any networked shared-data system can achieve at most two of the three desirable properties *consistency*, *high availability*, and *partitioning* [61].

- *Consistency (C)* means that the data is always the same across the entire system. All users have the same view of the data at all time.
- *High availability (A)* entails that every request results in a meaningful result rather than error messages or silence. The higher the availability, the faster these requests are processed.
- *Partitioning (P)* implies that the system can be separated without further compromising the before mentioned qualities. Consistency and availability can be maintained even in the event of message loss or partial system failure.

As described above, relational databases are designed for strong consistency and serializability. Following the CAP theorem, they are not able to offer high availability. Opposite to that NoSQL databases sacrifice consistency in order to provide high availability and serializability. Instead of being consistent at all time, temporary inconsistency is possible. Still, eventually, consistency at a future state is guaranteed [62].

3.3.2 Processing Pillar

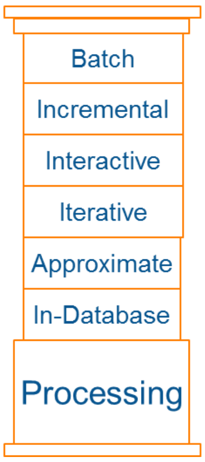


Figure 7: The processing pillar.

Processing is the second pillar of a Big Data analytic ecosystem. It includes all activities of manipulating and analysing data [18]. Similar to storage systems, volume, velocity, and variety of Big Data demand special requirements of Big Data processing systems: Analysing tera- or even petabytes of diverse data at once exceeds the capabilities of single computers [63]. Therefore, distributed systems are used to process data in parallel on multiple computers. Tasks are split into smaller parts that can be executed simultaneously on multiple machines. As shown in Figure 7, there are six different approaches how Big Data is processed: *batch*, *incremental (*also called *stream processing)*, *interactive*, *iterative*, *approximate*, and *in-database* processing.

Batch Processing

Batch processing describes executing a series of commands without manual intervention [18]. The program containing the commands is defined before the analysis and runs uninterrupted from start to finish. Therefore batch processing is best suited for complex analysing of large data sets [25]. After initialization, the process can take minutes, hours or even days. Correspondingly, it is not well suited for real-time processing. Figure 8 illustrates the workflow of a batch process. Before processing, individual data is collected and combined into one big data set. Additionally, the program to analyse the data set is written. Both are then submitted to the batch-processing engine, which produces the desired results.

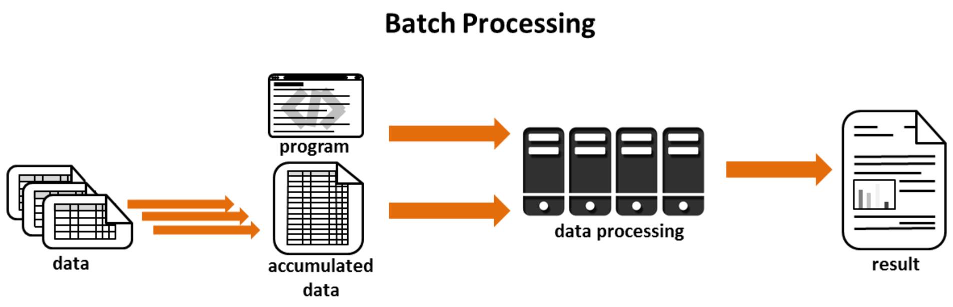


Figure 8: Workflow of batch processing.

Batch processing is generally used for big and complex analysis where execution time is not essential. For many Big Data analysis use cases such as sales forecast, customer segmentation, or medical diagnostics this method is sufficient.

Batch processing is not necessarily slow. There have been efforts to achieve almost real-time analysis using very small batch processes, executed in succession. Data is collected from a very small timespan and analysed in intervals of milliseconds [22].

Incremental Processing

Incremental or stream processing focuses on the analysis of moving data instead of resting data sets like batch processing. This means that data is processed immediately when it gets available [18].

In many cases today data is collected continuously and at a high rate: User behaviour is observed live, machines are monitored non-stop, and transactions are tracked steadily. Gathered information is only valuable a short amount of time and real-time analysis is essential. Product recommendations, as for example done by Amazon, must be calculated almost instantaneously or they do not benefit their customers.

Figure 9 displays the general workflow of stream processing. In contrast to batch processing, data is not accumulated but processed as soon as it enters the system. The analysis program runs continuously and waits constantly for new data. Rather than producing one single result, stream processing provides continuous analysis.

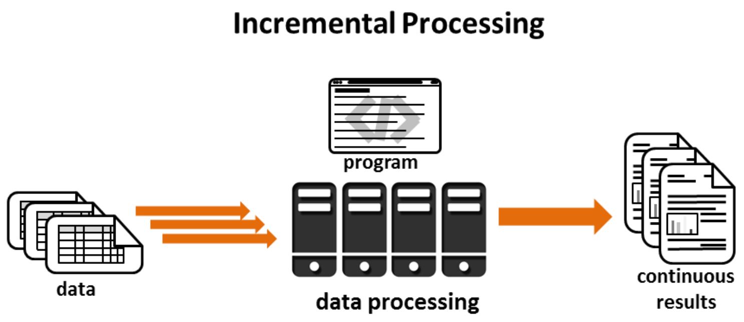


Figure 9: Workflow of stream processing.

The key challenge in stream processing is keeping the latency low [22]. Latency describes the time that passes from the moment data enters the system until it is processed. In order to handle large amounts of data, many modern frameworks for stream processing implement a distributed processing methodology [64].

Interactive Processing

Interactive processing allows for user interaction during an analysis. As Big Data analysis becomes more popular in companies – experiencing a year-to-year growth of 11% between 2015 and 2016 [58] - the user base grows steadily. Due to new applications in a

wide range of industries, new types of users have emerged. Instead of traditional long and complex analysis, ad-hoc queries and reports are gaining ground and are responsible for up to 80% of a company's workloads [65].
As shown in Figure 10, the user has multiple interactions with the system during interactive processing. In contrast to batch processing not all steps have to be predefined, but user input is possible after initialization.

The entire process can consist of multiple queries and analyses. The system fetches necessary data from the databank and processes it accordingly.

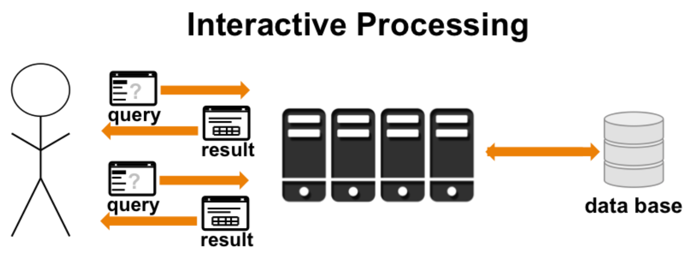


Figure 10: Workflow of interactive processing.

Interactive processing frameworks are designed for fast execution of small jobs consisting of data queries. There are multiple methods available to optimize query performance. Apache Tez, for example, reduces the overhead of launching queries resulting in faster initialization times [66]. Apache Spark saves intermediate results in-memory, enabling faster response times for future queries concerning the same data [67]. And Google Dremel as well as Apache Drill have optimized accessing data by only searching through relevant columns [68].

Iterative Processing

Iterative processing describes workloads that repeatedly do the same processing steps or run multiple times through the same data set [18] as illustrated in Figure 11. Machine learning and graph processing algorithms are common examples containing a lot of iterative computations [24].

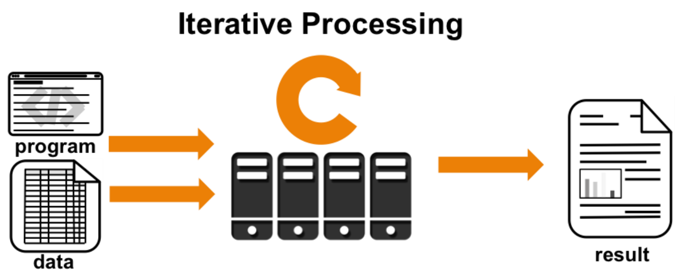


Figure 11: Workflow of iterative processing.

Iterative execution engines are optimized on reusing input data, code, or intermediate results. They often hold frequently used data in-memory for fast access [24].

Approximate Processing

The goal of *approximate processing* is to deliver very fast results from analysis of large amounts of data. In order to increase their speed, accuracy is compromised. Results are not found by analysing the whole data set but approximated using a representative sample [18].
This allows for almost real-time response time when querying peta and exabytes of data [69–71]. It is especially useful in situations where an exact result would not benefit the quality of decisions or where data is incomplete and noisy to begin with. Approximation is also applicable for predictions or other statistical analysis.
Figure 12 illustrates the general concept of approximate processing. The program does not analyse the whole data set but approximates the result processing only part of the available data.

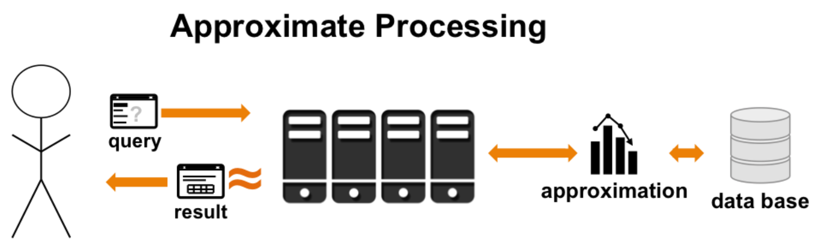


Figure 12: Workflow of approximate processing.

Nair [72] argues that the amount of data and data analysis will exceed the available processing capabilities. In order to cope with the vast amount of data, approximation techniques are necessary.

In-database Processing

In all other introduced concepts, processing of data is separated from data storage. This separation requires the movement of data between storage and processing layers. The concept behind *in-database processing* is to move data processing tasks inside the storage layer as shown in Figure 13. The goal is to eliminate data movement between storage and processing by analysing it without moving it out of the database [18].

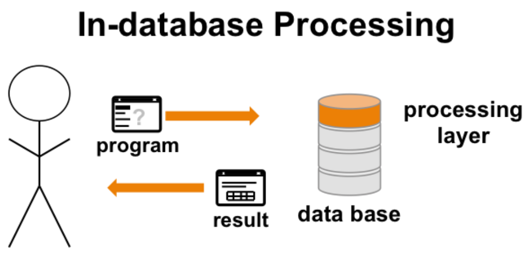


Figure 13: Workflow of in-database processing.

Apache MADLib[24], for example, offers machine learning in SQL. Its goal is to operate on the data in-database and eliminate unnecessary movement of data between multiple runtime environments. MADLib provides various supervised and unsupervised machine learning algorithms as functions that can be called via SQL.

3.3.3 Orchestration Pillar

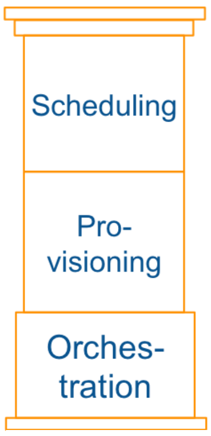


Figure 14: The orchestration pillar.

In a distributed system the workload is divided and spread around multiple nodes. Tasks are assigned to individual computers depending on available resources. Resources include memory, CPU, network capacity and disk storage space. The pillar of *Orchestration* covers the management of these resources inside a cluster [18]. The management of a cluster is generally separated from the clusters and the responsibility of the *resource manager* [18]. The responsibilities of a resource manager include monitoring status and current workload of all nodes. Furthermore it distributes the workload among the cluster and supervises the progress of individual tasks [73]. As illustrated in Figure 14, orchestration frameworks are differentiated depending on their techniques of resource allocation: scheduling or provisioning.

Scheduling

Scheduling frameworks maximize resource utilization or data locality. Maximizing resource allocation aims at using as many of the above mentioned resources of the cluster as possible instead of letting only a few nodes work. This means spreading the workload as wide as possible among the cluster. Maximizing data locality aims at minimizing data transfer within the cluster. The goal is to process data only on those nodes it is stored on. Maximizing resource utilization and data locality can almost never be achieved together.

[24] http://madlib.apache.org/

Data is not replicated on every node. Therefore, in order to utilize every resource of the cluster, data has to be transferred. Likewise, maximizing data locality results in processing data on only those nodes the information is stored on, disregarding others. Therefore, either resource allocation or data locality has to be prioritized [18].

Provisioning

Provisioning minimizes job execution time and monetary costs. This is especially helpful on cloud-based solutions. In cloud environments the user is charged depending on the amount and time resources are used. Provisioning frameworks take the increasing costs of additional nodes into account when distributing resources and weigh them against the benefits of more processing power [18].

3.3.4 Interface Pillar

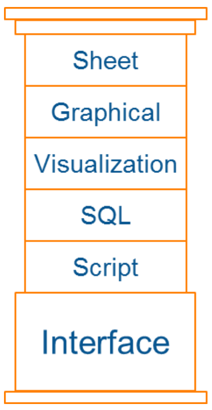


Figure 15: The interface pillar.

As mentioned before, Big Data analytic systems have a broad audience. In order to match their specific needs, user *interfaces* are developed offering different access points [18]. They allow users to interact with Big Data analytics in a familiar environment by providing an abstraction level from underlying functions such as processing or storage. Interviews with expert show that there is a lot of potential in improving user experience for beginners, since many programs today are primary designed for professional users [74].

Khalifa et al. [18] distinguish five main approaches to interfaces: *sheets, graphical* interfaces, *visualization* tools, interfaces providing *SQL* capabilities, and *scripts* (see Figure 15).

Sheet Interfaces

Sheet interfaces offer environments similar to popular spreadsheet based tools like Microsoft Excel. Although very consumer friendly, they are commonly only suitable for data exploration and preparation. For model building and complex analysis, other tools are necessary [18].

Spreadsheets are established tools in data science [75] and especially popular for business users. Projects such as the from Barga et al. [76] introduced Daytona or OpenRefine[25] aim at providing a spread-sheet like environment for Big Data analytics.

Graphical Interfaces

Graphical tools provide a visual interface for Big Data analytics. Instead of writing code, users are able to assemble workflows via drag and drop or use menus to analyse data [18]. Big Data analytic features can be accessed through a graphical user interface.
The field of traditional data analysis already offers many tools with a graphical interface. Solutions such as RapidMiner[26], IBM SPSS[27], SAS[28] and KNIME[29] provide highly developed graphical user interfaces, which offer a huge variety of data analytic functions. However, they are limited to processing data on one machine, limiting the amount of data to be processed at once.
In the last decade, there has been some effort to bring graphical user interfaces into the field of distributed Big Data analytics. Especially commercial products such as Microsoft Azure[30] and IBM Watson[31] have worked hard on providing user-friendly interfaces. Other examples of graphical user interfaces in the domain of Big Data analytic frameworks are Radoop[32], which extends RapidMiner to work on the distributed processing framework Hadoop [77], and WINGS[33], which provides a drag and drop interface to create workflows for large computational experiments [78].

Visualization Interfaces

Large data sets are difficult to process for the human mind. In the sense of the old proverb “a picture is worth more than thousand words”, visualization of data helps to comprehend and analyse information. While before mentioned graphical tools provide a graphical user interface to analyse data (i.e. menus or drag and drop), *visualization* tools provide a visual presentation of Big Data itself (i.e. bar graphs or diagrams) [18].

[25] https://openrefine.org/
[26] https://www.rapidminer.com/
[27] https://www.ibm.com/spss
[28] https://www.sas.com/
[29] https://www.knime.com/
[30] https://azure.microsoft.com/
[31] https://www.ibm.com/watson/
[32] https://rapidminer.com/products/radoop/
[33] https://www.wings-workflows.org/

Data visualization can be utilized during the whole process of data analytics: from initial data exploration to representation of results [79]. Visualization helps in analysing outliers, recognizing patterns, or determining important features. It can also serve as a communication tool. Companies consist of different stakeholders such as executives, functional leaders and data scientist. Each of them works with different methods and communication channels. Data visualization supersedes these differences and provides a single platform for discussions [80].

The visualization of Big Data introduces many additional challenges in comparison to traditional data visualization. Big Data is often composed of various different data types. Visualization tools must therefore be able to deal with semistructured and unstructured data. They must also be able to process large amounts of data, preferably in parallel to provide scalability. Big Data can be very complex, consisting of multiple dimensions. Multidimensional data is too complex for the human mind to process at once. One of the biggest challenges of data visualization is to reduce complexity enough to make information comprehensible without losing significance or missing important connections [81]. Many commercial Big Data analytic solutions such as IBM Watson or Microsoft Azure offer many features to visualize data in different ways such as bar graphs or pie charts. They utilize their underlying distributed computing power to cope with the large volumes of data. Microstrategy[34] and Tableau [35] offer tools that can easily visualize data from different sources and formats and present them in interactive visualizations.

Driven by the entertainment industry, there is a lot of research going into the field of visualization. New technologies such as virtual and augmented reality offer immersive user experience. In recent years, there has been some effort to combine Big Data with augmented reality, raising Big Data visualization into the next dimension [82–85].

Structured Query Language (SQL) Interfaces

Developed by IBM in 1976 [86], *SQL* is the de-facto standard language for relational database management systems and is supported by the most popular systems [57]. SQL enables the user to add, update, delete, or find data inside a database. However, SQL is limited to structured data formats. Due to the variety in formats within Big Data, many Big Data applications utilize NoSQL databases to manage unstructured and semi-

[34] https://www.microstrategy.com/
[35] https://www.tableau.com/

structured data. They are often a better fit [62] and grow steadily in their popularity [58]. Due to their different structure, not all of them natively support SQL.
Due to the wide popularity of SQL and its big support among many tools, *SQL interfaces* aim at providing NoSQL databases with a SQL interface. This way Big Data can be accessed through the well-known language of SQL, simplifying the switch to a NoSQL system [18].

Script Interfaces

Scripts offer algorithms, functions or other code fragments for Big Data analytics. Users do not have to implement algorithms from scratch but can build on prior work [18].
The lowest level of scripts provides support for higher languages such as R[36] or Python[37], which are currently the most popular languages for data analytics [87] This allows users to create analytic processes in a higher language and port it to different Big Data analytic platforms. Projects such as “R on Hadoop” [88] for example bring R to platforms for distributed computing. Other languages such as Pig Latin [89] or Jaql [90] are specifically designed as scripting languages for Big Data analytics and tailored for distributed processing. Code libraries present a higher level of scripts. They contain already implemented algorithms that can be reused. Especially complex algorithm families like machine learning have led to the development of many libraries [91–94].

3.3.5 Assistance Pillar

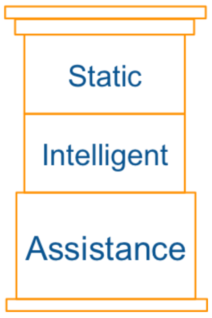


Figure 16: The assistance pillar.

According to McKinseys 2016 Big Data report [36], there is a severe shortage of qualified analytical talents, in particular data scientists. During the report, many executives across geographies and industries where interviewed, stating great difficulties finding qualified analytical personnel. The lack of qualified personnel greatly decreases a company’s capability of implementing Big Data analytics.
Assistance frameworks make Big Data analytics more accessible. They provide support during the design and implementation of analytical processes. As shown in Figure 16, assistance frameworks can be separated into the two major categories of *static* and *intelligent* tools [18].

[36] https://www.r-project.org/
[37] https://www.python.org/

Static Assistance Tools

Static assistance tools always offer the same assistance, regardless of the data set. They are not aware of the specific context and provide general support. This category includes well-known tools such as tooltips, wizards and help pages [18]

Static tools alone are often not sufficient. There is a vast amount of analysis techniques available. Choosing the correct one is usually highly dependent on the available data set and use case, which static tools do not take into account. Novices and experts alike often struggle during the selection process due to insufficient knowledge [95]. This has led to the development of intelligent tools.

Intelligent Assistance Tools

Intelligent assistance tools take the context of a data analysis into account. They provide different support depending on the use case and data set [18]. For example, they may suggest different analytical techniques depending on the size of a data set or on its value distribution.

Bernstein et al. [96] propose an intelligent discovery assistance tool, that suggests data mining processes depending on the input data as well as desired mining results. It searches the space of possible processes and ranks them by speed and accuracy. This greatly simplifies the selection process between multiple data analytic algorithms.

In recent years many tools have emerged [95], assisting users during the whole process of knowledge discovery [97]: *data selection*, *preprocessing*, *transformation*, *data mining*, and *interpretation*.

3.3.6 Deployment Pillar

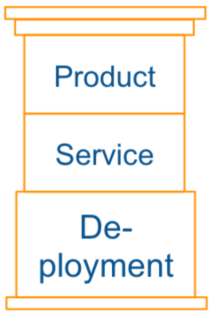


Figure 17: The deployment pillar.

The sixth and final pillar consists of *deployment* methods for Big Data solutions [18]. When a company decides to adopt Big Data analysis it is faced with many decisions beyond the analysis process itself. Questions such as “Does the company want to develop the software themselves or outsource it?” or “Do they use their own or third-party IT infrastructure?” have to be answered. Depending on strategic, resource and their operating environment factors, different strategies are best suited [98]. As shown in Figure 17, the product and service model can be distinguished during deployment.

Product Model

In the *product* model organizations buy or develop the software and deploy it on their own infrastructure. This ensures data security and privacy. It also benefits processing large on-site data since it does not have to be uploaded to third party servers [18].

Service Model

The *service* model describes the approach of outsourcing the infrastructure to an external service provider [18]. This offers companies flexibility in adding or reducing infrastructure and releases them from the responsibility of maintenance.
The extent of the service can differ in their depth. In general, three approaches can be differentiated: The *Infrastructure as a Service* (IaaS), *Platform as a Service* (PaaS) or *Software as a Service* (SaaS) model [37].

- In *IaaS*, the service provider offers its hardware with basic software like operating system, network security and similar utilities.
- *PaaS* provides an environment on which applications can be developed. The user does not have to maintain the platform but only his application.
- Following the *SaaS* approach the user consumes the whole analysis service. The service provider develops and maintains the software and the customer pays for the entire software.

3.4 Architecture of a Big Data Ecosystem

The above presented framework from Khalifa et al. [18] can now be used as the foundation to describe a general architecture of a Big Data analytic ecosystem. The six pillars can be transformed almost directly into the components of a general software architecture. Only the last pillar, deployment, represents implementation requirements rather than an architectural component and is therefore not considered. Retaining the names introduced before, the components can be separated into storage, orchestration, processing, interfaces, and assistance.

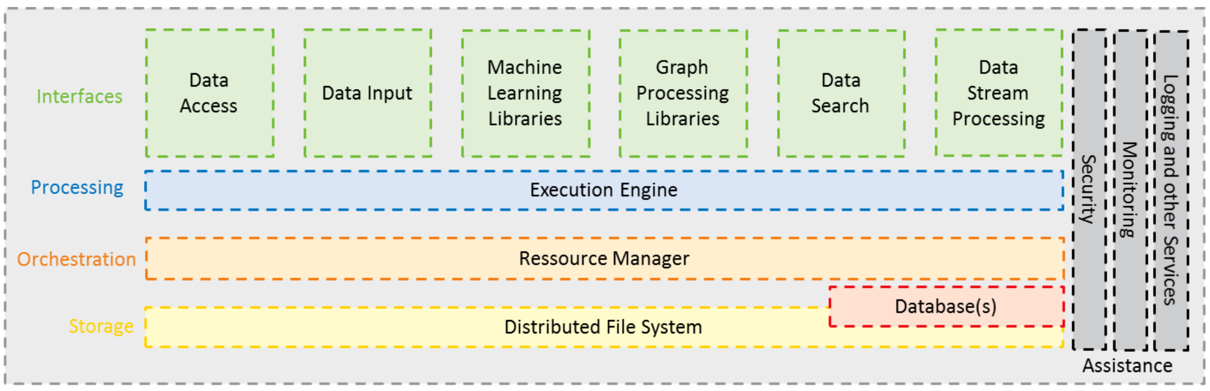


Figure 18: High level architecture of a Big Data analytic ecosystem.

Figure 18 shows a graphical representation of the architecture of a Big Data analytics ecosystem. Starting from the bottom, the first layer represents the storage component. Data can be stored either on a distributed file system or on (distributed) databases, as depicted in yellow and red. As mentioned above there are many types of databases. An ecosystem is not limited to only using one storage system but can combine multiple systems (i.e. a relational database for financial data and a NoSQL database for emails). The next layer, depicted in orange, is the resource manager. The resource manager is vital in distributed environments. It is responsible for managing the computer cluster and its resources as well as distributing the workload among all computing nodes. The processing layer, here represented in blue, focuses on the workload itself. It is responsible for analysis and manipulation of data. It defines the programming paradigm used for all data processing and therefore influences how applications must be written. As described before there are different programming paradigms that are implemented in different execution engines. Execution engines offer APIs for different programming languages. The famous Hadoop MapReduce framework for example offers an API for Java. Developers can write a program consisting of Mappers and Reducer in Java and feed it into the execution engine. Creating a program for one execution engine however is often linked to writing a large amount of engine specific code. This especially applies when implementing complex analysis. The interface layer, depicted in green, addresses this problem by adding an additional abstraction layer. Interfaces are situated between the execution engine and application developers. They provide already implemented functionalities that can be used by developers. They range from simple data access and search applications to complex machine learning libraries. While most of them are aimed at one specific execution engine, some offer support for more than one. This allows developers to switch between execution engines in the future and increases longevity of applications. The last layer, illustrated in black, are assistance tools. They provide functionalities over the entire system but are not directly related to the analytical process. This includes for example security, logging services and programs related to monitoring cluster health and performance.

The goal of this general architecture is to provide a frame of reference when describing and comparing Big Data analytic ecosystems. The architecture is a tool to compare Big Data analytic ecosystems in a uniform way. Differences can be recognized and formulated in a consistent manner. Singular components of the architecture can be exchanged to form a new ecosystem.

4 Predictive Maintenance

4.1 Overview of Maintenance Techniques

Maintenance describes all activities necessary to restore equipment, or to keep it in a specified operating condition [2]. Maintenance retains or prolongs the life of an item. Producers as well as consumers encounter the issue of maintenance on a regular basis. Cars, jet engines, laptops, manufacturing plants, houses and even software systems need to be serviced periodically to maintain their functionalities.

In many companies, maintenance related costs are only depicted as a simple cost centre, divided into direct and indirect costs [100]. Regarded as cost centres, maintenance costs rise and fall in direct relation to performed services. In reality however, maintenance can increase productivity and profitability of a company by influencing their entire value chain [101]: Poor maintenance results in higher deterioration, more breakdowns, and reduced reliability. Breakdowns delay production, decrease production capacity, and raise overall uncertainty of the production process. Poorly serviced machines are prone to making more mistakes, reducing product quality and increasing rejection rate. On the other hand, a well-implemented maintenance system can affect the whole process, increasing quality and reliability while minimizing costs for spare parts and fluids [100].

Many scientists have analysed the impact of maintenance on the production systems. Al-Najjar and Alsyouf [100] developed a model to identify, monitor and improve the impact of vibration-based maintenance. When the model was tested in a Swedish paper mill, it reduced maintenance costs over 25% (-0.353 Mio USD). Furthermore, potential profits due to elimination of unscheduled downtime were estimated at around 3 Mio USD. Carter [102] examined the possible gains through maintenance systems of high-capacity coal shovels. Since shovels are becoming bigger and more efficient any downtime has an increased effect on production outcome. Therefore, minimizing machine breakdowns has a significant impact on overall productivity.

The literature distinguishes *breakdown maintenance*, *time based preventive maintenance*, and *predictive maintenance* as the three main types of maintenance [3, 103, 104]. As explained below, their main difference lies in the method used to determine maintenance needs.

4.1.1 Breakdown Maintenance

Breakdown maintenance is the simplest and oldest type of maintenance. As illustrated in Figure 19, in breakdown maintenance a machine is repaired after a failure occurs.

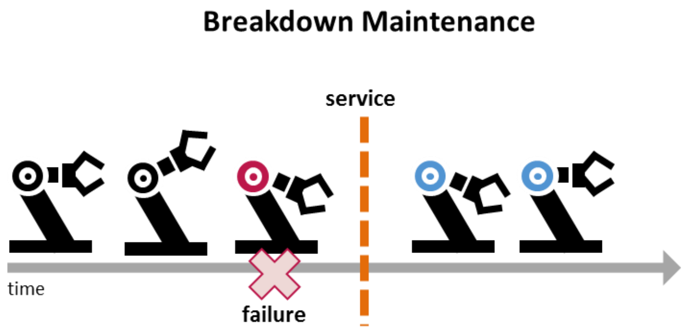


Figure 19: Process of Breakdown Maintenance.

The orange line indicates the time a broken component (red) is replaced (blue). Breakdown maintenance is very efficient, since each component operates its entire lifespan (from instalment until breakdown). However, breakdown maintenance also produces the most amounts of failures out of all maintenance types, since no proactive actions are taken to prevent them. It also takes some time between discovering and fixing a failure. Therefore, sudden breakdowns can lead to very expensive delays throughout the production chain [104].

4.1.2 Time Based Preventive Maintenance

Time based preventive *Maintenance* is based on the assumption that similar machines deteriorate in comparable fashion [105]. To prevent failures, machines are periodically serviced, regardless of their current health as shown in Figure 20: .

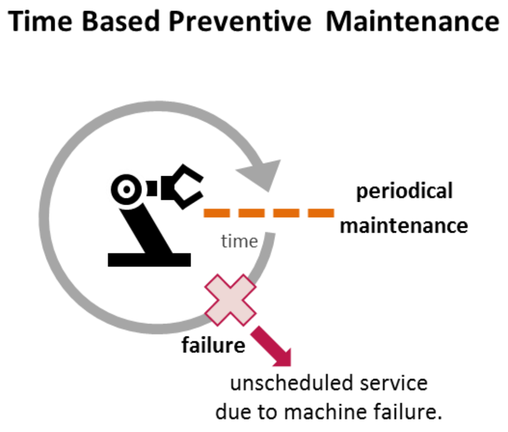


Figure 20: Process of Time Based Preventive Maintenance.

The orange line indicates the periodical service of the machine; the red X shows the remaining possibility of unscheduled services due to failures. Time and substance of the periodical services are based on failure statistics. These statistics can be derived from predecessor models, machine tests or experience. Preventive maintenance can prolong the remaining useful life of a machine by targeting vulnerable components directly. It also lowers the risk of unplanned failures and production breaks since maintenance can be scheduled during times of lower capacity utilization [105]

4.1.3 Predictive Maintenance

Predictive maintenance, also called condition-based maintenance, is the newest maintenance type. Building up upon the approach of time based preventive maintenance predictive maintenance follows the goal of exchanging components before they fail. In contrast to preventive maintenance, predictive maintenance does not only rely on average-life statistics but also considers the current condition of a machine. Specific environmental effects are part of the calculation at which time a machine should be serviced. This approach leads to increased efficiency for maintenance tasks. While preventive maintenance may exchange parts to soon, predictive maintenance ideally intervenes just before a component will fail. Monitoring the condition of a machine also enables catching premature failure due to higher wear and tear.

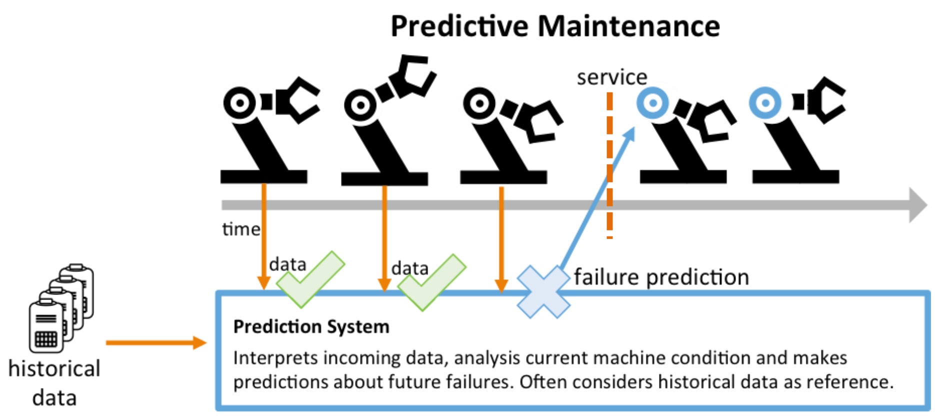

Predictive Maintenance
service
time
data
data
failure prediction
historical data
Prediction System
Interprets incoming data, analysis current machine condition and makes predictions about future failures. Often considers historical data as reference.

Figure 21: Process of Predictive Maintenance.

Figure 21 depicts the general process of a predictive maintenance system. Machines measure their current condition and report the data to a prediction system. Depending on an analysis of incoming measurements, often supported by historical data, the system makes predictions on possible machine failures. These predictions are then interpreted and may induce machine service [105].

A predictive maintenance system consists of three main steps. The first step is acquiring data to obtain information relevant to the systems condition. In the second step this data is processed for better understanding and interpretation. Finally, efficient maintenance policies are recommended during the last step of maintenance decision making [104] (see Figure 22).

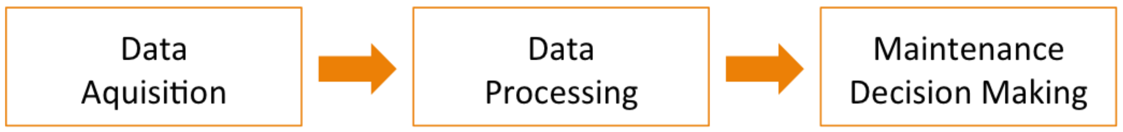


Figure 22: Three steps of predictive maintenance.

The subsequent chapters analyse these three steps in detail and introduce different techniques and approaches.

4.2 Data Acquisition

The foundation of every predictive maintenance system is information about the current condition of a machine. This information is the basis for future predictions resulting in maintenance decisions. Therefore, collection and storage of information is considered as first step of a predictive maintenance system. Gathered data can be separated into *condition monitoring data* and *event data* [104].

4.2.1 Condition Monitoring Data

Condition monitoring data consists of all measurements related to the health of a machine [104]. In general, sensors inside or next to a machine collect measurements automatically. Measurements are either periodically or continuously. Modern sensors allow for a multitude of available data sources. The most common sources include vibration data, acoustic data, oil analysis, temperature, electrical measurements, pressure, moisture, humidity, weather, or environment data [3, 104, 105].

Depending on the sensor, data is recorded in different data types: The simplest type is a single value, for example, determining the oil pressure within a machine. Another data type is a wave, which records continuous data such as sound or vibration. The last possible type is multidimensional data, where one measurement consists of multiple values. The most common multidimensional data type is an image [104].

Vibration Monitoring

Vibration monitoring refers to measuring the vibration of machines with non-destructive sensors and analysing equipment [105]. Vibration data is used to detect wear, imbalance, misalignment, loosened assemblies or turbulence on machines with rotational or reciprocating parts [106]. Rotational and reciprocating parts produce significant frequencies on various amplitudes. Deviations can be used as indicators on their condition [107]. Rotational and reciprocating parts are integrated in most machines used in manufacturing plants. Therefore, and due to the fact that vibration monitoring allows the detection of a multitude of different problems, it is the widest used technique in predictive maintenance [108]. Vibration sensors are able to produce both continuous data and periodical measurements. Analysis of vibration data is especially well suited for detecting failures in the early stage of a machine, right after its installation [100].

Sound or Acoustic Monitoring

Another way of monitoring the condition of a machine is analysing its sound emissions. This technique has a strong relationship with vibration monitoring and can be used in similar environments [105]. Goti [109] implemented a predictive maintenance system based on data collected by electronic stethoscopes in a Spanish manufacturing plant. His results indicate that sound monitoring can be a cost efficient alternative to vibration monitoring. However, it is often complicated to isolate the sound of single machines, especially in environments where multiple machines are working close to each other.

Oil Analysis and Lubricant Monitoring

Lubricants like oil are used to decrease effects of wear, friction, and heat generation of moving parts. Leaks, broken off fragments and oxidation can lead to contamination resulting in reduced effectiveness of the lubricants or even harming the machine [107]. Oil analysis and lubricant monitoring serve two purposes [105]. At first, it provides information about the current state of the fluid to analyse if it is suitable for further use, or if it needs to be exchanged. Second, it serves as an indicator for wear conditions of internal oil-wetted components. The use and viability of fluid monitoring for predictive maintenance has been analysed in some scientific papers. Gonzales et al. [110] for example implemented a predictive maintenance system for cogeneration engines based on the analysis of circulating fluids resulting in a working system able to make suggestions for component replacements. Lukas and Anderson [111] utilized lubricant analysis for condition monitoring of gas turbines. And Kalligeros [112] examined the possibility of lubricant analysis to determine maintenance needs of hydraulic lifts.

Temperature Monitoring

Temperature can be measured either by temperature sensors or infrared emissions. Deviations of temperature are signs of pending problems [107]. Excessive heat for example can indicate too much friction, problems with heat dissipation or lubrication issues. Temperature monitoring is often used on electric and electronic components to control power flow [105]. Especially infrared sensors are highly influenced by their environment. Changing seasons, open windows or the installation of a machine nearby can result in temperature fluctuations. The system has to account for these environmental effects and include them in the calculations.

Electrical Monitoring

There are two main ways to monitor electrical signal. First, it can be measured how much electricity a machine uses. Second, changes in equipment properties such as resistance, conductivity, dielectric strength, and potential can be observed [105]. Variations in either property indicates broken parts, excessive heat or shortages.

4.2.2 Event Data

In contrast to conditional monitoring data, event data is neither measured continuously nor periodically, but event based. Event data is created whenever a specific incident occurs. Event data often includes incidents such as installation, errors, breakdowns, repairs, or component changes. Although many events must be recorded manually, modern machines allow for at least partial automation. Error logs for instance allow an automated registration of errors. Although often receiving less attention, event data is as important as condition monitoring data for predictive maintenance systems since they are good indicators of future failures (i.e. frequent small errors can indicate a bigger error) [104].

4.3 Data Processing

Data processing is the second step of a predictive maintenance system. It describes handling and analysing the data collected during data acquisition [104]. Raw data often contains errors or missing values. These lead to reduced data quality. Therefore, data is often processed first, cleaning out poor quality data and increasing its analysability.

4.3.1 Data Cleaning

Data quality is the fitness of data in regards to its purpose [41]. Data quality is critical. Often referred as “garbage in – garbage out” [7, 104, 113], analysis performed on poor quality data can lead to misleading or wrong conclusions. Poor data quality can be caused by different factors: Sensors may malfunction or miss a measurement, data transmission can fail due to network errors, or humans enter faulty inputs. Before analysing data it is therefore important to clean “dirty data” [104]. Data cleaning is the process of identifying and possibly fixing data errors [114]. There is not one single method to handle data errors. The subsequent paragraphs give an overview to the most popular techniques. Since data cleansing has become a growing research area with an increasing amount of contributions, a more in-depth analysis is beyond the scope of this thesis.

Sensor Failures

With often hundreds of machines under surveillance, sensor failure is a common issue in the field of predictive maintenance. There are usually four types of sensor failures: bias, precision degradation, complete failure, and drift [115]. Biased failures describe errors due to measurements that are off by a specific amount, for example, a light sensor that catches only 80% of light due to dust particles on its lens. Precision degradation outlines the fact that many sensors loose precision over time due to wear and tear. Complete failure implies a sensor malfunction and drift errors arise when sensor measurements drift in one direction over time. Various strategies have been introduced to handle sensor failures. For instance, Xu and Kwan [115] approach the issue by building a residual model for a given system based on input-output measurement data. Future sensor measurements are tested against that model to detect failures. Koushanfar et al. [116] developed a cross-validation based technique for detection of sensor faults. They detect errors by analysing sensor data for inconsistent readings.

Missing Data

Missing data describes incomplete data rows. Data analysis processes are seldom able to handle missing values. Therefore, they must be dealt with before further analysis. There are three main approaches of handling missing data [117]. One strategy is to discard all data items containing missing values. Thus, only healthy data is considered for analysis. If, however, many items have missing values this approach may result in discarding most of the data. The second strategy is to substitute a missing value with the mean value of the data item (i.e. substituting missing information about the age of a machine with the mean age of all machines). This strategy implies a normal distribution of the data. The third strategy is to estimate the missing value based on other existing values (i.e. estimating the age of a machine based on its serial number). Statistical estimation techniques such as regression are often utilized for calculating missing values.

Even if one of the strategies is applied, missing data can still be a problem. Each strategy can introduce bias into the analysis, resulting in misleading or wrong conclusions [117, 118].

4.3.2 Data Analysis

Data analysis describes techniques to analyse the data in order to gain insights and deeper understanding. Depending on the specific data type, different analyses are available. As mentioned above, the three data types are *value*, *waveform* and *multidimensional* data [104].

Analysing Value Data

Value data is the simplest form, consisting of at least one item. To get a better understanding of a single data item basic statistical analyses can be applied. This includes calculating features like mean, standard deviation, variance, and min-max values. Another analysis technique is visualizing the data. Visualization allows for a comprehensible quick look at data to get a feeling for the overall value distribution, to detect outliers and trends, and to discover relationships [104].

The complexity of analysing value data increases with the number of variables. Beyond examining every variable for itself, correlations between variables can be analysed. One of the most popular techniques is the regression analysis, where one dependent variable is defined as a function of multiple independent variables. As with single variable data, data containing multiple variables can be visualized (i.e. a graph visualizing the development of variables over time). A modern approach for analysing multi dimensional data is utilizing unsupervised machine learning algorithms [104]. Machine learning describes a class of algorithms that is able to make predictions based on existing data. They “learn” data correlations from a set of input data. Generally, machine learning is divided into supervised and unsupervised learning [119]: Supervised learning techniques are used to make predictions, where all possible outcomes are known up front. A supervised learning algorithm is first trained on a data set, where possible outcomes are labelled. After training, the algorithm, it is able to make predictions on unlabelled data. For instance, a supervised machine learning algorithm is trained on multiple images showing cats and dogs. Afterwards the trained algorithm is able to detect whether a new image shows a cat or a dog but unable to identify other animals like bunnies because it has not yet learned them. Unsupervised learning algorithms do not have to be trained. They take in raw data and draw conclusions based on correlations they find. For instance, an unsupervised learning algorithm can find images showing similar content. Unsupervised machine learning algorithms are therefore best suited for data exploration [119].

Large amount of variables leads to high complexity in further analysis. Especially in prediction models, too many variables result in long execution times and can even negatively affect the accuracy of the outcomes [120, 121]. In order to reduce complexity, dimensional reduction techniques are utilized. The basic idea is to remove variables with no or little descriptive benefit. The simplest approach is leaving out data columns with a large amount of missing values since they do not convey a high amount of information. Similarly, variables with a very small variance have limited impact. Another way is to analyse correlations between variables. If at least two of them have high correlation and follow the same trends, they likely carry similar information and can be reduced to only one. In addition to these simple techniques, more sophisticated methods are available. Principal component analysis (PCA) is the most popular method for dimensional reduction [121, 122]. PCA transforms the data into in an equal or smaller amount of uncorrelated data. It uses the most expressive features to approximate the data [121].

Analysing Waveform Data

Many data such as vibration analysis or acoustic data are recorded as waves. Waves convey a lot of information, but their analysis is complex. Wave analysis can be split into three main categories: time domain analysis, frequency domain analysis and time frequency analysis [104].

Time domain analysis applies statistical methods directly to the wave itself. It analyses changes to the wave over time. Typical features are mean, peak, peak-to-peak interval, standard deviation, crest factor, skewness and kurtosis [104].

Frequency-domain analysis is not based on time but on frequency. Instead of analysing changes to the signal over time, frequency-domain analysis concentrates on how much of the frequency lies in between specific frequency bands [104].

Time-frequency analysis combines the two approaches by investigating waves both in the time and frequency domain. Waves are represented as two-dimensional functions of time and frequency [104].

Figure 23 illustrates the time and frequency dimension of a wave as well as their relationship. The conversion between the two is done by Fourier transformation. As displayed below, the Fourier transformation decomposes a given wave, which was measured over time, into separate waves of consistent frequency [104].

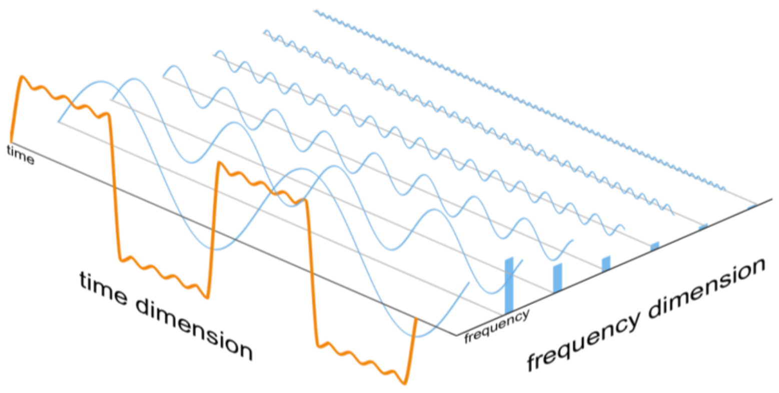


Figure 23: Illustration of the relationship between time and frequency dimensions of a wave as established by the Fourier Transformation (based on [38]).

Figure 23 illustrates the time dimension as orange line, which displays the changes to the wave over time. The blue bars in the frequency dimension show the various frequencies the wave is composed of.

Analysing Multidimensional Data

Multidimensional data like images are complex to analyse. Sometimes raw image data provides enough information to identify patterns. If this is not the case, image processing techniques have to be applied to extract useful information [104] .

Image processing techniques have found their way into the manufacturing industry and have proven to be an excellent data source for data analysis and decision making systems. Oikawa et al. [123] designed a system based on image and sound data to detect oil and steam leaks as well as fire and smoke in the vicinity for a thermal power plant. Demant et

[38] http://pgfplots.net/tikz/examples/fourier-transform/

al. [124] show how image processing can be used to implement a visual quality control system in a manufacturing plan. Connolly [125] introduces many more application areas for infrared images in the manufacturing industry like identifying heat leaks and product defects.

4.4 Maintenance Decision Support

After collecting and analysing data, the final step of a predictive maintenance system is to decide whether a machine has to be serviced or not. The decision is highly dependent on the previous steps. Independent of the chosen method, analysis can only be meaningful when performed on significant quality data. Maintenance decision making techniques can be separated into two categories: *diagnostics* and *prognostics* [104]. Fault diagnostics focuses on the detection, isolation, and identification of faults. Prognostics builds upon diagnostics and tries to predict failures before they occur [104].

4.4.1 Failure Diagnostics

Machine diagnostics is the process of analysing data to diagnose the state of a machine. Its goal is to recognize if a failure is currently present. One of the main tasks is to identify patterns indicating failures. These patterns can then be used to detect failures inside of a machine without manual inspection. Analysing faults and investigating their causes is often done by experts of the respective fields. This requires qualified personnel and results in time and monetary costs. To speed up the process and to make it universally applicable, automated diagnostic approaches have been developed. Diagnostic techniques can be generally categorized into three types: *statistical*, *artificial intelligence* and *model based* approaches [126].

Statistical Approaches for Machine Diagnostics

Statistical approaches utilize statistical methods to diagnose machine failures. They analyse and compare machine data to previous measurements to detect the current state of the machine. Common statistical methods include hypothesis tests, cluster analysis, and hidden Markov models.

Hypothesis Test

The problem of detecting a specific fault inside a machine can be described as a hypothesis test problem with the following hypothesis [104]:

H_0: Fault *x* is present

H_1: Fault *x* is not present.

In order to test the hypothesis, current measurements are compared to measurements taken during normal behaviour.

Nyberg [127], for example, introduces a framework of structured hypothesis tests for automated fault diagnostics. He splits the diagnostic problem into multiple hypotheses following the above-mentioned design. After testing for each hypothesis, he combines the results logically to detect which failures are able to describe the current state.

The key to hypothesis tests lies in careful definition of possible failure states as well as the presence of sufficient data measured during them. The initial development of the statistical model therefore requires expert knowledge and has to be thoroughly tested against reality [127].

Cluster Analysis

Cluster analysis is another statistical method to detect machine failures. The basic idea is to group similar data points together into fault categories. New measurements are then compared to these groups and are put into the ones they are most similar to [128].

Consider Figure 24 as an illustrated example. The first graph (left) plots two-dimensional data consisting of a x- and y-value. Green points indicate measurements taken during normal working time of the machine, red points indicate measurements during failure occurrences. A clustering analysis then groups similar data together into two groups: the failure and non-failure group (middle). To diagnose the current state of the machine, the new data point (orange) is analysed and compared to both groups (right): The new orange data point is most similar to the failure group. Hence the current state of the machine is diagnosed as failure.

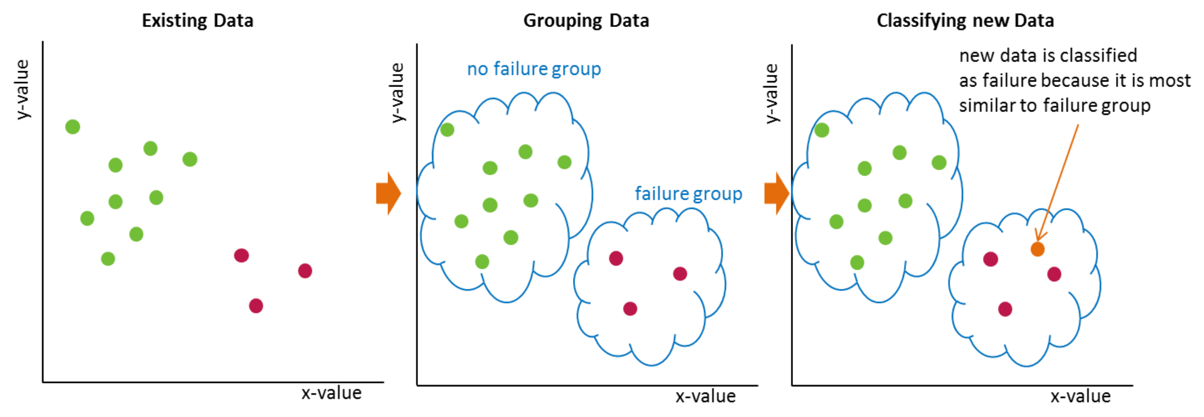


Figure 24: Example of failure identification by cluster analysis.

There are multiple ways to define groups. A common technique is to cluster points together, which have the smallest distance to each other. There are however many types of distances that can be utilized. Examples include Euclidean distance, Mahalanobis distance, Kullback–Leibler distance and Bayesian distance [104].

Hidden Markov Model

Named after the Russian mathematician Andrey Markov, the Markov process is mathematical model to represent a stochastic system. The system consists of states and transitions between these states. The Markov model predicts the next state of the system. The future state is only dependent on the present one. Past states and transitions are not considered [126]. In a hidden Markov model some, or all states are not directly, but indirectly observable. The states are therefore hidden from the observer [126].

Figure 25 illustrates an example for a hidden Markov model of a machine. The machine either runs normal (Fault-free) or experiences one of two possible failures (Failure 1 or Failure 2). The hidden Markov Model below models these three states of the machine. Every state can transition into each of the others as represented by the grey dotted arrows. The orange arrows show one possible sequence of states. Each transition has a probability p. The goal of the hidden Markov model is to estimate the most likely system sequence based on the current state at the present time T_k.

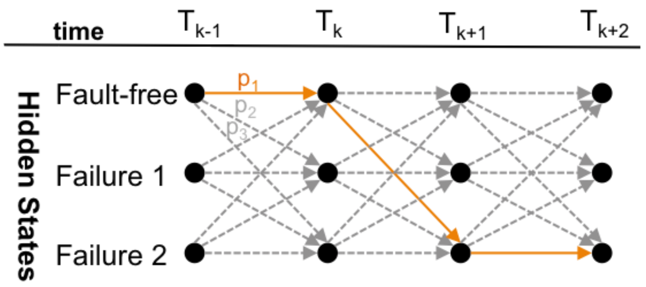


Figure 25: Illustration of a hidden Markov model consisting of three hidden states and four points in time.

Many experiments have been performed to test the application of hidden Markov models in the field of predictive maintenance. Ying et al. [129] introduce a hidden Markov based algorithm for fault diagnosis in systems with partial and imperfect tests. They model failures as hidden states and calculate the transition probabilities by the well-known Baum–Welch algorithm [130]. Tai et al. [131] explore the application of a hidden Markov model to detect machine failures in a production environment based on the quality of the manufactured products. Li et al. [132] show that a hidden Markov based fault diagnostic model can effectively predict failures during the speed-up and speed-down process of large rotating machinery (i.e. turbines in a power plant) based on vibration data. And Wu et al. [133] propose a real-time condition monitoring system based on acoustic sensors and a hidden semi-Markov model, yielding positive results in the detection of common machine failures.

Artificial Intelligence Approaches for Machine Diagnostics

Artificial intelligence describes a group of techniques that try to simulate intelligent behaviour. In the field of failure diagnostics, artificial neural networks, expert systems, and fuzzy logic systems are the most popular methods.

Artificial Neural Networks

Artificial neural networks have their origins in studies of the human brain [134]. They try to mimic the activities of the brain, where millions of interconnected neurons process information in parallel. The idea of neural networks for processing information was born in the middle of the 20th century. The first model was developed in 1943 by McCulloch and Pitts [135]. In 1954 Minsky [134] introduced the first working prototype Snark, consisting of 300 vacuum tubes and 40 variable resistors. It could be trained to run a maze. Although the research on artificial neural networks continued on, it really gained momentum in the 1980s with the rise of modern computers.

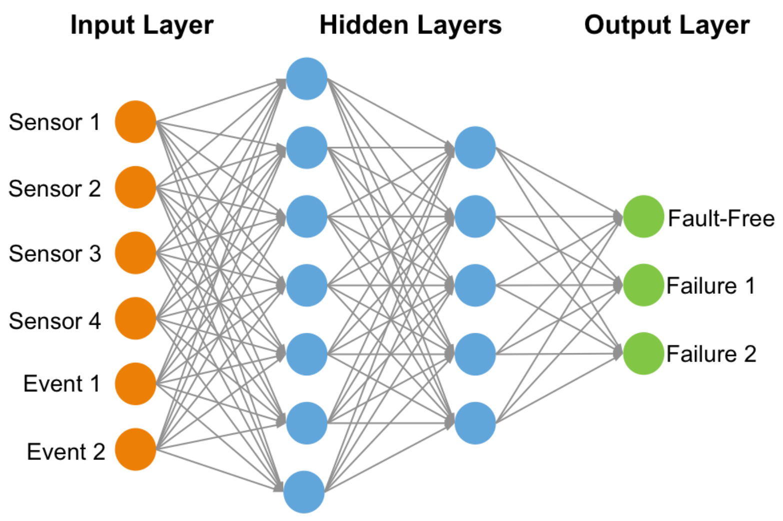


Figure 26: Structure of an artificial neural network with one input layer, two hidden layers and one output layer.

An artificial neural network transforms multiple inputs into multiple outputs. It utilizes interconnected nodes as processing units. Figure 26 displays the structure of an arbitrary artificial neural network. It consists of three main parts: the input layer, the output layer, and one or more hidden layers in between. Each layer consists of multiple nodes. Each node is connected to all nodes of the previous and the next layer. Each connection has a weight assigned to it. The weight represents how strong nodes are connected and can be a positive as well as a negative value.

Each node has a value. The value is calculated as the sum of all nodes of the previous layer multiplied by the weight of the connection to it. This calculation is done for each layer. The result is given by the node in the output layer with the highest value. The performance of the algorithm highly depends on set weights of the connections. They are usually set during a learning phase, in which the algorithm is fed with training data. The training data consists of multiple data items that are already labelled with the correct outcome.

The possible application of artificial neural networks to the issue of failure diagnostic is explored in multiple occasions. He and Li [136] use ultrasound data and an artificial neural network to successfully diagnose the condition of grinding machines. Verma et al. [137] introduce a system detecting faults of an air compressor based on acoustic data. Soliman et al. [138] utilize an artificial neural network to estimate current capacity of DC-link capacitors. Their input data includes in-/output current/voltage as well as loading power and DC-link voltage. Their results indicate that their model was able to detect even very small changes of capacity.

While artificial neural networks can be very accurate, Gowid et al. [139] point out that they bear high computational as well as development costs. Gowid et al. [139] compare the application of neural network with a fast Fourier transformation based segmentation algorithm to condition monitoring of centrifugal equipment.

Expert Systems

Instead of learning from historic data, expert systems use domain expert knowledge for problem solving. Inference engines are used to transform inputs into conclusions. In the field of machine failure, the most commonly used reasoning techniques are rule-based reasoning, case-based reasoning, and model-based reasoning.

In general an expert system is composed of a man-machine interface, an interpreter, a reasoning machine, a knowledge acquisition module and a knowledge base [140]. Figure 27 display its architecture and interaction.

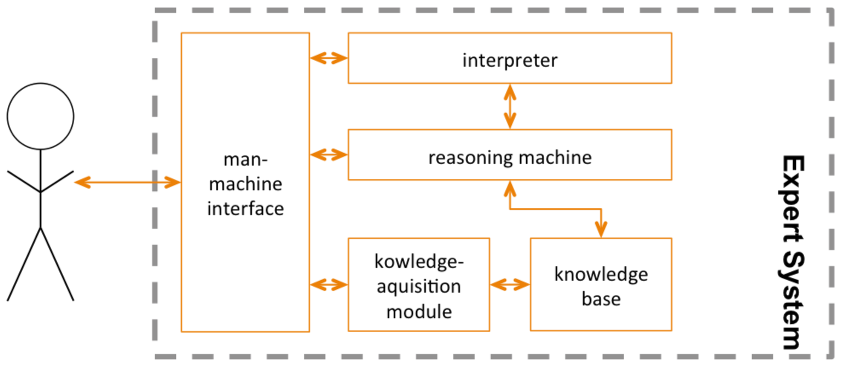


Figure 27: General architecture of an expert system.

During development of the system, domain experts fill the knowledgebase through the knowledge acquisition module. Depending on the reasoning method, this can be in form of rules, cases or an underlying model. New input by users is processed by the reasoning machine, which applies rules extracted from the existing knowledge in the knowledge base. The result is presented to the user through the interpreter, explaining all reason leading to the conclusion [140].

The feasibility of expert systems in the field of machine diagnostics is examined multiple times. Deng et al. [140] present a rule-based expert system to monitor the condition of wind turbines. In addition to identifying the state of the machine, the expert system advises the user on how to restore functionality. Gao et al. [141] developed an intelligent fault diagnostic system for gearboxes of rolling mills in the steel industry. Their hybrid reasoning machine uses a combination of rules and cases to generate a fault diagnostic report. Wen et al. [142] apply case-based reasoning to vehicle fault diagnostic. Their system is able to find root causes of vehicle faults. Their experiments with real data show its accuracy and effectiveness. Stanek et al. [143] use a combination of model and case based reasoning to diagnose high-voltage switching devices. Their model simulates the behaviour of the devices based on the input data. The case based reasoner complements the results of the model to get the final results.

Fuzzy Logic Systems

First mentioned by Zadeh in 1965 [144], the main goal behind fuzzy logic is to describe a system that is capable of dealing with classes that do not have precise defined criteria. Traditional logic knows only two conditions: true or false. In the real world we often encounter situations where this binary classification leads to problems. Fuzzy logic does not limit itself to true and false but utilizes a sliding scale between 0 and 1.

Consider the classical example of water temperature. In a traditional logic system the water has only two states: hot and cold [145]. At very high or low temperatures the classification into either one of these states is simple. The problem arises when defining the transition point at which the state changes from cold to hot. The question is, if there is really one point where all higher temperatures can be considered as hot and all lower ones can be considered as cold. Fuzzy logic helps to define the grey area between hot and cold. Instead of having two absolute values, a value between one and zero is assigned to each state. Figure 28 shows a possible temperature function in fuzzy logic. In this example the three states cold, warm and hot are possible. Each state is described by a membership function. The membership function defines the value at a specific temperature. For example, in point P, the membership functions of both the cold and the warm state assign a value of 0.5 each.

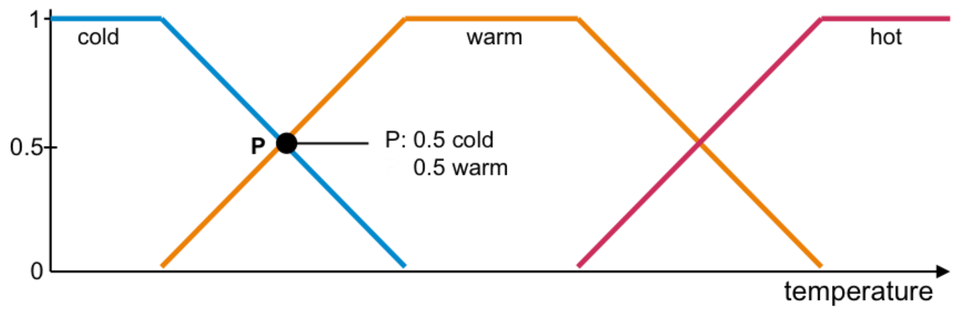


Figure 28: Fuzzy logic system modelling temperature.

Fuzzy logic can be utilized to diagnose machine faults as well. Instead of cold, warm and hot, measurements are classified into groups like low, normal, and high. Rules are then applied on these outcomes to reach conclusions. Fuzzy logic is also often coupled with inference engine. Mechefske [145] analyses the application of fuzzy logic for fault diagnosis based on vibration data. He experiments with different shapes of membership functions, namely linear, triangular, S-, and π-curve, with the last one yielding the best performance. Noreesuwan and Suksawat [146] use fuzzy logic and sound analysis to monitor the health of groove ball bearings. They present nine fuzzy rules to accurately determine the condition of the bearings. Hichem et al. [147] propose a system for detecting stator windings faults in induction machines. They combine fuzzy logic with an inference engine consisting of 14 rules to determine one of four possible machine states.

The challenge of developing an effective fuzzy logic system is defining the membership function as well as the rules applied to measurements. They can be either set manually or calculated analysing historic data. Especially manual configuration bears the risk of introducing bias into the system.

Model Based Approaches for Machine Diagnostics

Model based techniques utilize physical and explicit mathematical models of the machine to diagnose their current state [148]. The model simulates the behaviour of the physical machine and is used to predict its future condition. Figure 29 illustrates the workflow of a model based approach for failure diagnostics. It displays the physical, real system as well as the mathematical model of this system. Both, the system and the model are fed with input data (i.e. how fast a rotor should spin). The model simulates how the real system should behave under perfect conditions. Sensors monitor the real system and measure how the system actually behaves. The simulated data and the actual sensor data are then compared to find discrepancies. Analysing these discrepancies can lead to the detection of failures within the real system.

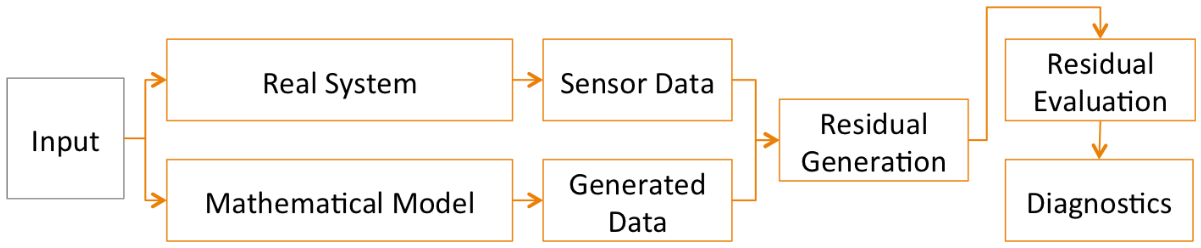


Figure 29: Workflow of a model based approach for failure diagnostics.

The mathematical model must be very precise to simulate the real system accurately. Developing the model is time-consuming and requires expert knowledge. Furthermore a model is only applicable to the type of machine it was designed for [148]. However, once constructed, they are very accurate in simulating the correct behaviour [148]. The complexity of the model increases with the number of its components. Most research available targets single components rather than entire machines. For instance, Bartelmus [149] presents a mathematical model for a one-stage and two-stage gearbox. The gearbox is simulated on a computer and compared to real-life sensor data to infer the current condition of the gearbox.

4.4.2 Failure Prognostics

While diagnostics focus on assessing the current condition of a machine, prognostics aim at predicting the machine's future condition. As illustrated in Figure 30, there are two main prediction types [104]: Predicting the remaining useful life (RUL) of a machine and predicting component failures.

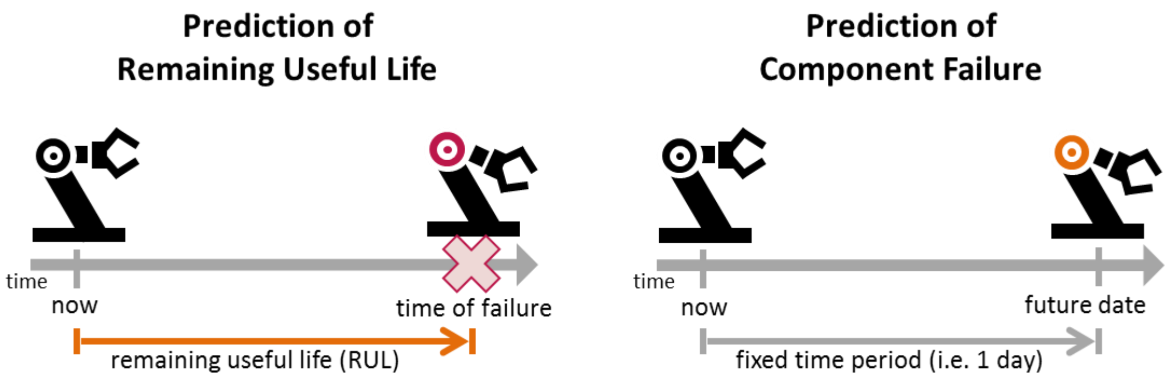

Prediction of Remaining Useful Life
time
now
time of failure
remaining useful life (RUL)
Prediction of Component Failure
time
now
future date
fixed time period (i.e. 1 day)

Figure 30: Prediction of RUL vs prediction of component failure.

Predicting the RUL of a machine is calculating how much time is left until a failure occurs. The problem can be formulated as follows: "*How much time will pass until machine X will fail*?" The result of this prediction is a number representing the time until failure. Predicting component failure calculates if a specific component will fail in a fixed time period, resulting in the problem statement "*Will component Y of machine X fail in the next 24 hours?*". Depending on the technique, this prediction can have two different outcomes: Either the result is a simple binary yes or no, or the result is a percentage representing the probability of component failure.

In general, both prediction types utilize similar techniques to diagnostics, which were already presented in chapter 4.4.1.

Predicting Remaining Useful Life

The RUL represents the time until a machine will fail. Calculating the RUL enables servicing the machine just in time before it would break down. RUL can be predicted on a component, machine, or system level (i.e. RUL of a bearing, engine, or car). The RUL can be represented in two main ways. It can be indicated in a time unit, for example hours until failure. This time unit is sometimes accompanied by a percentage indicating the certainty of the prediction (i.e. the component has a RUL of 50h with a certainty of 85%). Or the RUL can be presented in form of degradation, for example the component has been worn down to 50%. Similar to diagnostics, RUL prediction falls into the three categories of statistical approaches, artificial intelligence approaches, and model-based approaches.

Statistical Approaches for Predicting RUL

Statistical approaches to RUL prognostics utilize statistical methods to predict the time until failure of a machine. They compare current data of a machine to historic measurements to calculate the RUL. Typical statistical methods include the already discussed techniques of hypothesis tests and hidden Markov models. Besides them, regression is one of the main methods to determine the RUL.

Goode et al. [150] use a traditional statistic approach to solve the problem of determining the RUL. They utilize statistical process control to separate the whole machine life into the two intervals I-P (Installation - Potential failure) and P-F (Potential failure - Functional failure). While the machine is running correctly in the I-P interval, it runs with a problem during P-F. The current interval of a machine is predicted using a Weibull distribution and RUL is estimated. Another example is presented by Li et al. [151], who predict the RUL of rolling bearings based on R/S Statistic and fractional Brownian motion. The fractional Brownian motion is a mathematical function representing a continuous series of data. It is a semi-random continuous stochastic process. R/S Statistic is one of the oldest methods for estimating the Hurst-index, which can be used to describe the trend within a time series such as the fractional Brownian Motion [152]. The prognosis of Li et al. [151] is based on vibrational data and predicts the degradation status of rolling bearings.

As mentioned before, the Markov model is based upon states and the transition between them. Each transition only depends on the current state of a machine. Markovian-based models can be used to estimate the time passed between the current and a future state, for example *operating* and *broken* [153]. The model can be further refined by adding multiple states (i.e. *excellent*, *good*, *minor defects*, and *critical failure*). The main limitation of Markov-based models is their central assumption of independence of past states. To estimate RUL, Markovian models only consider current values. Banjevic and Jardine [154], for example, estimates RUL of a transporters transmission based on a Markov failure time process. They calculate RUL as a function dependent on machine age and current condition data derived from oil analysis.

Another method to estimate RUL is regression. Regression is a statistical process for estimating one (dependent) variable based on the values of one or more (independent) variables. Consider the following linear regression model. Linear regression is a special form of regression, where the relationship between the independent variable and the dependent variable is represented by a linear function.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_4 + \beta_4 x_4$$

In this function, the dependent variable y is represented as a function of the four independent variables x_1-x_4. β_1- β_4 are the unknown coefficients for the values of the independent variables and β_0 is the unknown constant. The unknown coefficients β_0 - β_4 are estimated by analysing historic data, where y and x are known. In the case of predictive maintenance, x_i represents the current condition of a machine and y the RUL. Regression based methods are widely used in industry and academic research due to their simplicity [153]. However, this simplicity creates some problems: Regression assumes a monotonic degradation process, which cannot always be observed in real life [153]. Regression also cannot model temporal variability and uncertainty in the degradation process [155].

Caesarendra et al. [156], for example, utilize logistic regression (a special form of regression used to calculate the percentage if an event will occur or not) to calculate the degradation status of bearings based on vibration data. Yan et al. [157] also applied a logistic regression model to determine RUL of an elevator door motion system. Kehlif et al. [158] use a regression model to predict RUL of the Turbofan machine based on a data set provided by NASA.

Artificial Intelligence Approaches for predicting RUL

Most of the artificial intelligence approaches for predicting RUL are based upon the artificial neural network (see chapter 4.4.1). While the input data to the neural network is the same as in diagnostics (current condition data of a machine), the output differs. Instead of outputting the current health of the machine, the neural network is trained to predict RUL. For instance, Wang and Vachtsevanos [159] predict the fault propagation process using a artificial neural network and estimate RUL as the time left before the fault propagates to a certain level. Another example is provided by Asmai et al. [160]. In a first step they predict the failure probability using a logistic regression model. In a second step, the result of the logistic regression is fed into an artificial neural network, predicting RUL.

While artificial neural networks can achieve good prediction results with only a few samples to train on [159, 161], they can take a long time to train due to a high amount of hidden layers and data points [162].

Model Based Approaches for Predicting RUL

Similar to model based approaches used for machine diagnostics (see chapter 4.4.1), model based approaches for failure prognostics build a mathematical model of a physical system. They predict RUL by simulating future behaviour on the theoretical model. Model based approaches require specific mechanistic and theoretical knowledge of the system of which the RUL is estimated. This results in long development periods and the need for experts. Models are also only applicable to one specific machine and cannot be easily applied to others. Ray and Tangirala [163], for example, developed a non-linear stochastic model to predict crack dynamics in order to estimate RUL. Li et al [164, 165] introduce a stochastic model to simulate defect propagation in bearings with the goal of calculating RUL.

Predicting Component Failures

Predicting component failures formulates the prediction problem differently than predicting RUL. Instead of estimating the average time until failure, a prediction is made whether a specific component will fail in a specified time horizon (i.e. 24h). The answer is a binary yes or no instead of a numerical time value. The binary result is sometimes accompanied by a percentage representing the certainty of the prediction. Predicting component failure is typical solved by classification machine learning algorithms. Classification algorithms identify to which class a certain set of values belong. In case of predictive maintenance, the classes are for example "*will fail*" or "*will not fail*". The values for the prediction are given by current condition data of a machine. The following sections introduce the most prominent machine learning algorithms for classification problems: decision tree, decision forest, naïve Bayes, artificial neural networks, and support vector machines.

Decision Tree and Decision Forest for Predicting Component Failure

The basic idea of decision trees is to break up a complex decision into a union of several simpler decisions [166]. Instead of making one decision, several small decisions are made to come to a conclusion. The structure of the possible decisions resembles a tree.

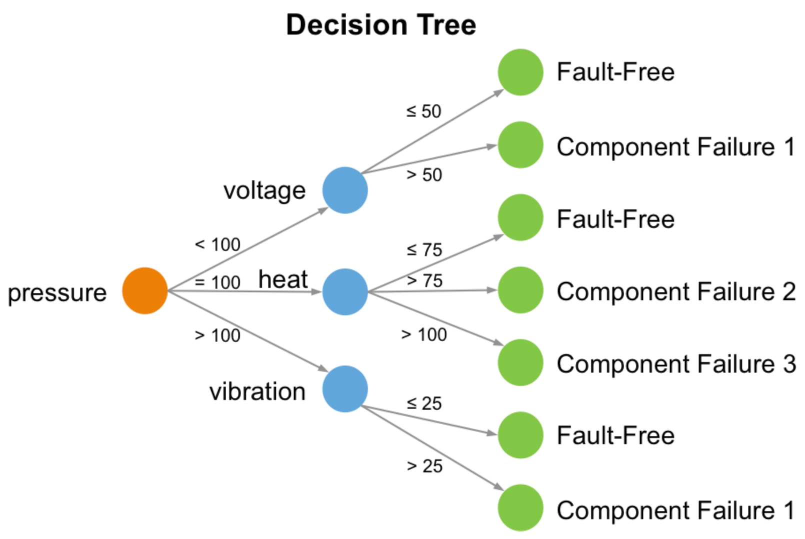


Figure 31: Example of a decision tree for predicting component failures.

Figure 31 illustrates an exemplary decision tree from the domain of predictive maintenance. It consists of one root node (orange), internal nodes (blue), and leaf nodes (green) holding the final prediction [167]. At each node a decision is made revealing the path to the next node. Consider a machine with four different sensors measuring pressure, voltage, heat, and vibration. The measurements of these sensors can indicate imminent component failure. Instead of considering all measurements at once, the decision tree splits the problem into multiple smaller decisions. Starting at the orange root node, at first only the pressure is considered. Depending on its value, a different path is chosen for the next node (in this case <100, =100, >100). At the subsequent node the next variable is considered (voltage, heat, or vibration) and leads to the final green leaf node holding the class prediction.

In machine learning a data set is used to create such a decision tree. The tree is recursively partitioned until all data items of the same class belong to the same label [168]. This process is called the training of the decision tree. Training is a computationally very expensive task, since the data set is traversed multiple times [167]. However, after training, the decision tree model is computational very efficient since new data is only checked for a few attributes when traversing through the tree [166]. One of the biggest advantages of the decision tree is its transparency regarding decision making. Decision trees produce understandable results since all steps leading to the final decision are made due to transparent rules.

The decision forest is an expansion of the decision tree. Instead of having only one decision tree, multiple trees are trained. The trees are uncorrelated and differ due to some random variable considered during training. Figure 32 illustrates an example for a decision forest made from three different decision trees. In a first step, new data is classified by every single tree. Afterwards, in a second step, a majority vote is used to conclude the final classification.

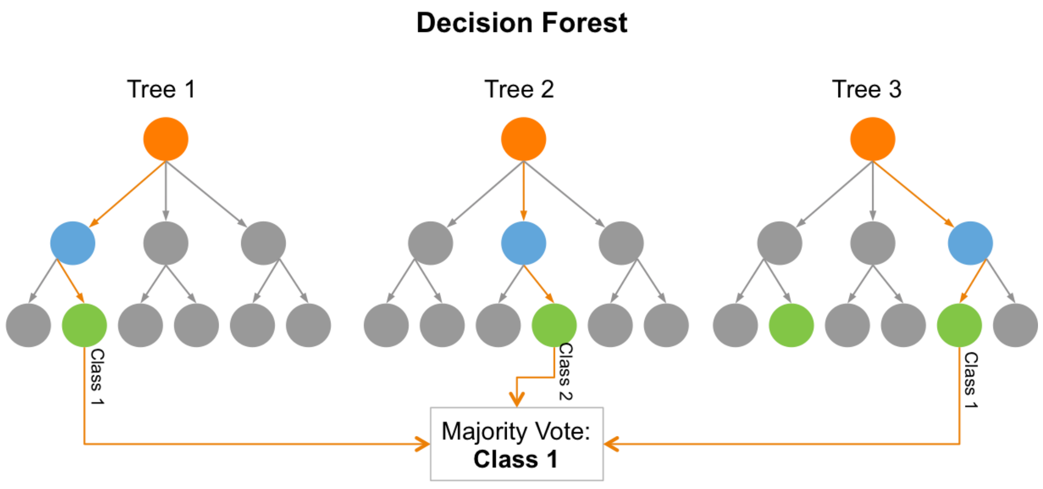


Figure 32: Example of a decision forest consisting of three decision trees.

Decision trees and forests have been employed to predict machine failures multiple times. For instance, Sylvain et al. [169] use a decision tree to predict aircraft component replacements. Their goal is to predict whether a specific component should be replaced within a given time period or not. The input data was gathered from an Airbus A-320 and the experiment was conducted for 16 different components with positive results. Bonissone and Goebl [170] present a model combining a neural network and a decision tree to predict imminent failure of a paper web in a paper mill. The paper web transports the paper through the process with speeds up to 60 mph. Due to high stress, the web breaks on average once per day, resulting in standstill of the entire machine for up to 90 minutes, leading to revenue losses of several million dollars per year. The model proposed by Bonissone and Goebl predictis imminent failures to enable personnel to take preventive actions. Guang et al. [171] propose a decision forest model to predict failures in a cloud computing system. They collect 83 runtime performance metrics from two clusters containing 166 servers each. The data is then used to train a decision forest to predict component failures within the cluster.

Support Vector Machines for Predicting Component Failure

First proposed for by Cortes and Vapnik [172], support vector machines (SVM) are a tool for classification in data analysis. Figure 33 shows the basic concept of SVM with two-dimensional data. In this figure, data points are plotted against a field. They are either a member of class *working* (green) or *failure* (red). The SVM then calculates a boundary between them (blue). The goal of the boundary is to separate both classes in such a fashion that it has the maximum distance to points of both classes, placing it right in the middle of them [172]. To classify a new data point (orange), it is plotted on the same plain. Depending on which side of the boundary the new data point resides, a classification is made. In the illustrated case, the new orange data point would be classified as *failure*.

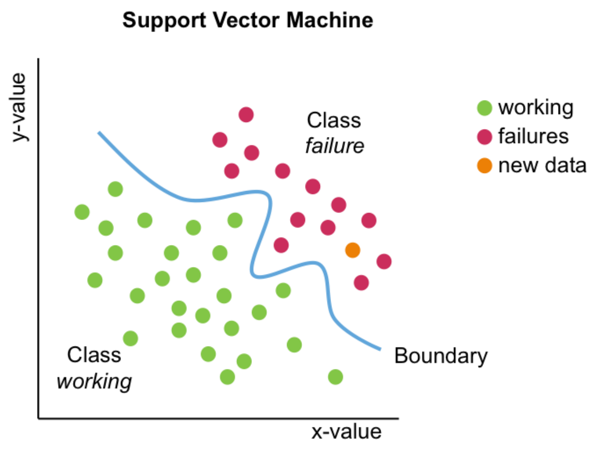


Figure 33: A Support Vector Machine for classification of two classes.

SVMs are able to achieve high accuracy on complex classification problems, but they are computational very expensive to train with increased dimensionality of the data [173]. There have been some experiments to use SVM to predict machine failures. For instance, Susto et al. [173] utilize multiple SVMs to predict failures of an ion-inducing machine used in the semi-conductor industry. The machine is critical to the process and considered a bottleneck due to its high costs. The tool utilizes a tungsten filament, which has to be replaced frequently, disabling the machine for up to three hours and slowing down the entire production line. The model proposed by Susto et al. determines the optimal time for the replacement of the tungsten filament.

Nearest Neighbour for Predicting Component Failure

Nearest neighbour is a method to classify data by finding the most similar known data points. Figure 34 illustrates this technique. New data (orange) is compared to all other data points, whose class is already known (in this case red for class *failure* and green for class *working*). The new data is classified the same class as its nearest neighbour (in this example *failure*).

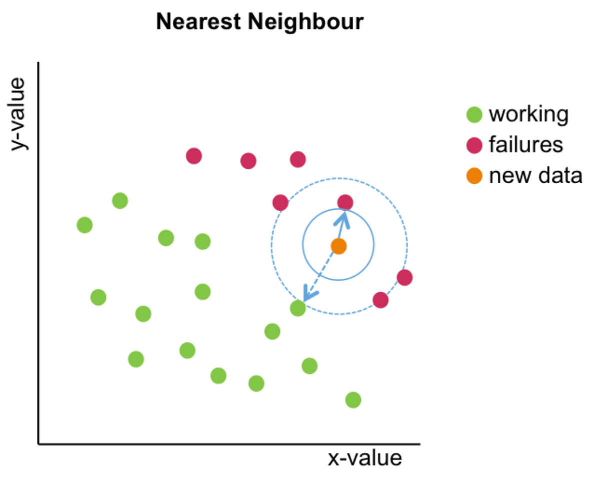


Figure 34: Nearest Neighbour algorithm for classification of two classes.

By considering only the nearest neighbour, the method is not aware of outliers in the training data. Therefore, instead of finding only the nearest neighbour, multiple close neighbours are considered. This approach is generally called k-nearest neighbours. Nearest Neighbours is one of the simplest classification algorithms, requiring just the computation of distance between samples [173]. No explicit training of the model is necessary. However, this leads to expensive calculations each time a new data point is classified. Thus, it is not surprising that nearest neighbour approaches have not yet been heavily researched for the domain of failure classification. Verma and Kusiak [174] give an example by using a k-nearest neighbour approach to predict failures of generators and blades of wind turbines. They compare its performance to implementations using a decision tree, support vector machine, and genetic algorithm, resulting in k-nearest neighbour performing worse than any other algorithm.

Naïve Bayes for Predicting Component Failure

The Naïve Bayes classifier is a simple probabilistic classifier with strong (naïve) independence assumptions [175]. It classifies a set of variables based on already classified data. The classifier strongly assumes that all variables are independent of each other. Although this assumption is generally not realistic, the Naïve Bayes classifier can perform surprisingly well even when classifying data with highly dependent variables [176]. Using the Naïve Bayes Classifier, the probability of a class c given the variable x can be expressed as follows:

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$$

P(c|x): the probability of class c given the attribute x

P(x|c): probability of variable x given the class c

P(c): the probability of class c

P(x): probability of variable x

With n variables X, the probability of class c is given by the product of the individual variables x_i:

$$P(c|X) = P(c|x_1) \times P(c|x_2) \times \ldots \times P(c|x_n)$$

Consider following example for clarification. A machine periodically measures pressure and heat. For simplification, the measurements are either *high* or *low*. Table 2 lists ten observations made of the machine. Each observation consists of the pressure and heat measurements as well as a binary yes/no, indicating if a failure was present at the time. Table 3 lists the frequencies of failures depending on the individual measurements. They are separated by variable. The table also lists the likelihood of failure given a specific measurement.

Table 2: Observations of pressure and heat and corresponding state of the machine.

Observations

Pressure	Heat	Failure
low	high	yes
low	low	no
high	low	no
low	low	no
high	high	yes
low	high	yes
low	low	yes
low	high	no
low	low	no
high	low	yes

Table 3: Tables of frequencies and likelihood of failures.

Frequency Tables and Likelihood of Failure

Pressure	Failure	No Failure	Likelihood
low	3	4	=3/10
high	2	1	=2/10
Sum	**5**	**5**	
	=5/10	=5/10	

Heat	Failure	No Failure	Likelihood
low	3	1	=3/10
high	4	2	=4/10
Sum	**7**	**3**	
	=7/10	=5/10	

Now, consider a new observation with high pressure and low heat. The question is whether the observation will likely result in failure or not. Using the formula above and the values calculated in Table 3, the probability of failure given the new observation is calculated as follows. At first, the probability of failure given the high pressure is calculated:

$$P_{pressure}(yes|high) = \frac{P(high|yes)P(yes)}{P(high)} = \frac{0.4 * 0.35}{0.5} = 0.28$$

P(*high*|*yes*) = 2/5 = 0.4
P(*yes*) = (5/10)*(7/10) = 0.35
P(*high*): 5/10 = 0.5

Afterwards, the probability of failure given the low heat measurement is calculated:

$$P_{heat}(yes|low) = \frac{P(low|yes)P(yes)}{P(low)}$$

P(*low*|*yes*) = 3/7 = 0.43
P(*yes*) = (5/10)*(7/10) = 0.35
P(*low*): 3/10 = 0.3

$$= \frac{0.43 * 0.35}{0.3} = 0.5$$

Multiplying both probabilities results in the overall probability for the observation of being in the class failure with 14%:

$$P_{Observationm}(yes|X) = P_{pressure}(yes|high) \times P_{heat}(yes|low) = 0.28 \times 0.5 = 0.14$$

There is only a limited amount of research available utilizing a Naïve Bayes classifier for predicting machine failures. Di Maio et al. [177], for example, use a Naïve Bayes classifier to predict failures of bearings. They use data collected by vibration sensors and classify the data to find the most similar degradation process.

Artificial Neural Networks for Predicting Component Failure

The artificial neural network as portrayed in chapter 4.4.1 can also be used to predict component failures. Instead of training the neural network to predict the current state, the network is trained to predict future component failures. Therefore, the training data set is labelled with future component failures instead of current state.

Bangalore and Tjernberg [178] use a neural network to predict failures of gearbox bearings from wind turbines. They collect the average temperature of the bearings every 10 minutes and use a neural network to predict its future development. Based on this prediction an assessment is made whether the bearing will fail or will continue to work. They tested their model with data from offshore wind turbines and were able to detect severe damages of gearbox bearings in advance.

5 Predictive Maintenance Benchmark (PMB)

This chapter introduces the Predictive Maintenance Benchmark (PMB). The PMB is a tool to compare Big Data analytic ecosystems in the domain of predictive maintenance. It follows an end-to-end approach and tests an entire ecosystem rather than single Big Data analytic frameworks.

Due to the broad employment of Big Data analytics across industries [179], it is important to compare Big Data analytic frameworks in context of their application area [12, 13]. Today's end-to-end Big Data benchmarks focus on the domains of retail, e-commerce, search engines, or social networks [14, 15]. PMB expands the available benchmarking domains into the field of predictive maintenance. It is based on a predictive maintenance use case, in which a machine learning algorithm is used to predict future component failures.

PMB is developed following the methodology of Han et al. [12]. As described in chapter 1.4, the methodology separates the development of a benchmark into the five steps of planning, data generation, test generation, execution, and analysis and evaluation. This chapter covers the first three steps of the methodology, while execution and evaluation are completed in chapter 6.

5.1 Planning of the PMB

During planning, the benchmark object, application domain, and evaluation metrics are determined [12]. As mentioned before, the goal of PMB is to test an end-to-end process in the not yet covered domain of predictive maintenance. Therefore, the object of PMB is a Big Data analytic ecosystem and the application domain is predictive maintenance. In accordance to the popular end-to-end Big Data benchmarks BigBench [14] and BigDataBench [180], as well as many other benchmarks [11], execution time is the main performance metric of PMB.

The workloads of PMB are based on a predictive maintenance use case, which is described below in section 5.3. The use case consists of the three steps data acquisition, data processing, and maintenance decision making (see chapter 4.2-4.4). The data is collected from multiple sources, including periodic condition monitoring data (i.e. vibration data), event based data (i.e. occurrences of errors), and additional information (i.e. machine age). The data is processed and combined to extract significant features for the prediction. Afterwards, a machine learning algorithm is used to make predictions on future machine failures based on the extracted features. The predictions are made following the classification approach described in chapter 4.4.2, where the algorithm is trained to predict whether a component will fail in a given time period. PMB defines workloads for training a new model as well as using an already trained model for new predictions.

5.2 PMB Data Model

The data model of a Big Data benchmark must be comparable to those of real live applications. Otherwise it cannot be considered representative and results are not useful for the process of implementing Big Data solutions [12]. Therefore, two main aspects have to be considered for the data model of PMB. First, the data model must be representative of predictive maintenance use cases. In particular this means the inclusion of various condition monitoring measurements as well as event text based data such as errors and maintenance information [104]. Second, the data model must represent Big Data and its main characteristics volume, variety and velocity.

This section introduces the chosen data model and explains how it meets the above mentioned criteria. After presenting the components of the data model, the model is analysed in respect to the three dimensions of volume, variety, and velocity.

5.2.1 Specifications of the PMB Data Model

The basis for the data model is a data set [181] published by Microsoft[39], which is part of the guide "Predictive Maintenance Modelling Guide" of their Big Data analytics platform Microsoft Azure[40]. The data set was created by monitoring 100 machines over the period of one year. Each hour four condition indicators were measured (volt, pressure, rotation, vibration). Furthermore, errors, failures, and services were logged. Each machine consists of five components which can break down. The data set is anonymised and does not specify which machines where monitored. Additionally, model, error types, and components are only available as generic names (i.e. error1, error2) to prevent conclusions about the machine itself. As illustrated in Figure 35 the data model consists

[39] http://www.microsoft.com/
[40] http://azure.microsoft.com/

of the five parts *telemetry data*, *machine metadata*, *error log*, *maintenance log*, and *machine failures*.

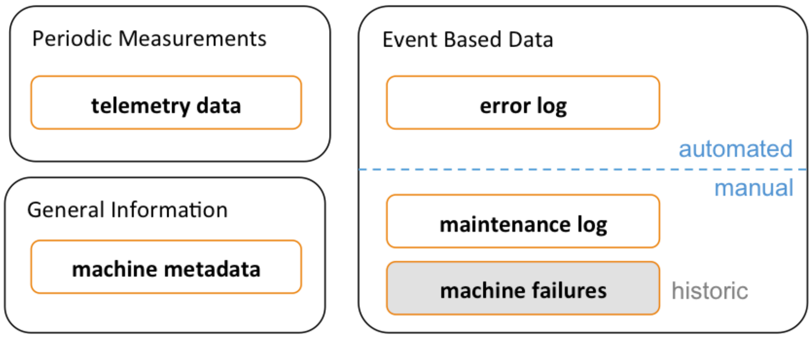


Figure 35: Data model of the PMB.

The periodical telemetry data holds information about the condition of the machines and is recorded each hour. Furthermore, the data model consists of three event based data logs: the error log which automatically registers all minor errors a machine experiences, the maintenance log which keeps records of time and substance of regular services and is maintained by the service personnel, and the failure log which documents machine breakdowns. The last data set of the data model is the machine metadata, which holds additional information about the machine such as type and age.

Telemetry Data

The telemetry data set consists of condition data of 100 machines. Each hour, voltage, pressure, rotation, and vibration of the machine is measured. Each set of measurements is marked with a timestamp and the ID of the machine. Table 4 lists the information recorded in the telemetry data. Additionally, the range of the variables, their arithmetic mean μ, and the standard deviation σ are presented.

Table 4: Telemetry data set of the PMB data model.

Name	Description	Unit	Range	μ	σ
machineID	ID of machine	number	1-100		-
datetime	date and time of record	date	1.1.– 31.12.2015		-
volt	voltage level of the machine	volt	97 – 256	170.8	15.5
pressure	pressure inside the machine	bar	51 - 186	100.9	11.0
rotation	rotation speed of the rotor		138 – 695	446.6	52.7
vibration	vibration indicator		15 - 77	40.4	5.4

Figure 36 displays the distribution of the individual values in respect to the number of their measurements. The highest peak of the graph represents the mean value. The higher the value of the standard deviation, the flatter the distributions curve. Vibration data has the smallest standard deviation and therefore shows the steepest graph while rotation displays the flattest curve due to its high standard deviation.

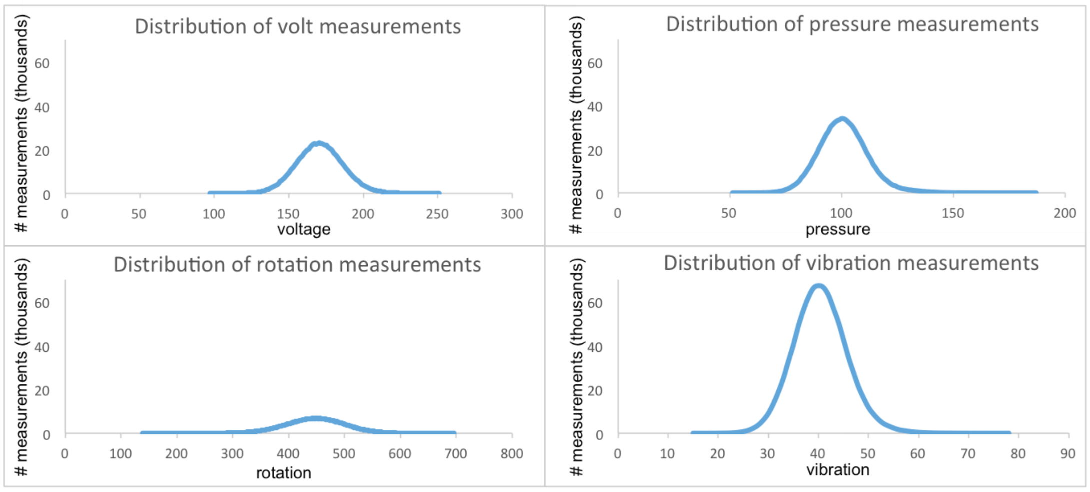


Figure 36: Distribution of telemetry data.

As an example, Figure 37 shows the development of telemetry data over one week. During this time three errors and one failure occurred as indicated by the orange (error) and red (failure) lines. Errors are small disturbances during runtime while failures occur when a component breaks down. The illustrations clearly show the ups and downs of vibration measurements during normal runtime as well as heavy drops just before or after errors and failures.

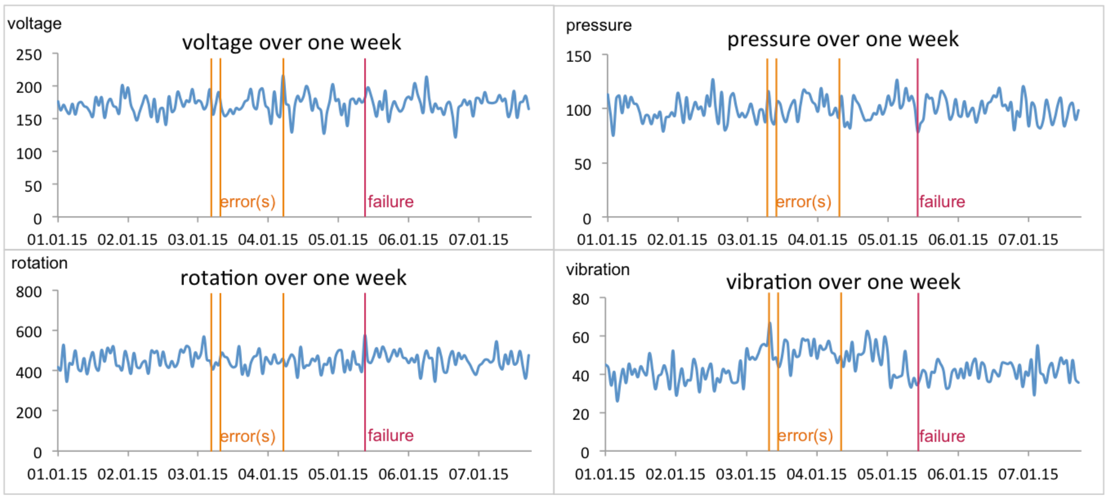


Figure 37: Telemetry data over one week.

Machine Metadata

The machine metadata stores additional information about machines such as model type and age. There are 100 machines of four different types and age between 0 and 20 years. The model types are anonymised and only available as generic names (modelX). Table 5 lists the individual features.

Table 5: Machine metadata of the PMB data model.

Name	Description	Unit	Range
machineId	ID of machine	number	1-100
model	type of machine	name	Model1-Model4
age	age of machine	year	1-20

Figure 38 shows the model and age distribution of the machines. There are four different machine models available. More than 65% of all machines are either Model 3 or Model 4 (Figure 38, left graph). The smallest group is Model 1 with a size of only 16%. Machines are between 0 and 20 years old. The average age is 11.3 years. 35% of all machines are 15 years or older (right graph).

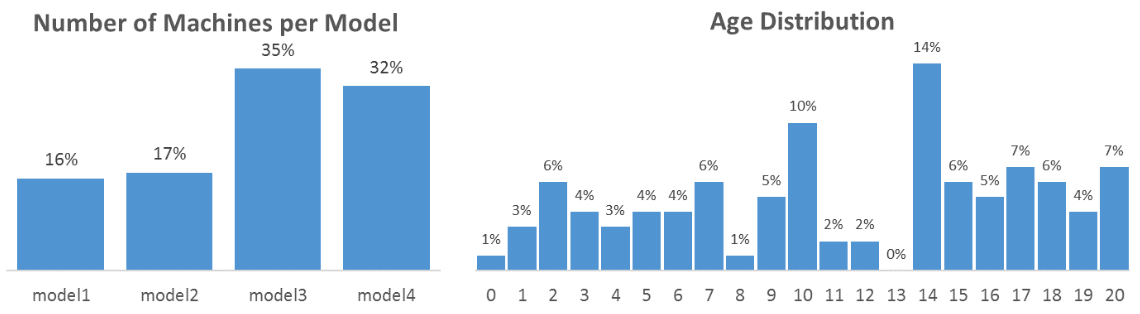


Figure 38: Model and age distribution of machines.

Error Log

Each time a minor error occurs the machine records it automatically. In contrast to fatal errors, minor errors do not result in failure. The machine is able to detect five different errors. The nature of these errors is not further specified in the data set. Each record is accompanied by a timestamp. The time is rounded to the closest hour. Table 6 lists the individual features recorded in the error log.

Table 6: Error log data of the PMB data model.

Name	Description	Unit	Range
machineId	ID of machine	number	1-100
datetime	date and time of record	Date	1.1.– 31.12.2015
errorID	ID of error	name	Error1-Error5

During the monitored year 3.919 errors have been recorded. As shown in the left bar graph of Figure 39, Error 1 and 2 are most common with over 50% of all recorded errors falling into these two categories. The least frequent error is Error 5 with only 9.1%. On average, a machine experiences 39 errors per year. Over half of all machines recorded between 35 and 45 errors (Figure 39, middle graph). No machine recorded less than 20 errors and only one had more than 55. Errors seem to be independent of age (Figure 39, right graph). All four age groups experience approximately the same amount of average errors.

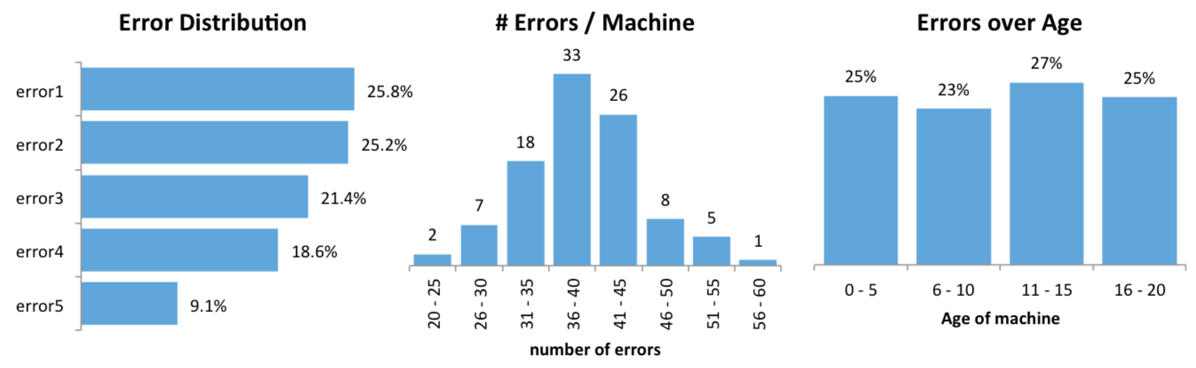


Figure 39: Distribution of errors (left), number of errors per machine (middle), and errors per age of machine (right)

Maintenance Log

The maintenance team services machines periodically. During these services the machine is inspected, and components are replaced if necessary. The service crew records the date and exchanged components in the maintenance log. The components are not further specified in the data set than by generic names (CompX). There are four different types of components. Table 7 displays the individual features, their data type and range.

Table 7: Maintenance log data of the PMB data model.

Name	Description	Unit	Range
machineID	ID of machine	number	1-100
datetime	date and time of record	date	1.1 – 31.12.2015
comp	ID of changed component	name	Comp1-Comp4

In total 3.268 services with component changes have been recorded in 2015. As shown in Figure 40, replacements are distributed evenly among all four components (left graph). On average, a machine experiences 33 component changes per year. Only 9 machines have more than 38 replacements and only 9 less than 28 (Figure 40, middle graph). The number of services is independent of age (Figure 40, right graph). All machine age groups experience approximately the same amount of component replacements.

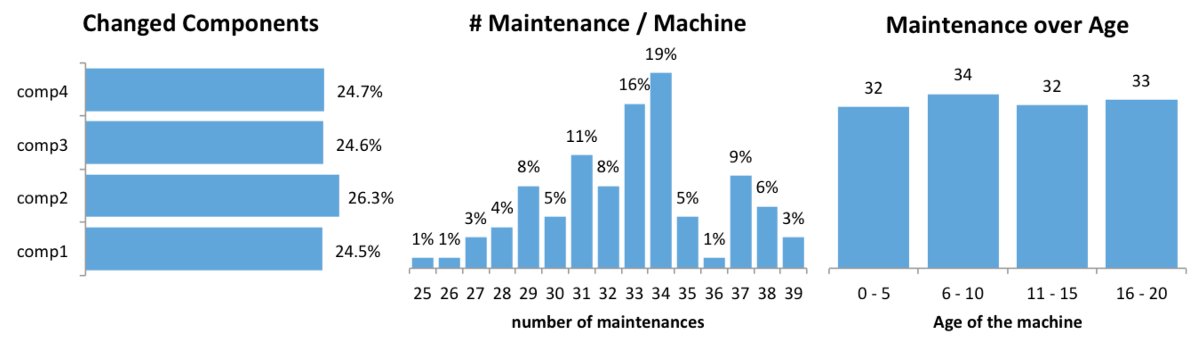


Figure 40: Distribution of exchanged components (left), number of services per machine (middle), and number of services per age of machine (right).

Failure Log

Machine failures occur if a component breaks down. The affected component is replaced by the maintenance team. Each time a component is exchanged due to failure, it is recorded in the failure log. The failure log follows the same structure as the maintenance log and is also kept manually by the service personnel. Each record consists of the time of failure, the failed component, and the id of the machine. As before, the individual components are not further specified. The time is rounded to the closest hour. Table 8 lists the individual features, their data type and range.

Table 8: Failure log data of the PMB data model.

Name	Description	Unit	Range
machineID	ID of machine	number	1-100
datetime	date and time of record	date	1.1 – 31.12.2015
failure	ID of changed component	name	Comp1-Comp4

Overall, there have been 761 failures during the entire year. The components vary in their susceptibility to failure. As shown in Figure 41, Component 1 and 2 experienced the most break downs, while Component 3 is least prone for failure (left graph). On average, machines experience 7.6 component failures per year. Most machines record 6-10 failures (Figure 41, middle graph). Only 2 % of all machines have not had any breakdowns. In general, older machines are more error-prone to failure than new ones (Figure 41, left graph). Over 60% of all failures occur on machines older than 10 years. Since machines are also more vulnerable right after commissioning, the slight increase of failures in young machines is expected.

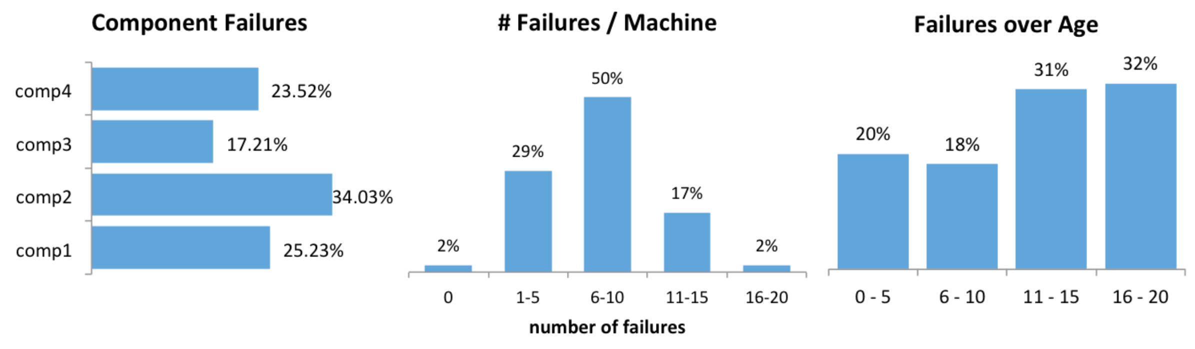


Figure 41: Distribution of component failures (left), number of failures per machine (middle), and number of failures per age of machine (right).

5.2.2 Volume, Velocity, and Variety of the PMB Data Model

This section covers how the PMB data model addresses the Big Data key characteristics volume, velocity, and variety.

Volume

To address the dynamic volume characteristic of Big Data, the data model of a Big Data benchmark must be able to provide different data sizes and possibilities for future data expansion. Below, all possibilities for adjusting the volume are discussed. The volume of the PMB data model is determined by three main factors:

- Number of monitored machines
- Number of records per machine
- Number of measured variables (i.e. voltage)

The number of monitored machines influences all five data sets of the PMB data model. Altering the number of monitored machines therefore impacts the entire data model. The data sets have information of 100 machines, making one machine make up 1% of the data volume. Decreasing the number of machines is as simple as removing all records with a given machine id from all five data sets, enabling data sets between 1 and 100 machines. Increasing the number of monitored machines is achieved by duplicating existing machine data and assigning it new machine ids. By duplicating existing machines, the data size of the PMB data model can be increased indefinitely. Therefore, altering the number of monitored machine is a viable option to influence the data set size of the PMB data model.

The number of records per machine is influenced by the length of observation as well as the interval between measurements. For the data sets of the PMB data model, the length of observation is one year. The observation period directly influences all five data sets and can be decreased by removing all data items collected after a given point in time. Increasing the observation time is done by duplicating data from one year to the next (i.e. January 2015 is duplicated to January 2016). However, increasing the observation period by duplicating data does not consider potential aging effects of the machines, making it suboptimal for data size adjustments. The measurement intervals are fixed at 1 hour. Shortening the intervals results in more measurements and extending the intervals leads to less measurements. Extending the intervals can be achieved by deleting data points within them. Shortening intervals however is not possible without additional measurements. Thus, it is not feasible to adjust the data size by altering the measurement periods.

The number of measured variables of the PMB data model are given by the data set. There is not enough information available for adding meaningful variables to the data set and removing a variable may take away too much information, leading to an unrealistic data model. Thus, changing the number of variables is not a viable option for altering the volume of the PMB data model.
In summary, altering the number of monitored machines is the best method to adjust the volume of the data model. This method will be used to provide different data sizes for the PMB.

Velocity

The data model of Big Data benchmarks must be capable to simulate different speeds of data generation. In the context of the PMB, the velocity is given by the number of measurements entering the system in a time period (i.e. measurements per hour). In the PMB data model, data velocity is influenced by the number of monitored machines and the time interval between measurements. As discussed before, altering the measurement interval is not a viable option since it is not possible to shorten the intervals without additional information. Therefore, the velocity of the PMB data model is adjusted by altering the number of machines.

Variety

The data model combines periodic measurements (telemetry), event based data (maintenance, errors, failures) and long-term information (machine metadata), which are all typical for predictive maintenance [104]. All measurements are value type data and do not include other data types such has waves. There is also no unstructured or semi-structured data in the data set. However, while unstructured or semi-structured data is common in many Big Data applications [38], it is seldom considered in predictive maintenance use cases [104].

The data model includes the most popular data types for predictive maintenance. Nevertheless, it does not cover all possibilities (i.e. waveform analytics). In future work, the PMB can be extended by adding more data types to the data model.

5.3 Workload Specification

This section presents the workload of the benchmark. PMB is designed to represent common tasks within a predictive maintenance system. PMB is based on a predictive maintenance use case, where future machine failures are predicted using multiple data sources. The workload can be separated into the two phases of *training* and *running* the system.

During the training phase, the system learns to predict future machine failures. More precise, a supervised machine learning algorithm learns to predict whether a specific component will fail during the next 24 hours. The machine learning algorithm is trained on data collected over one year, consisting of telemetry data, error logs, maintenance log, failure logs, and meta information. Before training, the different data sources are first combined and processed to extract important features. During training, the algorithm processes the data set to examine correlations between conditional data and failures. After completing the training, the precision of the algorithm is tested. Training an algorithm is generally done on big data sets. Therefore, this part of the PMB tests how Big Data analytic ecosystems perform when processing big data set at once.

The running phase simulates normal operation of a predictive maintenance system. During the running phase, the trained algorithm is utilized to make predictions for new data input. After loading, the model is used to classify new information. Rather than making multiple predictions at once, multiple single predictions are made in a row. In contrast to the training phase, the running phases tests how a Big Data analytic ecosystem handles multiple small requests to predict machine failures.

5.3.1 Phase 1: Training the Predictive Maintenance System

In the training phase, the predictive maintenance system learns how to predict machine failures. As illustrated in Figure 42, this training process can be separated into the three main steps of preprocessing, training, and testing. During preprocessing, collected data is combined and important features are extracted from the data. After preprocessing, the created data set is split into a training and test set. The training set is used to train the machine learning algorithm into a trained model. The test set is then utilized to test the accuracy of this model. The individual steps are portrayed in detail below:

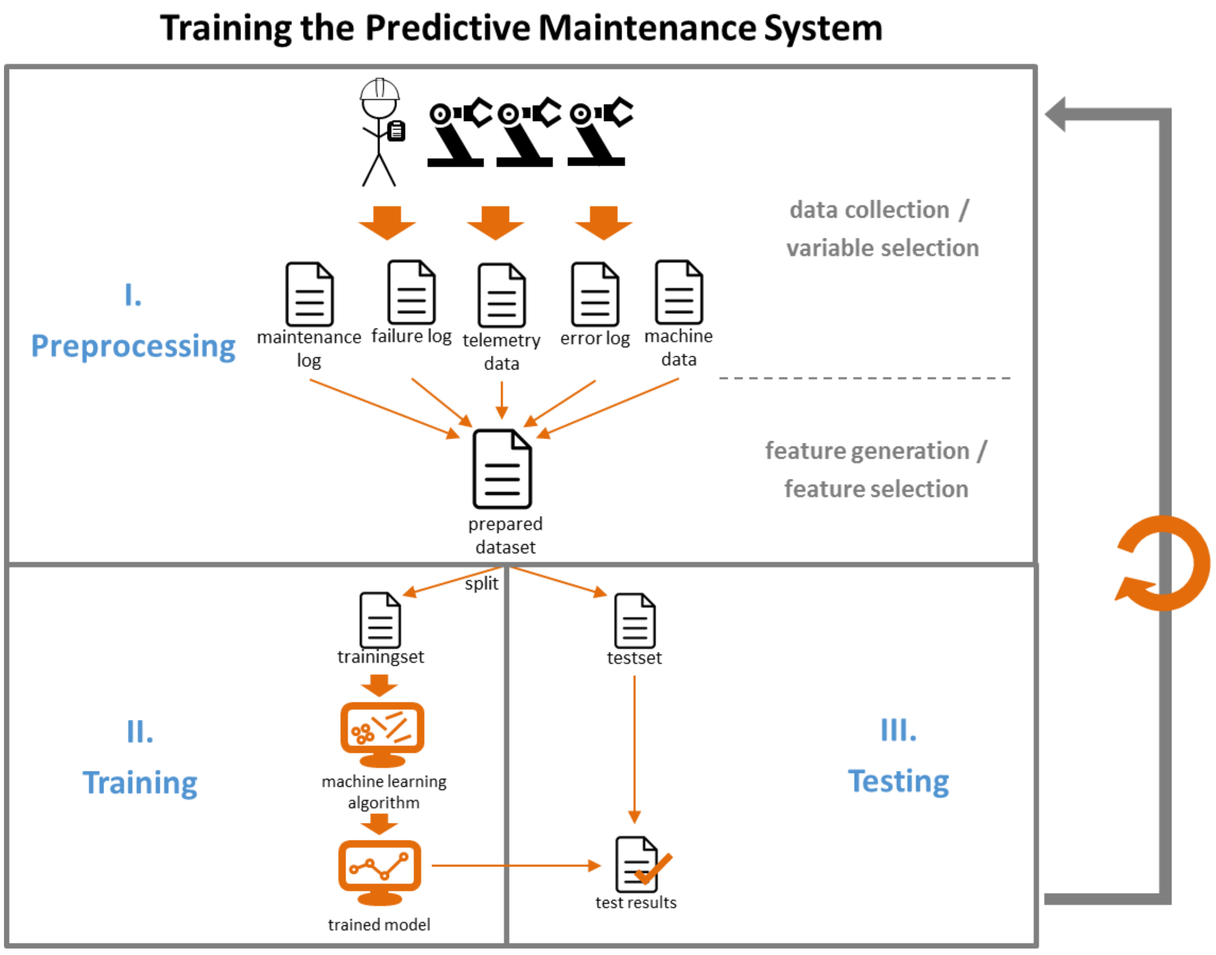


Figure 42: Phase 1 of PMB: Training the predictive maintenance system.

Preprocessing the Data

The preprocessing step covers all tasks to prepare raw incoming data for the training of the machine learning algorithm. Preprocessing can be separated into the three subtasks of ingesting data into the system, combining the individual data sets, and splitting the data into training and test set. For each step, the execution time is measured and used as performance indicator.

<u>Data Ingestion</u>

Ingesting the data describes all necessary steps to bring the data into the Big Data analytic ecosystem. In general, this includes tasks such as creating databases or file systems and copying data into them. At the end of data ingestions, the information should be accessible for further processing. The execution time of the entire process is timed for the benchmark metric.

Data Processing

After data ingestion, the data sets are combined and prepared for the training of the machine learning algorithm, resulting in one single data set. The final data set combines information from the telemetry data set with the event based error and maintenance log. Furthermore, it contains information about age and model of the machine. Each data row is labelled with the information if a component will fail in the next 24 hours. This label represents the value that will be predicted by the machine learning algorithm. The entire process is timed and used for the PMB metric.

Table 9 shows the content of the final data set. The date and machine ID identify each record set uniquely. Information about the machine (model and age) is added from the machine meta data. The telemetry data (volt, pressure, rotation, and vibration) is accompanied by calculated short time trend indicators such as mean and standard deviation for the past three, and 24 hours. Information about the errors is added by calculating how often a specific error occurred in the last 24 hours and the maintenance log is used to calculate the days since a component was last changed. At last, the label is added to the record set. The label represents the data field that should be predicted by the predictive maintenance system. It states whether a component will fail within the next 24 hours.

Table 9: Final data set after preprocessing.

Name	Description	Unit	Source
datetime	date and time of record	Date	telemetry data
machineID	ID of the machine	numerical	telemetry data
General Machine Data			
model	Model of the machine	numerical	meta data
age	Age of the machine	numerical	meta data
Telemetry Data: Current Information and short-time Trend			
volt	voltage level of the machine	numerical	telemetry data
pressure	pressure inside the machine	numerical	telemetry data
rotation	rotation speed of the rotor	numerical	telemetry data
vibration	vibration indicator	numerical	telemetry data
volt-mean 3h	μ voltage of last 3h	numerical	mean of last 3 hours
rotation-mean 3h	μ rotation of last 3h	numerical	mean of last 3 hours
pressure-mean 3h	μ pressure of last 3h	numerical	mean of last 3 hours
vibration-mean 3h	μ vibration of last 3h	numerical	mean of last 3 hours

volt-sd 3h	σ volt of last 3h	numerical	σ of last 3 hours
rotation-sd 3h	σ rotation of last 3h	numerical	σ of last 3 hours
pressure-sd 3h	σ pressure of last 3h	numerical	σ of last 3 hours
vibration-sd 3h	σ vibration of last 3h	numerical	σ of last 3 hours
volt-mean 24h	μ voltage of last 24h	numerical	mean of last 24 hours
rotation-mean 24h	μ rotation of last 24h	numerical	mean of last 24 hours
pressure-mean 24h	μ pressure of last 24h	numerical	mean of last 24 hours
vibration-mean 24h	μ vibration of last 24h	numerical	mean of last 24 hours
volt-sd 24h	σ volt of last 24h	numerical	σ of last 24 hours
rotation-sd 24h	σ rotation of last 24h	numerical	σ of last 24 hours
pressure-sd 24h	σ pressure of last 24h	numerical	σ of last 24 hours
vibration-sd 24h	σ vibration of last 24h	numerical	σ of last 24 hours
Error Information			
error1count	number of occurrences of error1 in the last 24 hours	numerical	error log
error2count	number of occurrences of error2 in the last 24 hours	numerical	error log
error3count	number of occurrences of error3 in the last 24 hours	numerical	error log
error4count	number of occurrences of error4 in the last 24 hours	numerical	error log
error5count	number of occurrences of error5 in the last X days	numerical	error log
Maintenance Information			
comp1_lastchange	days since last replacement of component 1	numerical	maintenance log
comp2_lastchange	days since last replacement of component 1	numerical	maintenance log
comp3_lastchange	days since last replacement of component 1	numerical	maintenance log
comp4_lastchange	days since last replacement of component 1	numerical	maintenance log
Failure Information			
failure	ID of component that fails within the next 24h. 0 if none fail.	numerical	Failure log

Splitting the Data Set

The final step of preprocessing is splitting the data set into training and test set. The training set is used for training the machine learning algorithm. Afterwards the trained algorithm predicts failures based on the data from the test set. These predictions are then compared to the real failure data to calculate the precision of the algorithm. For the PMB, the data set is split in a ratio of 70:30 for training and test set.

Training the Machine Learning Algorithm

In this step the machine learning algorithm is trained on the training set produced during preprocessing. The resulting model should be able to make predictions if any of the five machine components will fail during the next 24 hours. Therefore, a supervised machine learning algorithm for classification is necessary. As mentioned in chapter 4.4.2, the most commonly used classification algorithms for predictive maintenance are decision trees, support vector machines, and artificial neural networks. While all three are viable options, the decision tree is currently available in most machine learning libraries for Big Data analytics. The decision tree also produces the most transparent model, making it easy to understand the reasoning behind predictions. Therefore, the PMB uses a decision tree as machine learning algorithm. Other machine learning algorithms may be added in future work.

After initializing the data and the machine learning algorithm, the decision tree is trained. The training of the algorithm should be done in a distributed fashion, utilizing multiple nodes of a cluster.

Testing the Machine Learning Model

The trained decision tree is tested using the test set produced in preprocessing. Testing a machine learning model is done in two steps. First, the model makes predictions based on the data from the test set without knowing the real labels. Second, the predictions made by the model are compared to the real labels, and the accuracy is calculated.

5.3.2 Phase 2: Running the Predictive Maintenance System

The second phase of PMB tests the predictive maintenance system in a running environment. It can be divided into the two steps of preprocessing and scoring as illustrated in Figure 43. Similar to Phase 1, preprocessing covers all tasks from data ingestion to preparing the data set. During scoring the trained model is loaded and used to make predictions.

Phase 2 is tested using different data and request sizes to examine their effects. The data sizes are determined by the number of telemetry data items that have to be predicted by the system. There are three different data sizes consisting of 10, 100, and 1.000 data items respectively. By using different data sizes, it can be analysed how the predictive maintenance system scales with the number of predictions it must make at once. The request sizes. Running one data size through the entire process of Phase 2 represents one request. Besides testing the effect of data size, the PMB also tests the effect of multiple subsequent requests. Therefore, three request sizes are used within Phase 2 of the PMB. The small size consists of 1 request, medium size of 10 requests, and big size of 100 requests. By varying data and request size, the PMB tests how the system handles multiple predictions at once as well as multiple predictions after each other.

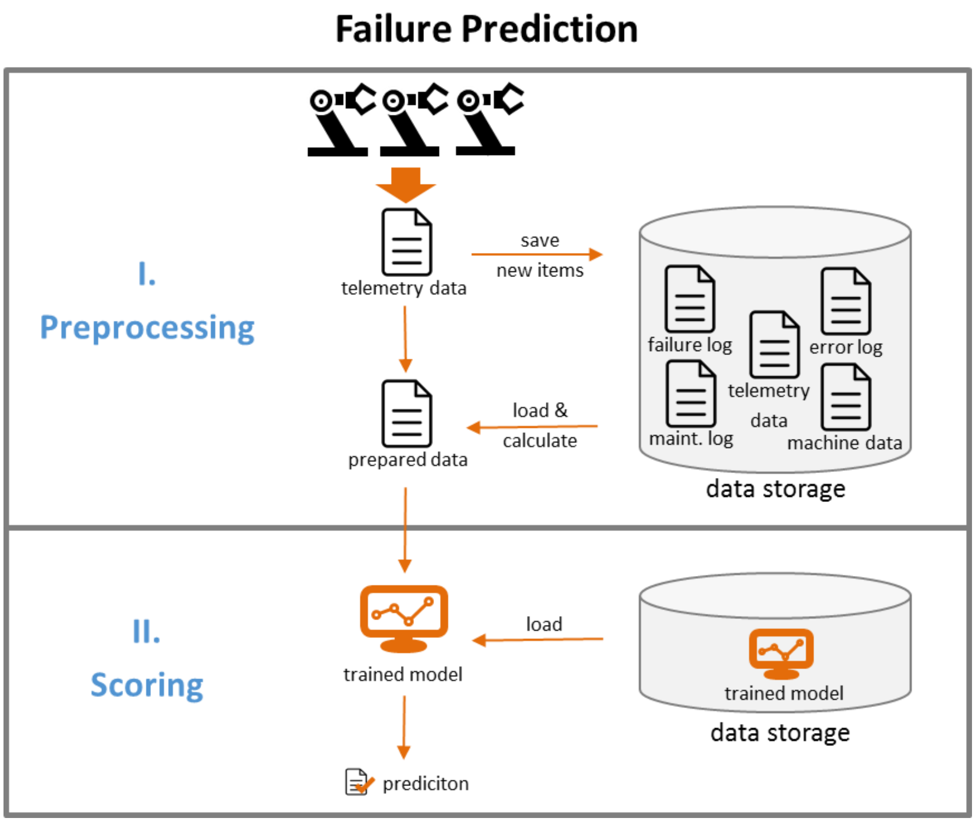


Figure 43: Phase 2 of PMB: Running the predictive maintenance system.

Preprocessing the Data

In a production environment, preprocessing covers all activities from ingesting new data into the system to preparing it for the trained machine learning model. The prepared data set has to be in the exact same format as the data set the algorithm was trained on. Otherwise a prediction is not possible.

When new telemetry data enters the system, all values as described above in Table 9 have to be calculated. The necessary information is loaded from data storage. For the PMB, the two tasks of ingesting new data as well as preparing the data set are timed.

Scoring the Data

During data scoring, the trained model is used to predict failures based on new information. The trained machine learning algorithm is loaded and used to classify the newly prepared data.

5.4 Metrics

The PMB tests the performance of Big Data analytic ecosystems in the field of predictive maintenance. The performance is measured by the execution time of the individual tasks. As summarized in Table 10, the workload is separated into the two phases of training and running the predictive maintenance system.

Table 10: Workloads of the PMB.

Workload	Description
Phase 1	Training the predictive maintenance system
Preprocessing	All steps from data ingestion to preparing it for training
Data Ingestion	Create database and copy data sets into the system
Processing	Combine data and calculate necessary features
Training	Training of the decision tree
Loading	Load data and initialize model
Training algorithm	Train the decision forest on training data set
Testing	Test the algorithm
Loading	Loading data and model
Predicting	Predict failures based on the test set
Phase 2	Running the predictive maintenance model
Preprocessing	All steps from data ingestion to preparing it for training

Data Ingestion	Ingest new telemetry data into the system
Processing	Calculate all necessary features
Scoring	Score new telemetry data
Loading	Load the prepared data and machine learning model
Predicting	Use trained decision tree to make predictions

Each phase is executed multiple times, using different data sizes. This way, PMB also analyses how the performance of the Big Data analytic ecosystem scales with different growing data.

Table 11 lists the various data set sizes for both phases. Phase 1 is executed using three different data set sizes: A small data set, containing data of 33 machines; a medium data set, containing data for 66 machines; and a big data set, containing data of 100 machines. Phase 2 is executed using different request sizes and data sizes. One request covers the entire workload described in Phase 2. Multiple requests therefore execute Phase 2 multiple times after each other. Phase 2 is tested with a small data size of 10 items, a medium data size of 100 items, and a big data size of 1.000 items. Additionally, three request sizes of 1, 10, and 100 requests are used.

Table 11: Data set sizes for both phases of the PMB.

Data Size	Specification
Data set sizes for Phase 1:	
small	Data set includes data of 33 machines
medium	Data set includes data of 66 machines
large	Data set includes data of 100 machines
Data set / request size for Phase 2:	
small / small	1 prediction request with 10 data items
small / medium	1 prediction request with 100 data items
small / large	1 prediction request with 1.000 data items
medium / small	10 prediction requests with 10 data items each
medium / medium	10 prediction requests with 100 data items each
medium / large	10 prediction requests with 1.000 data items each
large / small	100 prediction requests with 10 data items each
large / medium	100 prediction requests with 100 data items each
large / large	10 prediction requests with 1.000 data items each

6 Benchmark Evaluation

The PMB is evaluated by a case study. The benchmark is implemented and executed on two popular Big Data analytic ecosystems, which run on a 5 node Raspberry Pi cluster. The results are then analysed and compared to the findings of similar benchmarks.

This chapter starts with a scientific literature research, identifying popular open source Big Data analytic frameworks. Based on this research, the frameworks for implementing the PMB are chosen. The chosen frameworks are then portrayed in detail. Afterwards, the testing environment, the individual benchmark implementations, and their results are presented. Finally, the PMB is evaluated by comparing the results of the implementations and analysing findings gathered during the process.

6.1 Selection of Big Data Analytic Frameworks

To determine the most popular open source Big Data analytic frameworks, a ranking was established using two main criteria:

I. How many times is a framework mentioned in survey papers?

II. How many times is a framework cited in peer reviewed papers?

The above listed criteria must be first formulated as research questions. Based on these research questions, search queries are formulated to find related papers using the *Catalog Plus*[41] search engine provided by the TU Vienna. The resulting documents are screened to filter out relevant papers, which are finally used to create the popularity ranking of open source Big Data analytic frameworks. The problem resulting from the first criteria (I) can be formulated as two separate research questions:

R1: What scientific survey papers of Big Data analytic frameworks are available?

R2: How many times are the individual Big Data analytic frameworks mentioned within the papers resulting from R1?

The second criteria (II) analyses how often Big Data analytic frameworks are cited in peer reviewed papers. To limit the search space, only the most popular frameworks resulting from R2 are considered.

[41] http://catalogplus.tuwien.ac.at

R3: How many times are the popular Big Data analytic frameworks (based on R2) mentioned in the title of peer-reviewed papers?

R4: How many times are the popular Big Data analytic frameworks (based on R2) mentioned inside the entire text of peer-reviewed papers?

R1: What scientific survey papers of Big Data analytic frameworks are available?

The following query searches for scientific survey papers of Big Data analytic frameworks:

R1/Q1: "Survey" in the title AND "Big Data analytic framework" in the text

The query resulted in 137 potentially relevant documents. The filtering is done in three steps: First, papers are filtered based on their title. Second, papers are filtered based on their abstract. And third, papers are filtered based on their entire content. This screening process resulted in 9 relevant survey papers as listed in Table 12. Khalifa et al. [18] also provide a broad survey while describing their six pillar model (see chapter 3.3). Therefore, their paper was added. Table 12 lists title, reference, journal, and publication year of relevant papers. All journals where published between 2013-2016.

Table 12: Scientific survey papers of Big Data analytic frameworks.

Title and Reference	Journal	Year
A Survey on Real-time Big Data Analytics: Applications and Tools [182]	IEEE	2016
Parallel and Distributed Collaborative Filtering: A Survey [183]	ACM Computing Surveys	2016
The Six Pillars for building Big Data Analytics Ecosystems [18]	ACM Computing Surveys	2016
A survey of open source tools for machine learning with big data in the Hadoop ecosystem [94]	Journal of Big Data	2015
Big Data Analytics: A Survey [184]	Journal of Big Data	2015
In-Memory Big Data Management and Processing: A Survey [49]	IEEE	2015
A Survey on Platforms for Big Data Analytics [185]	Journal of Big Data	2014
Big Data: A Survey [186]	Springer Science	2014
Toward Scalable Systems for Big Data Analytics: A Technology Survey [187]	IEEE	2014
A Survey on Big Data Analytic Tools [188]	IDEAS 2013	2013

R2: How many times are the individual Big Data analytic frameworks mentioned within the papers resulting from R1?

The second research question analyses how many times open source Big Data analytic frameworks are mentioned within the survey papers discovered during R1. Therefore, there is no specific query for the search engine but a screening of the survey papers. Table 13 lists the individual Big Data analytic frameworks and the references of the papers they are mentioned in. The frameworks are further grouped by the 6 pillars and their subcategories [18] (see 3.3). Hadoop and Spark are often mentioned in broader sense and therefore are categorized as ecosystems rather than processing engines.

Table 13: Number of citations in Big Data analytic frameworks survey papers.

Frameworks	Pillar	Subtype	Mentioned in	Sum	Rank
Hadoop	Ecosystem		[18][94][182][183][185][186][188]	7	1.
Spark	Ecosystem		[18][49][94][182][183][185]	6	2.
Mahout	Interface	Scripts	[18][94][183][185][186][187]	6	2.
MLBase	Interface	Scripts	[18][185]	2	6.
MLlib	Interface	Scripts	[18][94]	2	6.
Radoop	Interface	Scripts	[18][186]	2	6.
SAMOA	Interface	Scripts	[94]	1	7.
deeplearning4j	Interface	Scripts	[18]	1	7.
FlinkML	Interface	Scripts	[94]	1	7.
HiveMall	Interface	Scripts	[18]	1	7.
QDrill	Interface	Scripts	[18]	1	7.
Hive	Interface	SQL	[18][94][185][187][188]	5	3.
Drill	Interface	SQL	[18][94][188]	3	5.
YARN	Orchestration	Scheduling	[18][94][185]	3	5.
IReS	Orchestration	Scheduling	[18]	1	7.
Pegasus	Orchestration	Scheduling	[18]	1	7.
MapReduce	Processing	Batch	[94][185][186][187]	4	4.
Flink	Processing	Batch/Incr.	[94][183]	2	6.
DistWEKA	Processing	Batch	[18][94]	2	6.
H2O	Processing	Batch	[18][94]	2	6.
Storm	Processing	Incremental	[18][94][182][183][186][187]	6	2.
Spark Streaming	Processing	Incremental	[18][49][185]	3	5.
Samza	Processing	Incremental	[18]	1	7.
Tez	Processing	Interactive	[18]	1	7.
Cassandra	Storage	Column	[18][94][186][187]	4	4.
HBase	Storage	Column	[18][94][186][187]	4	4.
HyperTable	Storage	Column	[186]	1	7.
HDFS	Storage	DFS	[18][185][94][186]	4	4.
Alluxio	Storage	DFS	[185]	1	7.
FastDFS	Storage	DFS	[186]	1	7.
QFS	Storage	DFS	[186]	1	7.
HyPer/ScyPer	Storage	RDBMS	[49]	1	7.
MySQL Cluster	Storage	RDBMS	[18]	1	7.
ScaleDB	Storage	RDBMS	[18]	1	7.

Overall, Hadoop, Spark, Mahout, Storm, and Hive are clearly the most mentioned technologies, being cited seven, and six times respectively. MapReduce, Cassandra, HBase, and HDFS follow with 4 citations each. YARN, Drill, and Spark Streaming are the only other frameworks mentioned more than twice.

R3: How many times are the popular Big Data analytic Frameworks (based on R2) mentioned in the title of peer-reviewed papers?

R4: How many times are the popular Big Data analytic Frameworks (based on R2) mentioned inside the entire text of peer-reviewed papers?

Both, the third and fourth research question analyse how many times the most popular frameworks as determined in R2 are mentioned in peer-reviewed journals. For each framework, three different search queries are introduced as listed below. The first query searches for papers containing the name of the framework in the title. The second query builds upon this and adds the term "Big Data" as a search parameter. In the third query, the name of the framework is searched for within all search fields. To ensure only relevant papers are included, the term "Big Data" is given as a second search parameter. The first and second queries are expected to give similar results. Table 14 shows the results of the executed queries.

R3/Q1: "*Name of the Framework*" in the title

R3/Q2: "*Name of the Framework*" in the title AND "Big Data" over all fields

R4/Q1: "*Name of the Framework*" over all fields "Big Data" over all fields

For each pillar, regardless of subtype, the top 3 frameworks as determined by R2 are chosen. Only the two subtypes scripts and sql where separated, since they serve fundamentally different purposes. For the category of orchestration, no further analysis is undertaken, since YARN is the only framework mentioned more than once.

Due to the ambiguous names of many frameworks, the name of the developing company was added as search parameter over all fields (i.e. Apache for Storm). The ranking shows the overall rank of the framework for R3 and R4. The score is the mean rank within each query resulting in the overall rank.

Table 14: Number of peer-reviewed papers by Big Data analytic framework.

Frameworks	R3/Q1	R3/Q2	R4/Q1	Score/Rank
		Processing		
Hadoop	171	85	338	1.33 / 1.
Spark	34	28	168	2.66 / 2.
Storm	5	5	44	6.33 / 7.
		Interface / Scripts		
Mahout	3	1	108	6.66 / 8.
MLlib	0	0	40	9.33 / 10.
MLBase	0	0	10	9.66 / 11.
Radoop	0	0	4	10.00 / 12.
		Interface / SQL		
Hive	13	7	163	3.66 / 4.
Drill	1	1	81	7.66 / 9.
		Storage		
HDFS	13	7	451	2.33 / 2.
HBase	5	1	257	5.00 / 5.
Cassandra	3	3	126	6.00 / 6.

The conducted literature survey clearly shows that – in accordance with the former results – Hadoop and Spark dominate this research area. Especially Hadoop, being the oldest and most developed framework, has been studied extensively. In the category of Scripts, Mahout is mentioned in most papers, with three of them carrying the framework in the title. MLlib, in second place, has no dedicated research papers but is mentioned in 40 others. Hive leads the area of SQL interfaces with 13 papers mentioning the framework within their title and HDFS is in front within the storage category.

Summary and Framework Selection

Apache Hadoop leads both rankings. It is by far the most mentioned and analysed open source Big Data framework and has accumulated a vast number of other related projects around itself. Nonetheless Apache Spark seems to challenge its positions and has gained significant traction by introducing its faster in-memory based execution engine. In third place is the data stream processing engine Storm. Concerning machine learning libraries Mahout is the most prominent framework with Sparks machine learning library MLlib being in second place. In the area of SQL interfaces, Hive leads clearly before Drill in both rankings. And finally, HDFS is ahead in the realm of storage frameworks due to its better performance in the second ranking.

For the implementation of the PMB, a framework from each of the above mentioned categories is necessary: processing, interface/script, interface/SQL, storage, and orchestration. Together, these frameworks form the Big Data analytic ecosystem that is tested by the PMB. For evaluation, the PMB is implemented in two ecosystems. The frameworks are selected depending on their ranking.

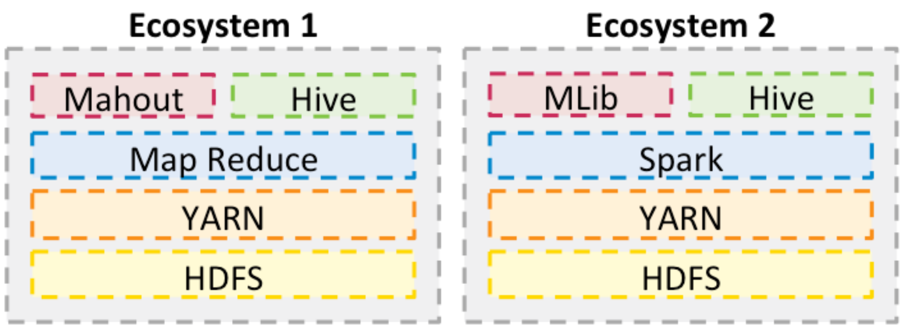


Figure 44: Selected ecosystems for PMB implementation.

Figure 44 illustrates the two chosen ecosystems. Due to the high individual rankings, the two frameworks Hadoop MapReduce and Spark are selected as processing engines (blue). Mahout and MLlib are chosen as machine learning libraries (red). Mahout currently provides the decision tree only for Hadoop, and MLlib only supports the spark processing engine. Each ecosystem is orchestrated (orange) by YARN. HDFS and Hive serve as storage system (yellow) and sql interface (green) for both ecosystems.

6.2 Selected Big Data Analytic Frameworks

6.2.1 Apache Hadoop

Apache Hadoop[42] is an open-source framework for distributed storage and processing of Big Data. It started as an open source implementation of the Google Distributed File System (GFS) [59] and its associated programming model MapReduce [23]. Hadoops first version 0.1.0 was released in April 2006[43]. In 2017, Hadoop released its third big release step with version 3.0.0-alpha. The framework is written in Java and therefore runs on any platform capable of running a java virtual machine.

Today, Hadoop has become an integral part of big companies such as Facebook, Ebay or Adobe [44] . Many companies provide products that build upon Apache Hadoop. Hortonworks[45] and Cloudera[46] are two of the biggest companies offering customized releases of Hadoop with additional features as well as support. Many platform providers like Amazon Web Services (AWS)[47] or Microsoft Azure[48] also offer preconfigured Hadoop environments to their customers.

The Hadoop project includes three main modules: A storage module (*Hadoop Distribued File System (HDFS)),* a cluster management module (*Hadoop Yet Another Resource Negotiator (YARN)*), and a processing module (*Hadoop MapReduce*). Additionally, Hadoop offers a Java API for programming MapReduce applications and provides some support functions such as a web based interface for application tracking. Figure 45 displays how Hadoop can be embedded within the architecture of a Big Data ecosystem.

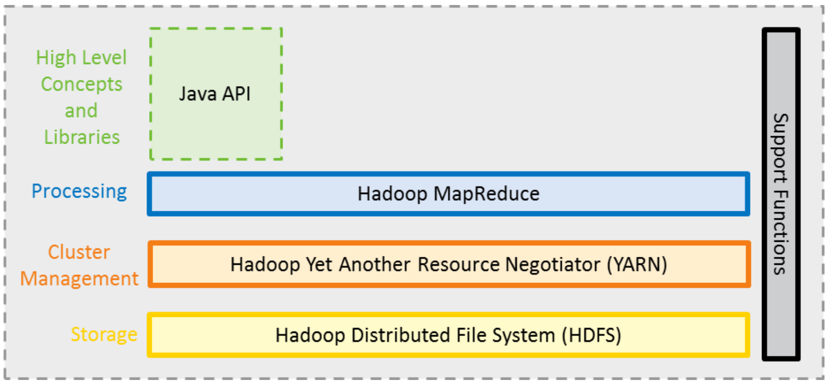

High Level Concepts and Libraries
Java API
Processing
Hadoop MapReduce
Cluster Management
Hadoop Yet Another Resource Negotiator (YARN)
Storage
Hadoop Distributed File System (HDFS)
Support Functions

Figure 45: Embedding Hadoop in the architecture of a Big Data analytic ecosystem.

[42] https://hadoop.apache.org/
[43] https://archive.apache.org/dist/hadoop/core/
[44] https://wiki.apache.org/hadoop/PoweredBy
[45] https://hortonworks.com/
[46] https://www.cloudera.com/
[47] https://aws.amazon.com/
[48] https://azure.microsoft.com/

Hadoop Distributed File System (HDFS)

The Hadoop distributed file system (HDFS)[49] is one of the four core modules of the Apache Hadoop framework and implements a distributed file system designed to store large amounts of data [60]. It is an open source implementation of the Google File System (GFS). HDFS is designed to run on a cluster of commodity hardware, containing hundreds or even thousands of nodes. Due to the high component failure rate in big clusters, data reliability represents a key issue [60]. To address this issue, data is replicated among multiple nodes.

As shown in Figure 46, HDFS implements a master/slave architecture [60]. The master is responsible for managing the file system. It decides where new data should be saved, keeps track where data is stored, and regulates file access to clients. When clients want to store or retrieve information, they contact the master, which in return refers them to a slave node. The slave nodes store data and respond to approved read and write requests from the client. In the Hadoop environment the master is called the NameNode and the slaves are called DataNodes. The system is designed to have only one NameNode but up to hundreds or thousands of DataNodes.

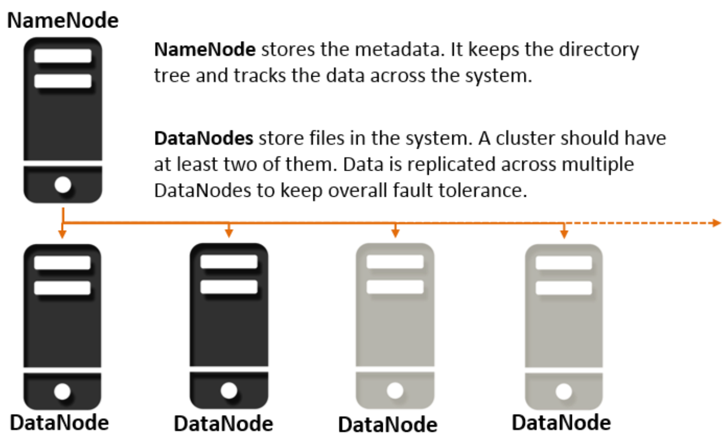

NameNode
NameNode stores the metadata. It keeps the directory tree and tracks the data across the system.
DataNodes store files in the system. A cluster should have at least two of them. Data is replicated across multiple DataNodes to keep overall fault tolerance.
DataNode
DataNode
DataNode
DataNode

Figure 46: HDFS Architecture.

The NameNode in HDFS stores the namespace of the system. The namespace in HDFS is a hierarchy of files and directories. For each file and directory attributes like permission, access times, and disc space quotas are recorded. The NameNode also maintains the locations of files. For faster access the HDFS stores the whole namespace in RAM.

DataNodes are responsible for storing the data. The HDFS interface is patterned after the UNIX file system. Behind this interface, data is stored as multiple smaller data blocks of

[49] http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html

the same size. Each block is represented by two files on the local filesystem of the DataNodes. The first file holds metadata like checksums for the block data and generation stamps, and the second file contains the data itself. Blocks are replicated on multiple DataNodes to ensure data safety. DataNodes send heartbeats every three seconds to the NameNode to confirm their operational status. After 10 minutes of not receiving a heartbeat, the DataNode is considered dead and the NameNode schedules to recreate the lost blocks on other, functional DataNodes. The heartbeat contains data about total storage capacity, storage used and currently conducted data transfers. The NameNode uses this information for load balance across the system.

Figure 47 shows the process of reading and writing data from and to HDFS. If a client wants to store information in HDFS it first creates an HDFS file and fills it with the desired data. The HDFS file is then split into multiple data blocks of the same size. Next the client contacts the NameNode and requests a DataNode to host the first data block and additional DataNodes to store its replicas. Then the client transfers the data block directly to the suggested DataNodes. After the completion of the transfer the client contacts the NameNode again to decide where the next data block and its replicas will be stored. This process repeats for all data blocks of the file. For reading data from HDFS the client first contacts the NameNode for information on what blocks the desired file is made of and their location. The locations of each block are ordered according to their distance to the client. The client tries to fetch the closest replicas of the blocks first and assembles the file after receiving all blocks.

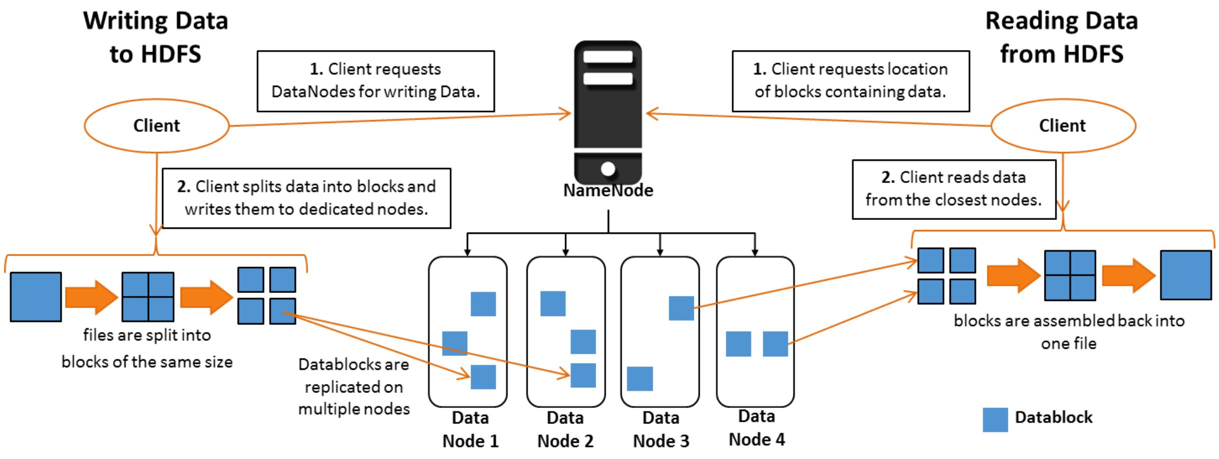

Writing Data to HDFS
1. Client requests DataNodes for writing Data.
Client
2. Client splits data into blocks and writes them to dedicated nodes.
files are split into blocks of the same size
Datablocks are replicated on multiple nodes
NameNode
Data Node 1
Data Node 2
Data Node 3
Data Node 4
Reading Data from HDFS
1. Client requests location of blocks containing data.
Client
2. Client reads data from the closest nodes.
blocks are assembled back into one file
Datablock

Figure 47: Reading and writing data from/to HDFS.

HDFS implements a single-writer, multiple-reader model. While one client writes a file, no other client is permitted to write on that same file. However, multiple users may read the same file simultaneously. The file system is optimised for high throughput instead of low latency. This means, one single request of many read/write commands outperforms multiple single requests containing only a few read/write commands. Therefore, HDFS is streamlined for batch processes rather than interactive user requests.

Yet Another Resource Negotiator (YARN)

Yet Another Resource Negotiator (YARN)[50] is the cluster management framework of Hadoop. Its main responsibility is to manage the computing resources of a cluster [73]. This includes keeping track of available resources and assigning them to applications.

Introduced in 2012 with the second generation of Hadoop (Version 2.0.0-alpha[51]), the basic idea of YARN is to separate the functionalities of resource management and job scheduling/monitoring [73]. In Hadoop 1.0 both tasks where tied together in the MapReduce module. By detaching the resource management, the system is no longer dependent on the MapReduce programming model but may also run other applications.

Like HDFS, YARN follows a master/slave architecture, which is depicted in Figure 48 [73]. The master, or Resource Manager, is the central manager of the cluster resources. It keeps track of available resources and assigns them to applications. Slaves, or Node Managers, are responsible for processing data. They also monitor the health of the node and report their status to the Resource Manager.

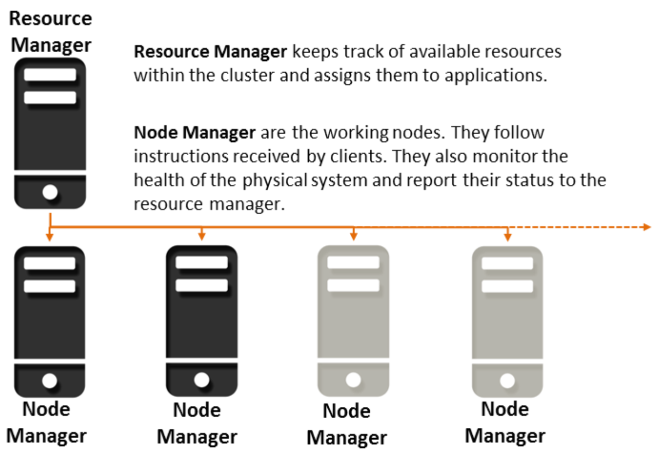

Resource Manager
Resource Manager keeps track of available resources within the cluster and assigns them to applications.
Node Manager are the working nodes. They follow instructions received by clients. They also monitor the health of the physical system and report their status to the resource manager.
Node Manager
Node Manager
Node Manager
Node Manager

Figure 48: YARN Architecture.

The Resource Manager is the central point for managing the cluster resources. In YARN, resources are represented as containers, where a container is a logical set of resources (e.g. 1 GB Ram, 1 CPU). Each container is bound to a node. Applications request resources

[50] http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html
[51] http://hadoop.apache.org/releases.html

from the Resource Manager. Their request submission includes the number and size of required containers as well as locality preferences. The Resource Manager then allocates resources to the application through a scheduler. It assigns them containers and issues tokens that enables the application to contact them. If not all requests can be satisfied, a scheduler decides how to allocate the resources. YARN offers three different schedulers: FIFO, Fair, and Capacity scheduler. The FIFO (first in first out) scheduler allocates resources according to their submission order. Jobs submitted first, are executed first. The Fair scheduler allocates resources in such a way, that over time all applications get an equal share of them on average. The capacity scheduler is designed for sharing Hadoop clusters among multiple organisations. Each organization is guaranteed a certain overall capacity of the cluster. However, if the cluster is not utilized fully, organizations may access additional resources beyond their capacity. This way the cluster is utilized in its full potential. The Resource Manager is not responsible for coordinating execution and for providing fault tolerance for applications. Both these tasks fall into the responsibility of the application itself.

Node Manager represent the working force of the cluster. They authenticate requests and monitor container execution. A Node Manager communicates with the Resource Manager via heart-beats, where they report information about overall and available resources as well launches and terminations of containers. Node Manager also monitor the health of the underlying physical system. If a software or hardware problem on the local system is detected, the status of the node is changed to *unhealthy* and reported to the Resource Manager.

Figure 49 shows YARNs process of resource allocation. At first, a client contacts the Resource Manager and submits an application. After passing a security credential validation as well as administrative checks the Application Master is launched on a container in the system. The Application Master is the manager of an application. It is responsible for the lifecycle management of an application as well as handling faults. It is part of the application itself and may be written in any programming language. The Application Master requests resources from the Resource Manager. Depending on availability and active type of scheduler, the Resource Manager assigns resources in form of container to the applications and issues tokens to the Application Master for activating them. The Application Master then coordinates its tasks and utilizes the containers as necessary. It may dynamically request additional resources or terminate containers if no longer needed. After completion the Application Master terminates their containers and reports to the Resource Manager

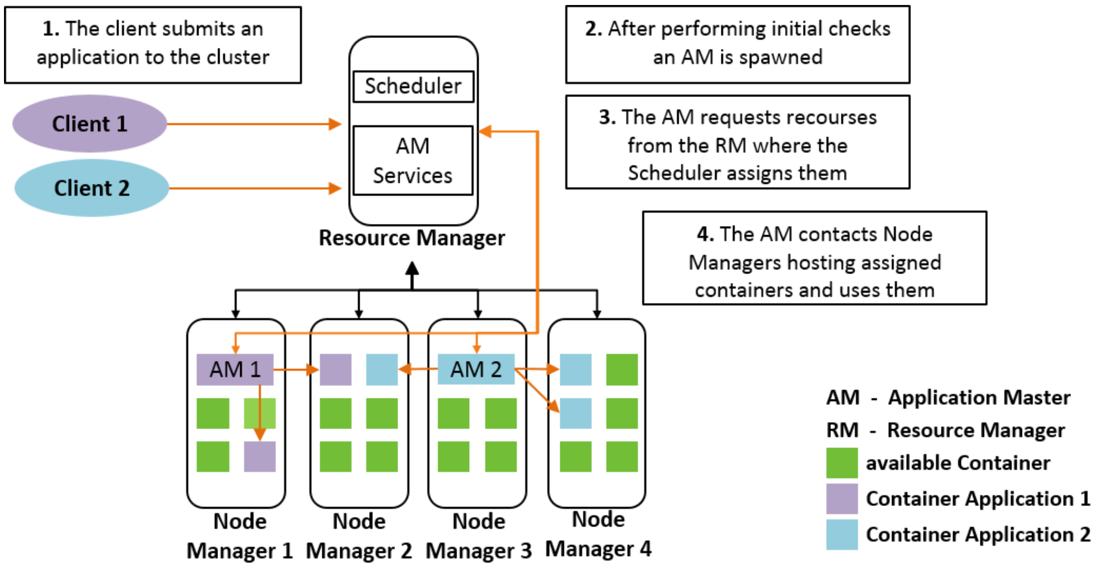

1. The client submits an application to the cluster
Client 1
Client 2
Scheduler
AM Services
Resource Manager
2. After performing initial checks an AM is spawned
3. The AM requests recourses from the RM where the Scheduler assigns them
4. The AM contacts Node Managers hosting assigned containers and uses them
AM 1
AM 2
Node Manager 1
Node Manager 2
Node Manager 3
Node Manager 4
AM - Application Master
RM - Resource Manager
available Container
Container Application 1
Container Application 2

Figure 49: The process of resource allocation in YARN.

Hadoop MapReduce

Hadoop MapReduce[52] is Hadoops native programming framework. It is an open source implementation of Googles MapReduce [23] programming model. It was inspired by the *map* and *reduce* primitives already present in Lisp and other functional programming languages [23]. The purpose of MapReduce lies in processing large amounts of data on computer clusters. It takes a set of input key/value pairs and produces a set of output key/value pairs. Following the principle of divide and conquer, tasks are split into smaller ones and processed in parallel on multiple machines. Generated intermediate results are then merged back into a final output. The process of a MapReduce program can be separated into two main phases: the *map*-phase and the *reduce*-phase [189].

Hadoops implementation of MapReduce consists of two main modules [190]. The first one is the job tracker. Its main purpose is the management of MapReduce jobs. The job tracker receives all jobs from the client, schedules map and reduce tasks, monitors failing tasks and reschedules them if necessary. There is only one job tracker in a Hadoop cluster. The second module is the task tracker. It purpose lies in executing and reporting back to the job tracker. In a Hadoop cluster there exists one task tracker per cluster node.

[52] http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

Figure 50 depicts the steps of a MapReduce program, commonly referred to as a *job*. In the beginning, the input data is split into multiple smaller key/value pairs. In the map phase, the worker nodes take the input data and follow a user defined *map function* to produce a set of intermediate results. In between the map and reduce phase, all intermediate results with the same key value are merged together. This intermediate step is often referred to as shuffle phase. The shuffle phase not considered one of the main phases since it cannot be programmatically influenced by the user. Finally, in the reduce phase, workers follow the user defined *reduce function* to group intermediate results together.

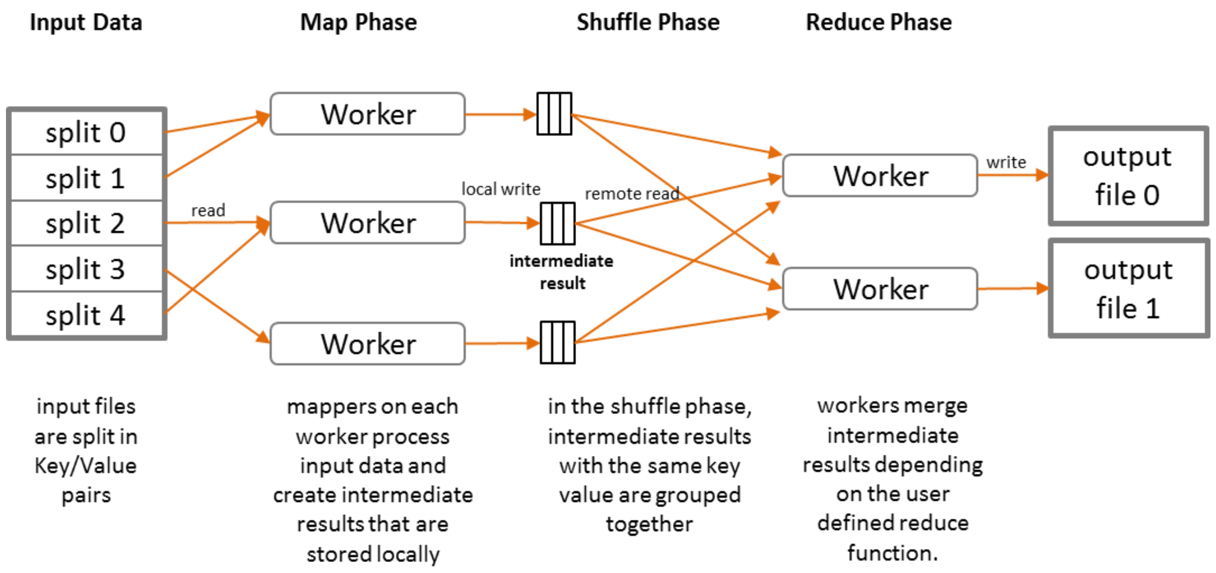

Input Data
Map Phase
Shuffle Phase
Reduce Phase
split 0
split 1
split 2
split 3
split 4
read
Worker
Worker
Worker
local write
remote read
intermediate result
Worker
Worker
write
output file 0
output file 1
input files are split in Key/Value pairs
mappers on each worker process input data and create intermediate results that are stored locally
in the shuffle phase, intermediate results with the same key value are grouped together
workers merge intermediate results depending on the user defined reduce function.

Figure 50: Execution of a MapReduce job.

For better understanding the process of a MapReduce job, consider the word-count example provided by Dean and Ghemawat [23]. Imagine having a large set of text documents and wanting to know how many times a word is mentioned within them. To solve this problem, it is necessary to counts overall occurrences of each word. This task can be fulfilled by a MapReduce program. The MapReduce program takes a set of input key/value pairs (document name/content) to create the desired output key/value pairs (word/count). In the map phase, the workers execute the user defined map function to count the words in each document. The map function takes a document as input and creates a new key/value pair for each word as an intermediate result. The key is the word itself and the value is its occurrence, which initially is always "1". The following pseudo-code is an example of how this map function may look like:

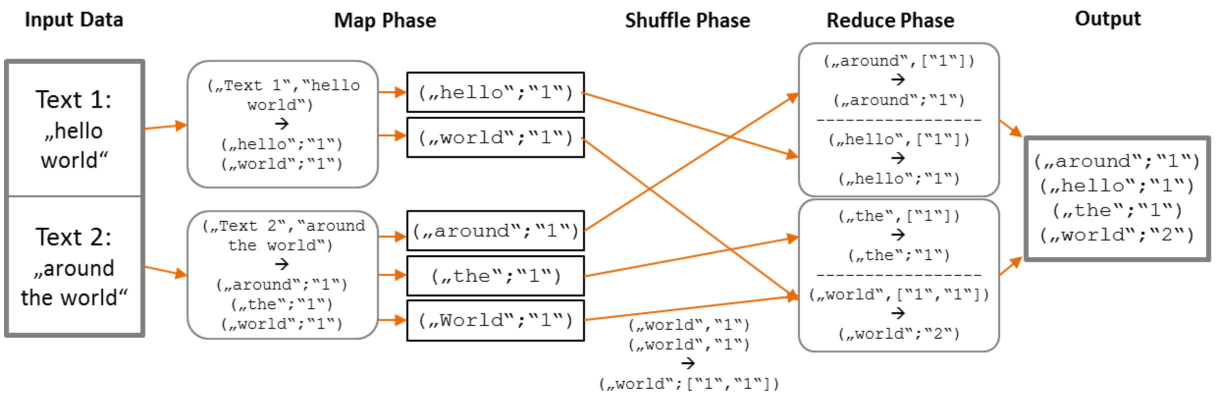
```
map(String key, String value):
   //key: document name
```

```
    //value: document content

    for each word w in value:
       create IntermediateResult(w,1)
```

If a map process has finished, intermediate results are sorted by key and saved to local disk. They are not saved to HDFS to avoid unnecessary duplication. The shuffle phase describes the process of getting intermediate results to the workers for the reduce phase. During shuffle, all intermediate results are grouped together depending on their key value. Finally, the reducers add up the intermediate results. Their input is a key/value pair where the key is the word and the value is a list of its occurrences. The following pseudo-code shows the concept of such a reduce function:

```
reduce(String key, Iterator values):
   //key: word
   //value: list of counts

   int result = 0
   for each v in values:
      result += ParseInt(v)
   return result
```

Figure 51 illustrates the word-count example with its inputs and outputs during each phase. The two texts "hello world" and "around the world" have been chosen as an example. The map- and reduce functions follow the logic of the above defined pseudo code examples.



Input Data
Map Phase
Shuffle Phase
Reduce Phase
Output
Text 1: „hello world"
Text 2: „around the world"
(„Text 1","hello world") → („hello";"1") („world";"1")
(„Text 2","around the world") → („around";"1") („the";"1") („world";"1")
(„hello";"1")
(„world";"1")
(„around";"1")
(„the";"1")
(„World";"1")
(„world","1") („world","1") → („world";["1","1"])
(„around",["1"]) → („around";"1")
(„hello",["1"]) → („hello";"1")
(„the",["1"]) → („the";"1")
(„world",["1","1"]) → („world";"2")
(„around";"1") („hello";"1") („the";"1") („world";"2")

Figure 51: MapReduce example: counting words in text documents.

6.2.2 Apache Spark

Apache Spark[53] is a framework for distributed, parallel processing [191]. In comparison to Hadoop, Spark utilizes in-memory technology to achieve better performance. Spark started as a project at the university of California, Berkley, and was later donated to the Apache foundation, where it became a top level Apache project in 2014[54]. The framework is written in Java and Scala, and therefore runs on any JVM capable system. It offers APIs in Scala, Java, Python and R. Today, over 50 contributors actively develop the framework[55].

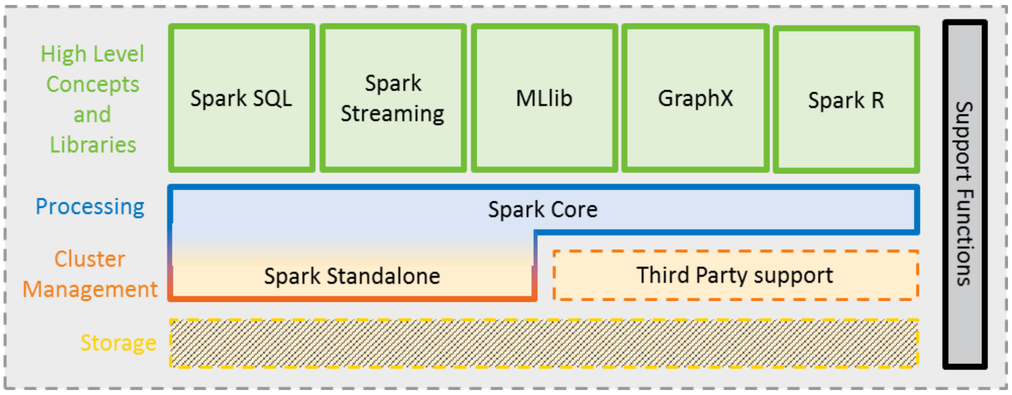


Figure 52: Embedding Spark in the architecture of a Big Data analytic ecosystem.

As shown in Figure 52, Spark consists in its current version 2.2.1[56] (released 1.12.2017), of 6 main modules. The centre of Spark is Spark Core. It is responsible for executing and monitoring data processing. Spark Core also includes a cluster management module and is therefore able to run its own cluster (Spark Standalone). However, Spark also provides native support for third party cluster management frameworks Hadoop YARN and Apache Mesos[57]. The other 5 components (Spark SQL, Spark Streaming, MLlib, GraphX, and Spark R) offer high level features on top of Spark Core. Spark SQL provides an SQL-like interface to work with structured data. Spark Streaming enables processing of data streams. MLlib is a library of machine learning algorithms for Big Data analysis. GraphX provides an API for graphs and graph-parallel computation and Spark R is an API for R. Additionally, Spark offers support functions such as central logging and a web based graphical user interface for job monitoring. In contrast to Hadoop, Spark does not provide a distributed storage system. It is dependent on other frameworks like Hadoop HDFS. The following sections presents the most prominent modules of Spark (Spark Core, Spark SQL,

[53] https://spark.apache.org/
[54] https://blogs.apache.org/foundation/entry/the_apache_software_foundation_announces50
[55] https://spark.apache.org/committers.html
[56] https://spark.apache.org/docs/2.2.1/
[57] https://mesos.apache.org/

Spark Streaming and MLib). Since neither Spark R nor GraphX are necessary for a PMB implementation, they are not portrayed.

Spark Core

Spark Core is the data processing engine of Spark [191]. It is responsible for data processing as well as management functions such as scheduling and monitoring tasks. Spark Core implements, and is built around a programming abstraction called Resilient Distributed Data Sets (RDD) [24].

The main characteristic of RDDs is that they may exist beyond the lifetime of a job. This allows for the reuse of intermediate data across multiple computations. While reusing results in Hadoop would require the data to be saved on disk, RDDs can be stored in-memory. The reuse of data is especially common in iterative machine learning and graph algorithms. Reusing data also benefits interactive use cases, for example running multiple ad-hoc queries against the same set of data. Stored intermediate query results can be used to speed up future ones.

As shown in Figure 53, RDDs are created from data in stable storage or other RDDs through deterministic operations called transformations [24]. Transformations include mapping data to RDDs (*map)*, *filter* or *join* operations. Once a RDD is created, it cannot be changed. RDDs are read-only. They are split up into partitions that are distributed around the cluster to enable distributed processing.

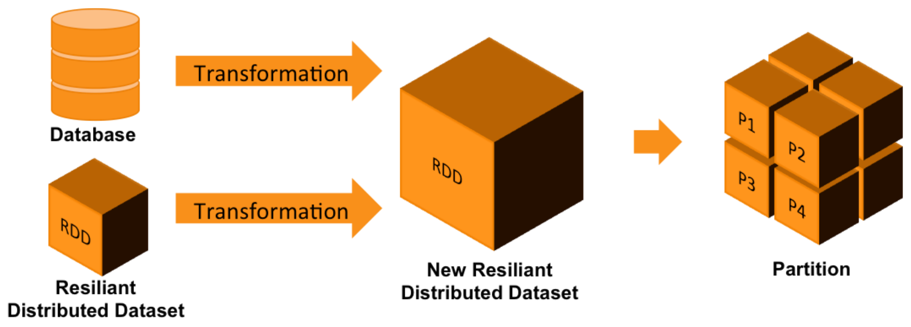

Database
Transformation
RDD
Resiliant Distributed Dataset
Transformation
RDD
New Resiliant Distributed Dataset
P1
P2
P3
P4
Partition

Figure 53: Creating RDDs and splitting them into Partitions

RDDs are fault tolerant [24]. In many distributed systems like Hadoop HDFS, fault tolerance is provided by replicating data on multiple cluster nodes [60]. This is associated with a lot of network traffic and data replication. In contrast to that, RDDs ensure fault tolerance by saving transformations needed for creating it, rather than its data. Storing transformations instead of raw data represents just a fraction of the data volume but still ensures there is always enough information available to rebuild a RDD if lost.

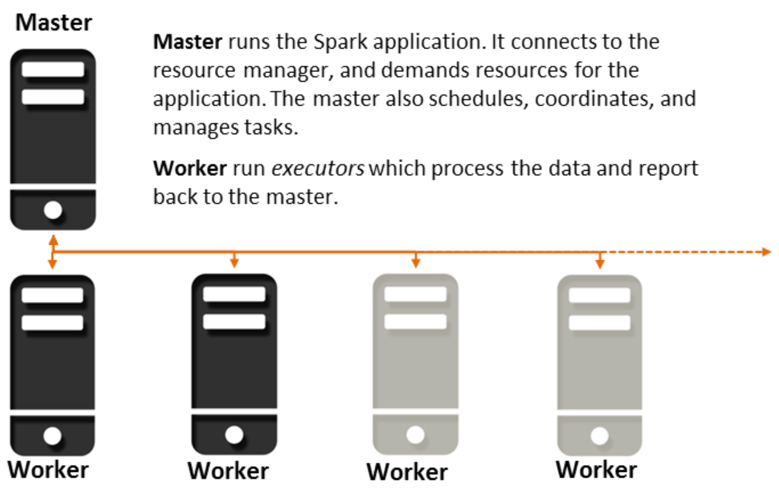

Master
Master runs the Spark application. It connects to the resource manager, and demands resources for the application. The master also schedules, coordinates, and manages tasks.
Worker run *executors* which process the data and report back to the master.
Worker
Worker
Worker
Worker

Figure 54: Master/slave architecture of Spark.

As illustrated in Figure 54, Spark follows a master/slave architecture similar to the one of Hadoop. The master runs the Spark application that is provided through the so-called *Driver Program*. The Driver defines the RDDs and their analysis. It launches the SparkContext, which coordinates the application. The SparkContext connects to the cluster manager to demand resources, schedules tasks, and monitors them. The slaves or workers process data. They host *Executors* which execute tasks provided by the SparkContext. They report back to the SparkApplication.

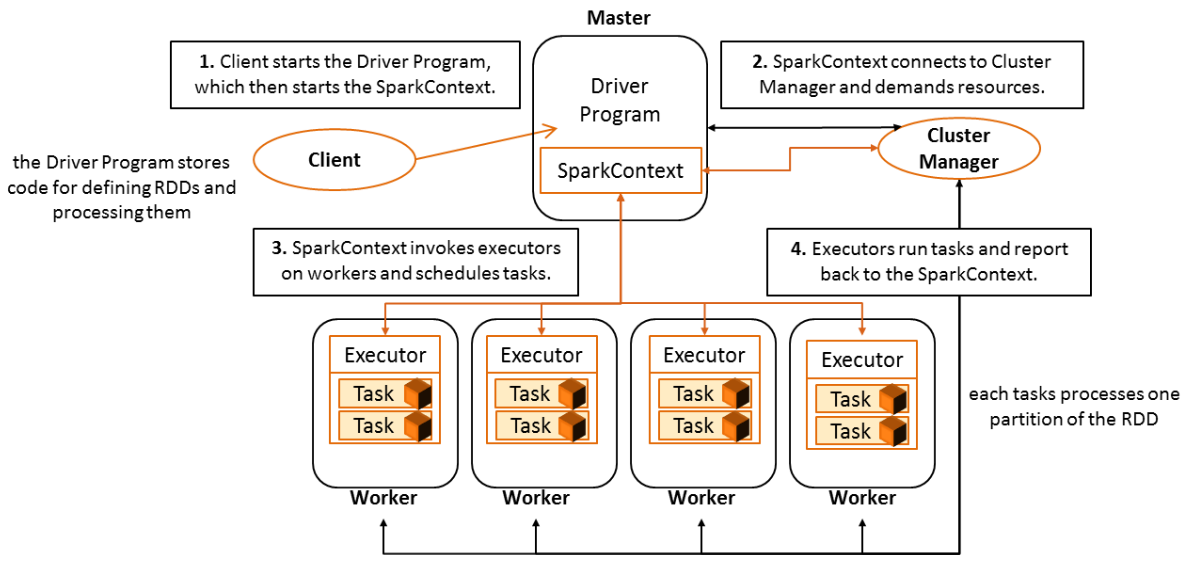

Master
1. Client starts the Driver Program, which then starts the SparkContext.
Driver Program
SparkContext
2. SparkContext connects to Cluster Manager and demands resources.
Cluster Manager
Client
the Driver Program stores code for defining RDDs and processing them
3. SparkContext invokes executors on workers and schedules tasks.
4. Executors run tasks and report back to the SparkContext.
Executor
Task
Task
Worker
each tasks processes one partition of the RDD

Figure 55: Workflow of a Spark application.

Figure 55 presents the workflow of starting an application inside Spark. To run a Spark application a user has to define a Driver Program, which defines RDDs and their processing. Each Spark application is coordinated by its own SparkContext, which is launched on the master. The SparkContext takes the user defined transformations and translates them into a directed acyclic graph. The graph is then submitted to the scheduler. After scheduling, the SparkContext connects to the cluster manager to demand computing resources in the cluster. Then the SparkContext connects to the allocated workers and invokes Executors, which process RDD partitions. Each worker runs one Executor per application. After completion, Executors report results back to the SparkContext.

Spark SQL

Spark SQL[58] is Sparks module to work with structured data and was introduced in 2014 during the release of Spark 1.0.0[59]. It allows users to access many popular data sources, including Hive, Avro, Parquet, ORC, JSON, and JDBC. Spark SQL enables clients to perform SQL-queries but also offers APIs in Scala, Java, and Python to connect to data sources programmatically [67]. Independent of the utilized language, all queries are executed by the same engine, allowing for high flexibility and developers to switch languages depending on the use case.

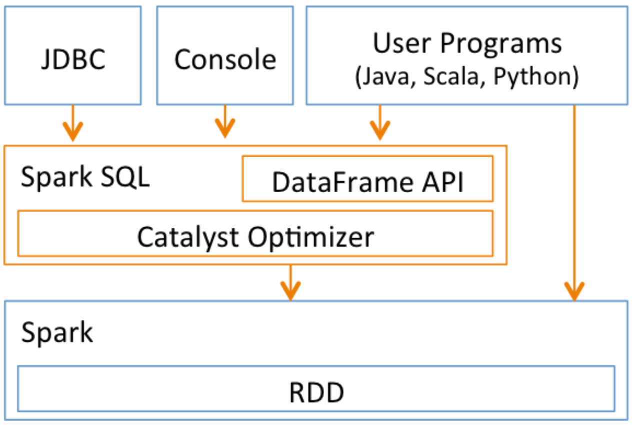


Figure 56: Interfaces of Spark SQL and interaction with Spark.

Figure 56 displays the interfaces of Spark SQL and how it interacts with Spark Core [67]. Spark SQL offers a SQL interface that can be accessed through JDBC/ODBC or through command-line console. Additionally, Spark SQL offers a DataFrame API for user programs. DataFrames are the main abstraction in Spark SQL. They are equivalent to a table in a relational database. DataFrames can be created either from relational data sources or RDDs. The Catalyst Optimizer is the internal interface to Spark Core. It plans queries and translates them into Spark executable code [67].

[58] https://spark.apache.org/sql/
[59] https://spark.apache.org/news/spark-1-0-0-released.html

Spark Streaming

Spark Streaming[60] is Sparks dedicated processing engine for data streams. It is designed to offer real-time data processing and was introduced in 2013 as part of Sparks release 0.7.0[61].

The general idea behind Spark Streaming was to create a system capable of real time data processing that is able to provide fast recovery in case of failure [22]. Other stream processing systems often utilize a continuous processing model, where data is processed as soon as it enters the system. However, continuous processing results in high costs for providing fault tolerance [22]. Instead of continuous processing, Spark Streaming implements a mini-batch processing approach as shown in Figure 57. Input data of continuous data streams is first stored and grouped into small data sets. Periodically these data sets are processed in batch using the Spark Core processing engine. The time intervals between batch processes are very small, achieving almost real-time computation speed.

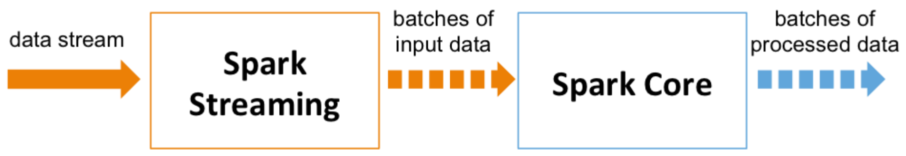


Figure 57: Mini-batch processing model of Spark Streaming.

Structuring stream processing as a set of short, stateless, deterministic tasks is called discretized streams [22]. To highlight the advantages of discretized streams, consider traditional approaches for providing fault tolerance in continuous data processing system. As displayed in Figure 58, there are two main methods. The first method sustains fault tolerance through replication. The entire data processing stream is duplicated to provide a backup if one stream fails, doubling the hardware requirements of the system. Furthermore, to ensure identical outcomes, all nodes down the stream must be synchronized. The second method implements upstream backups. Each node retains a copy of sent messages. If a failure occurs, all messages are resent to a backup machine. The recovery therefore rests solely on the backup machine, which has to reprocess data on its own.

[60] https://spark.apache.org/streaming/
[61] https://spark.apache.org/releases/spark-release-0-7-0.html

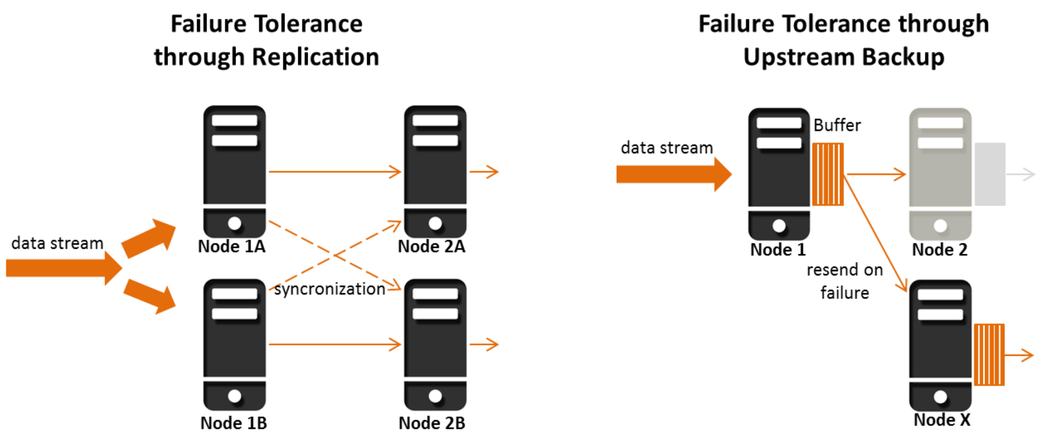


Figure 58: Traditional approaches for providing fault tolerance in continuous data processing systems.

Spark Streaming uses discretized streams to provide fault tolerance. Figure 59 shows the processing model of discretized streams. Input data is stored in the form of RDDs among the cluster. After a short period of time, the accumulated data is processed within a batch process using the Spark Core execution engine. Spark Streaming stores all operations, a discretized stream goes through. In case of failure, the affected stream checks its history and repeats all lost transformation steps. The recomputation can be done in parallel on multiple nodes.

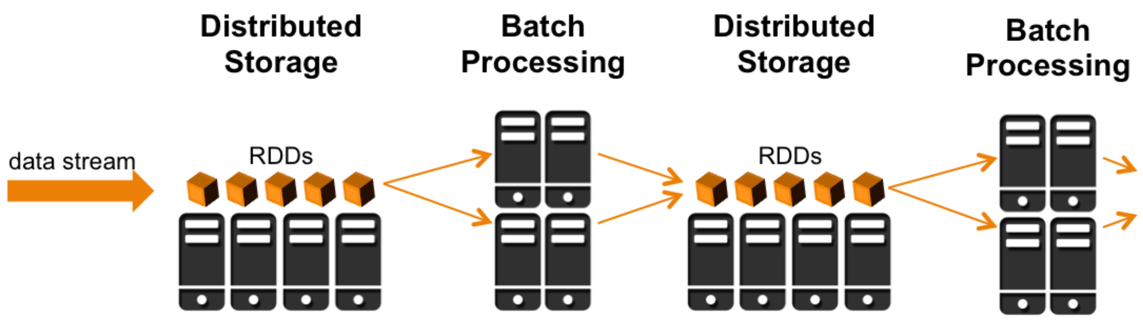


Figure 59: Processing model of discretized streams.

The Spark Streaming library offers APIs in Java, Scala and Python. Spark Streaming allows for custom data source configuration but natively supports HDFS, Flume, Kafka, Twitter, and ZeroMQ.

MLlib

MLlib[62] is a library of machine learning algorithms and had its debut in Spark version 0.8[63]. Its development began in 2012 as part of the MLBase project [92] and was open sourced in September 2013. MLlib is written in Scala and provides interfaces for Java,

[62] https://spark.apache.org/mllib/
[63] https://spark.apache.org/releases/spark-release-0-8-0.html

Scala, Python and R. It offers a wider variety of implemented algorithms for classification, regression clustering, and collaborative filtering [91].

6.2.3 Mahout

Apache Mahout[64] is a programming framework for creating distributed machine learning algorithms. Additionally, Mahout provides some premade machine learning algorithms for Hadoop MapReduce, Spark, H2O and Flink. In 2016, Mahout introduced a new math environment called Samsara with its 0.11.1 release[65]. In Samsara, developers can specify machine learning algorithms in an abstract language similar to R or MATLAB. The Mahout project started in 2009 as part of Apache Lucene[66] (a text search engine library) with the goal to provide scalable machine learning algorithms[67]. In 2010, Mahout split from Lucene to become an independent Apache project. In recent years Mahout shifted its focus from a library of machine learning algorithms to an environment for building them.

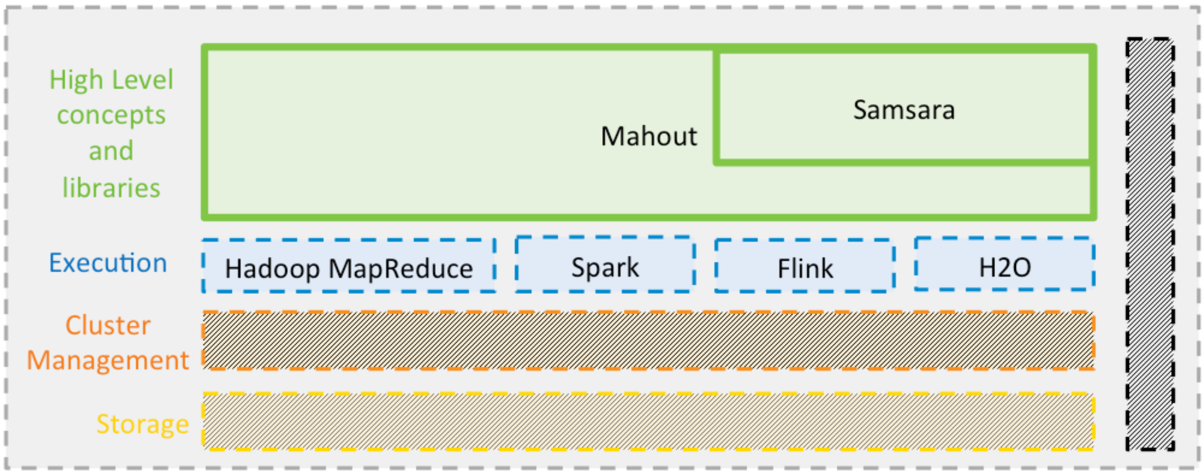


Figure 60: Embedding Mahout in the architecture of a Big Data analytic ecosystem.

Considering the architecture of a Big Data ecosystem, Mahout is a high level library on top of an existing Big Data environment as shown in Figure 60. Mahout natively supports the execution engines of Hadoop MapReduce, Spark, Flink and H2O.

[64] https://mahout.apache.org/
[65] https://mahout.apache.org/general/release-notes.html
[66] https://lucene.apache.org/
[67] https://lucidworks.com/2009/04/07/apache-mahout-01-released/

6.3 Testing Environment

6.3.1 The Raspberry Pi

The Raspberry Pi[68] is a series of single-board computers developed in the United Kingdom by the Raspberry Pi Foundation. It is a low cost, high performance computer, intended to promote computer science in education and developing countries. Since its first release, over 17 million Raspberry Pi's have been sold worldwide [192]. The current model of the Raspberry Pi is the Raspberry Pi 3 Model B[69], which is illustrated in Figure 61. Table 15 lists its specifications.

Table 15: Specifications of the Raspberry Pi 3 Model B.

RaspberryPi Model B specifiations
1.2 GHz 64-bit quad-core ARMv8 CPU 1 GB RAM Micro SD card slot 4 USB ports Full HDMI port Ethernet port Combined 3.5mm audio jack and composite video Camera interface (CSI) Display interface (DSI) VideoCore IV 3D graphics core 40 GPIO pins

Figure 61: Raspberry Pi Model 3 B[70].

6.3.2 Raspberry Pi Based Computer Clusters

Due to its low costs, small size, and good performance, the Raspberry Pi is a multipurpose tool that can be utilized in many different application areas such as education, media centres and game machine[71]. These attributes also promote the Raspberry Pi as basis for computer clusters – an approach, whose viability has been explored in scientific papers. Cox et al. [193] introduce Iridis-Pi, a cluster comprising of 64 Raspberry Pi Model 3 B. Fung et al. [194] present the Glasgow Raspberry Pi Cloud, a scale model of a datacentre composed of clusters of Raspberry Pi devices. Abrahamsson et al. [195] constructed the

[68] http://www.raspberrypi.org/
[69] http://www.raspberrypi.org/products/raspberry-pi-3-model-b/
[70] http://www.Raspberry Pi-spy.co.uk/2016/02/introducing-the-raspberry-pi-3-model-b/
[71] https://www.raspberrypi.org/forums/

Bolzano Raspberry Pi cluster, which connects 300 Raspberry Pi 3 model B to an energy efficient computing cluster. Ashari and Riasetiawan [196] compare the performance of a cluster comprising of 14 Raspberry Pi Model B to a the performance of the multicore processor chips Intel[72] i5 and i7 in the area of matrix calculations. Their results show that the Intel chips outperform the cluster significantly. Schot [197] analyses the capabilities of the Raspberry Pi 2 as basis for a micro data centre. The micro data centre consists of 8 Raspberry Pi 2 running Hadoop and shows low power consumption with a moderate performance.

Overall, the Raspberry Pi can be considered a viable basis for computer clusters. Its low costs and high performance allow to create cluster environments on a budget. Therefore, a Raspberry Pi cluster was chosen as hardware for running the two Big Data analytic ecosystems.

6.3.3 Raspberry Pi as Basis for PMB Implementations

There are good reasons for choosing the Raspberry Pi as basis for the cluster running the Big Data analytic ecosystems. Firstly, the Raspberry Pi uses an SD-Card as hard drive. Its content can be stored as an image, enabling simple replication of experiments. Secondly, distributed processing frameworks are designed to work best on clusters consisting of multiple nodes of the same hardware. And thirdly, the Raspberry Pi Model 3 is affordable.

Therefore, the PMB is implemented on two Big Data analytic ecosystems running on a Raspberry Pi cluster. The utilized Raspberry Pi cluster consists of five Raspberry Pi 3 Model B. The Raspberry Pi's operate on Raspbian[73], a lightweight Linux Debian distribution. For storage, each unit has a MicroSD card with a capacity of 32 GB. The individual units are connected by LAN, using a standard 8-Port network switch as connection device.

[72] http://www.intel.com/
[73] http://www.raspbian.org/

6.4 Benchmark Execution

The PMB is implemented on two Big Data analytic ecosystems. This section presents their configuration, implementation details, and benchmark results. To prevent distortion effects of outliers, each benchmark is performed ten times and the results are calculated as the mean of these ten repetitions.

6.4.1 Cluster Network Setup

The cluster consists of five Raspberry Pi 3 Model B, which are connected to an 8-port gigabit switch via Ethernet cable. As illustrated in Figure 62, the switch is connected to a router, which manages the network. The Raspberry Pis are capable of network transfers speeds of 100 Mbit/s, the switch and router are able to achieve transfer speeds of up to 1.000 Mbit/s. Therefore, the single Ethernet cable between switch and router is able handle all incoming requests from the five Raspberry Pis.

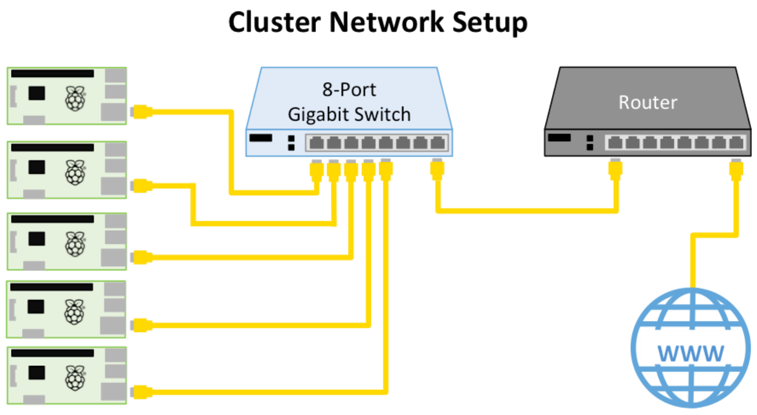


Figure 62: Test environment cluster network setup.

The Raspberry Pis operate on Raspbian74, a free operating system based on Debian, which is optimised for the Raspberry Pi hardware. In the network, the individual Raspberry Pis are named as master, slave-01, slave-02, slave-03, and slave-04. For easy identification, are assigned static ip-addresses by the router (192.168.1.10-192.168.1.10). The names are assigned on each machine by editing the hostname file located at /etc/hostname. External access to the cluster can be achieved via ssh75 or ftp76. SSH is used to grant password free access between all Raspberry Pis, since the orchestration framework YARN needs all nodes to be able to communicate with each other directly and without password authentication.

[74] https://www.raspberrypi.org/documentation/raspbian/
[75] https://www.ssh.com/
[76] https://tools.ietf.org/html/rfc959

6.4.2 Ecosystem 1: HDFS, YARN, MapReduce, Mahout, Hive

Ecosystem 1 combines the most popular Big Data analytic frameworks Hadoop, Mahout, and Hive (see Figure 63). It is based on Hadoop, covering the filesystem HDFS, the resource manager YARN and the execution engine MapReduce. Mahout serves as the machine learning library and Hive as the sql interface for data access and preparation.

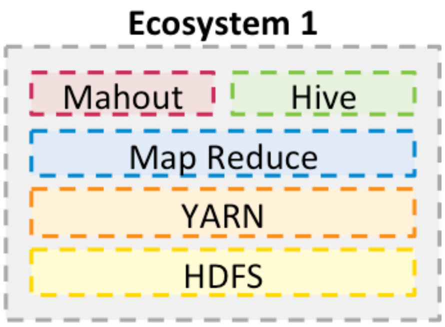


Figure 63: Ecosystem 1.

Ecosystem 1: Cluster Configuration

This section specifies the frameworks, their respective versions, and configurations used for implementing the PMB. Table 16 lists the frameworks and their respective versions. For Hadoop, version 2.7.5 is chosen, since the newest version Hadoop 3.0.0 was still in Alpha release phase during the development of PMB. For Hive and Mahout, the most recent versions 2.3.2 and 0.13.0 are selected.

Table 16: Ecosystem 1: frameworks and versions.

Frameworks	Version
Hadoop	2.7.5
Hive	2.3.2
Mahout	0.13.0

Hadoop and Hive must be configured to run on the limited resources of the Raspberry Pi cluster. All configurations are done by editing xml-files within the respective installation folder on each node. Table 17 list the adjusted parameters, their values, and in which files they are set.

Table 17: Cluster specific configuration of frameworks in Ecosystem 1.

File	Parameter	Value	Comment
hdfs-site.xml	dfs-replication	3	number of data replication
	dfs.blocksize	67108864	the size of HDFS blocks in bytes (here 64 MB)
mapred-site .xml	mapreduce.reduce. memory.mb	512	max memory in MB assigned for reduce tasks
	mapreduce.map. memory.mb	512	max memory assigned in MB for map tasks
	yarn.app.mapreduce.am. resource.mb	512	size of a container requested from YARN in MB
yarn-site.xml	yarn.nodemanager. resource.memory-mb	1024	memory of the NodeManager in MB
	yarn.scheduler.minimum-allocation-mb	128	minimum allocation for every container request in MB
	yarn.scheduler.maximum-allocation-mb	1024	maximum allocation for every container request in MB
hive-site.xml	hive.execution.engine	mr	sets MapReduce as execution engine

Ecosystem 1: PMB Implementation

The PMB is implemented using a Bash[77] script. The script invokes Hadoop commands, calls Hive scripts, and executes a java program for training and testing the decision tree. It also measures execution time and saves performance results to a csv file. Figure 64 illustrates how Phase 1 of the PMB (see 5.3.1) is implemented on Ecosystem 1. The first step of Phase 1, preprocessing, is separated into data ingestion and data processing. For data ingestion, the data is copied into HDFS using its command line interface. Hive then creates corresponding tables for easy data access. For data processing, Hive transforms the individual data sets into the final processed data set according to a Hive script written in HiveQL[78], a language similar to SQL. Hive creates a MapReduce application that requests resources from YARN. The resulting data set is split into training set and test set and stored in HDFS. In the second step of Phase 1, training, a Java program is invoked. The program trains a decision tree implemented by the Mahout library using the training set stored in HDFS. The resulting decision tree model is saved in HDFS. In the third step of Phase 1, testing, a Java program loads the trained model and tests it against the test set.

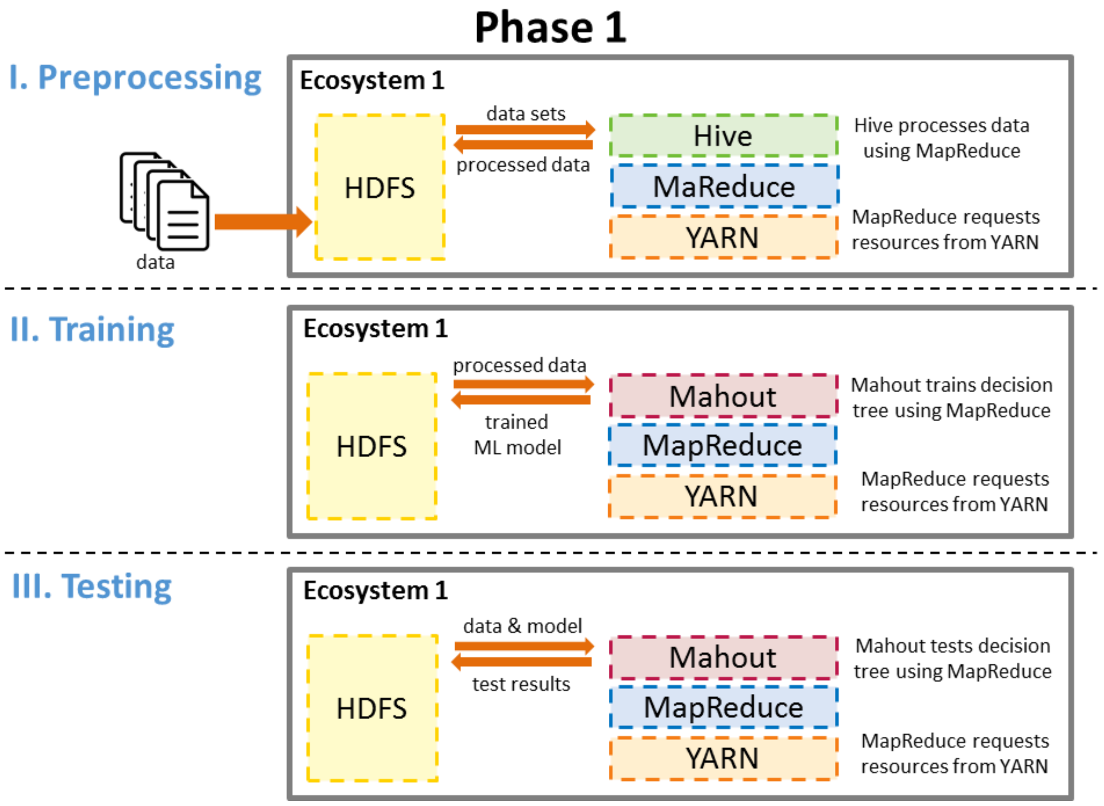

Phase 1
I. Preprocessing
Ecosystem 1
data
HDFS
data sets
processed data
Hive
MaReduce
YARN
Hive processes data using MapReduce
MapReduce requests resources from YARN
II. Training
Ecosystem 1
HDFS
processed data
trained ML model
Mahout
MapReduce
YARN
Mahout trains decision tree using MapReduce
MapReduce requests resources from YARN
III. Testing
Ecosystem 1
HDFS
data & model
test results
Mahout
MapReduce
YARN
Mahout tests decision tree using MapReduce
MapReduce requests resources from YARN

Figure 64: Implementation of Phase 1 of the PMB on Ecosystem 1.

[77] https://www.gnu.org/software/bash/
[78] https://cwiki.apache.org/confluence/display/Hive/LanguageManual

Phase 2 of the PMB (5.3.2) consists of preprocessing new data and scoring it (see Figure 65). During the first step, preprocessing, data is ingested into HDFS using the Hadoop command line interface. Then the data is preprocessed in Hive according to a HiveQL script that calculates all values necessary for prediction. Hive translates the HiveQL commands into MapReduce jobs, which in turn requests cluster resources from YARN. For the second step of Phase 2, scoring, a Java program loads the Mahout decision tree model created during Phase 1 and classifies the new processed data. The results are saved into HDFS.

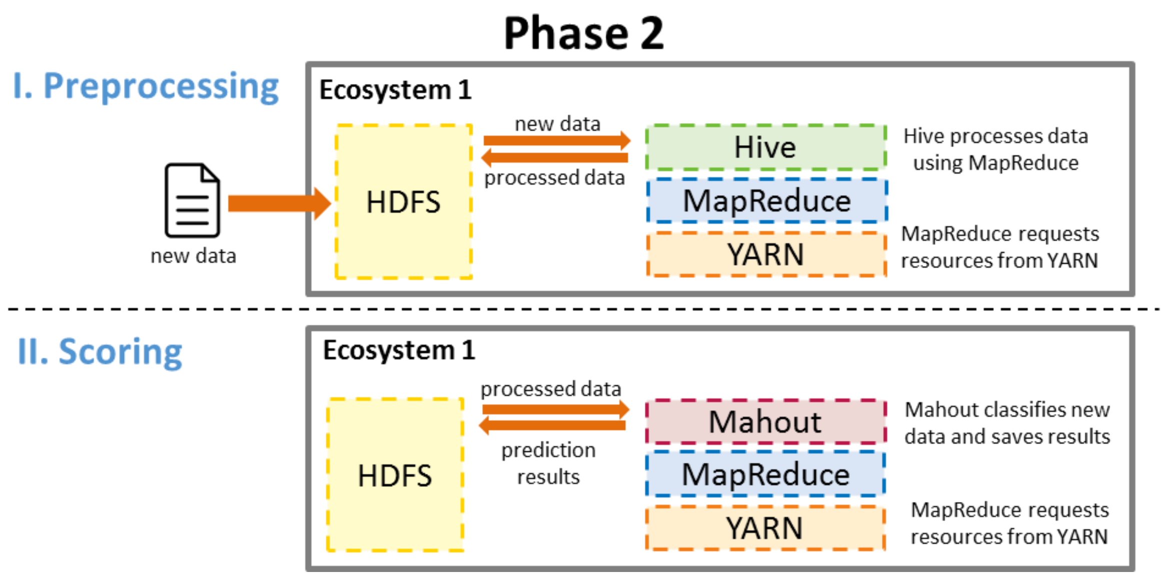


Figure 65: Implementation of Phase 2 of the PMB on Ecosystem 1.

For each step of Phase 1 and Phase 2, the execution time is measured. Where possible, the execution time is further divided into subcategories such as time for initializing a framework, loading data, and processing it. The results of the PMB implementation are shown below.

Ecosystem 1: PMB Results

The performance of Ecosystem 1 is determined by measuring the execution time of Phase 1 and Phase 2 as defined by the PMB in seconds. Both phases are executed with three different data set sizes. In Phase 1 the data sets vary in the number of machines they contain. The small data size contains 33 machines, the medium size 66 machines, and the big size 100 machines. During Phase 2 the data sizes vary in the number of data items that need to be classified. The data sizes for Phase 2 are 10 items (small), 100 items (medium), and 1.000 items (big). Figure 66 shows the PMB performance results of Ecosystem 1.

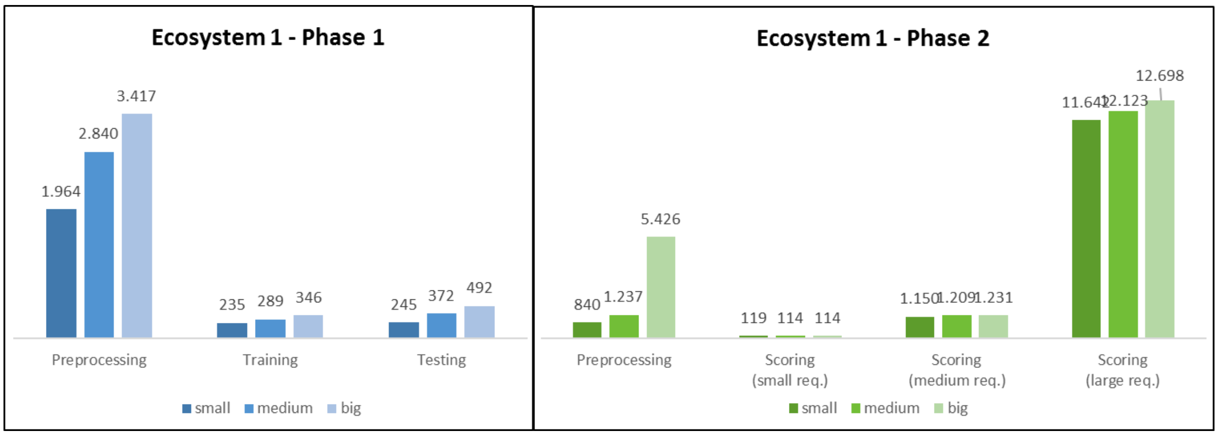


Figure 66: PMB results of Ecosystem 1 in seconds.

In Phase 1 (Figure 66, blue), most of the execution time is accumulated during preprocessing of data in Hive (blue bar graph, left). Hive needs 876 seconds more for processing the medium data set (66 machines) than processing the small data set (33 machines). It takes only 577 additional seconds to process the big data set (100 machines). This observation suggests good performance scalability with increasing data volume since adding machines decreases the average preprocessing time per machine. During training and testing, the scaling effect is almost linear. For training, the time increase is 54 seconds for the medium data set, and 57 seconds for the big data set (blue bar graph, centre). For testing, additional time of 127 seconds (medium data set) and 120 seconds (big data set) is necessary (blue bar graph, right).

In Phase 2 (Figure 66, green), data sizes do not grow linear as in Phase 1 but by a factor of ten. Therefore, even though the absolute execution time of preprocessing during Phase 2 increases rapidly with each data set, the processing time per data item decreases (green bar graph, left). In particular, the processing time per item decreases from 84 seconds (small data set) to 12,4 seconds (medium data set) to 5,4 seconds (big data set). Scoring is separated into three request sizes consisting of 1 request (small), 10 requests (medium), and 100 requests (big) respectively. In each request, the entire data set is classified. The collected data shows that an increase in data size has only marginal effects on the execution time of the requests. For the small request size, the execution time for all three data sets is between 114 and 119 seconds (green bar graph, centre left). When comparing the measurements of small, medium, and big request Scoring (green bar graph, right three) it shows that execution time scales linear with the requests size. Increasing the number of requests by ten results in roughly 10 times longer execution time (i.e. execution time for the small data set increases from 119 to 1.150 to 11.642 seconds).

Figure 67 provides a detailed view of Phase 1 (blue) and Phase 2 (green). The individual steps discussed above are broken down further into sub steps for in-depth analysis.

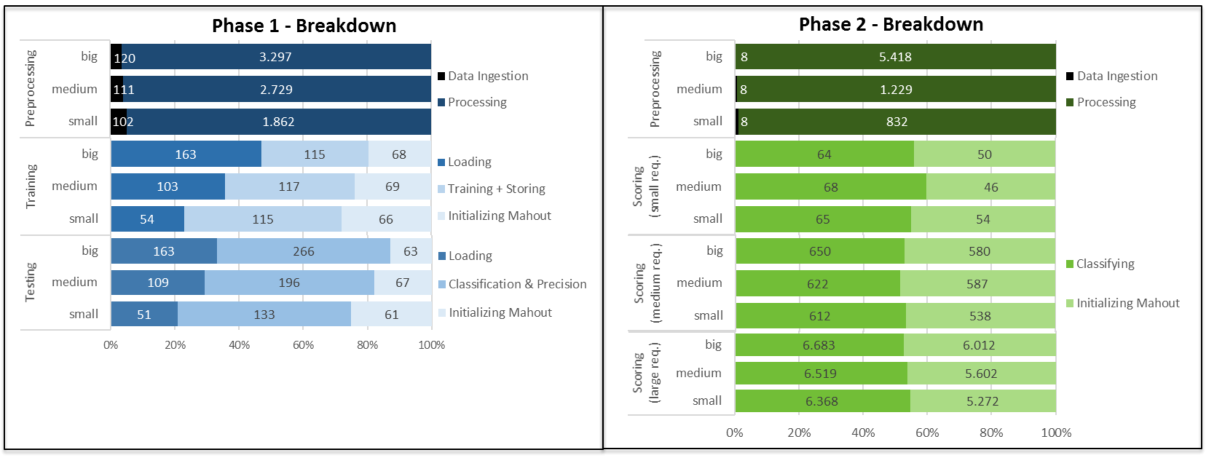


Figure 67: Ecosystem 1: Performance breakdown of Phase 1 and Phase 2 in seconds.

In Phase 1, preprocessing (Figure 67, blue bar graph, top) is divided into the two tasks of data ingestion and processing. Most of preprocessing is spent processing the data with data ingestion taking a maximum of 5% of overall time. The breakdown of training (blue bar graph, centre) shows that only the loading of the data is affected by the file size. The time for initializing Mahout (66-69 seconds) and training the model (115-117 second) is almost constant throughout all three data sets. During testing (blue bar graph, bottom), the time for loading and classifying data increases with growing data size, while the time for initializing Mahout stays at 61-67 seconds.

Preprocessing of Phase 2 (Figure 67, green bar graph, top) almost exclusively consists of processing the data with only 8 seconds spent for data ingestion. All three types of scoring (small, medium, and large request) show a similar percentage distribution between classifying data and initializing Mahout (green bar graph, bottom three sections). Classifying data accounts for minimum 51% (medium request/medium data size) to maximum 61% (small request/medium data size) of overall scoring execution time.

6.4.3 Ecosystem 2: HDFS, YARN, Spark, MLlib, Hive

Ecosystem 2 implements the second most popular Big Data analytic frameworks Spark and MLlib (see Figure 68). The remaining frameworks are the same as in Ecosystem 1, with the difference of Hive not running on MapReduce but Spark. In contrast to the Hadoop MapReduce execution engine, Spark is based on in-memory technology.

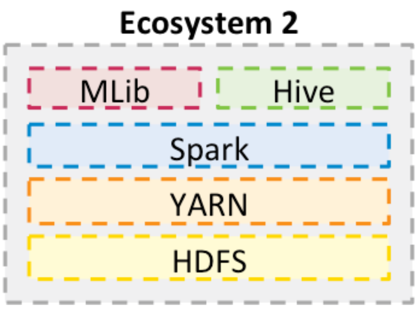


Figure 68: Ecosystem 2.

Ecosystem 2: Cluster Configuration

Ecosystem 2 is built upon Spark, Hadoop, and Hive. For Hadoop and Hive the same versions as in Ecosystem 1 are selected. For Spark and its machine learning library MLlib version 2.2.1 (prebuilt for Apache Hadoop 2.7 and later) is chosen. Figure 19 lists the frameworks and their respective versions.

Table 18: Ecosystem 2: frameworks and versions.

Frameworks	Version
Spark	2.2.1
Hadoop	2.7.5
Hive	2.3.2

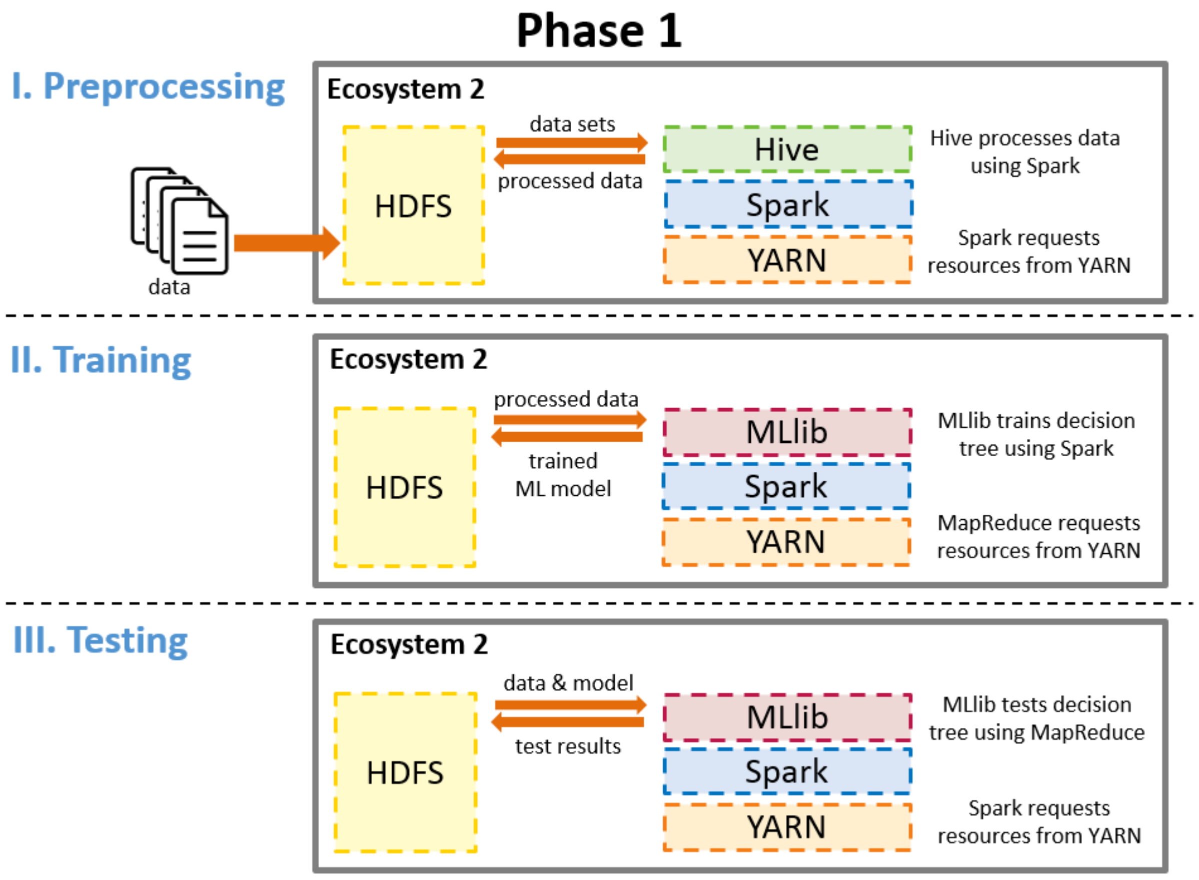
Similar to Ecosystem 1, Spark must be configured to run on the limited resources provided by the cluster. In particular, available resources, storage block size, and execution engine for Hive must be configured. Table 19 list the individual parameters, their values, and in which files they are set.

Table 19: Cluster specific configuration of frameworks in Ecosystem 2.

File	Parameter	Value	Comment
hdfs-site .xml	dfs-replication	3	how many time data is replicated on the system
	dfs.blocksize	67108864	the size of HDFS blocks in bytes
spark-env .sh	SPARK_EXECUTOR_MEMORY	512	memory to use per executor process, in MB
	SPARK_DRIVER_MEMORY	512	memory to use for the driver process (i.e. SparkContext initialization) in MB
	SPARK_WORKER_MEMORY	512	memory to use per worker process in MB
	SPARK_DAEMON_MEMORY	512	memory to use per deamon process
yarn-site .xml	yarn.nodemanager. resource.memory-mb	1024	memory of the NodeManager in MB
	yarn.scheduler.minimum-allocation-mb	128	minimum allocation for every container request in MB
	yarn.scheduler.maximum-allocation-mb	1024	maximum allocation for every container request in MB
hive-site .xml	hive.execution.engine	spark	sets the execution engine of hive to spark

Ecosystem 2: PMB Implementation

To offer maximum comparability of the PMB results of Ecosystem 1 and Ecosystem 2, the respective implementations are as similar as possible. The PMB in Ecosystem 2 is also implemented using a Bash script. HDFS and Hive are used in both ecosystems. Therefore, all data ingestion and preprocessing steps use the same HDFS commands and HiveQL scripts. The only difference is present in the Java program loading the MLlib library for training and testing the decision tree, since Mahout and MLlib offer different APIs. However, the overall procedure in both Java programs was kept the same. Figure 69 illustrates the implementation of PMB Phase 1 (see 5.3.1) in Ecosystem 2. Phase 1 consists of preprocessing the data, training the decision tree model, and testing the trained model. The data is ingested using HDFS and processed in Hive, which runs on Spark. For training, a Java program is executed, that creates a decision tree model using MLlib and the processed data. The model is stored in HDFS and tested using MLlib on Spark.



Phase 1
I. Preprocessing
Ecosystem 2
data
HDFS
data sets
processed data
Hive
Spark
YARN
Hive processes data using Spark
Spark requests resources from YARN
II. Training
Ecosystem 2
HDFS
processed data
trained ML model
MLlib
Spark
YARN
MLlib trains decision tree using Spark
MapReduce requests resources from YARN
III. Testing
Ecosystem 2
HDFS
data & model
test results
MLlib
Spark
YARN
MLlib tests decision tree using MapReduce
Spark requests resources from YARN

Figure 69: Implementation of Phase 1 of the PMB on Ecosystem 2.

In Phase 2 of the PMB (5.3.2), new data is preprocessed and scored. Figure 70 illustrates how Phase 2 is implemented in Ecosystem 2. After ingesting the data into HDFS, it is processed in Hive using the same HiveQL scripts developed during the implementation of Phase 2 in Ecosystem 1. However, since Hive uses Spark and not MapReduce, the HiveQL commands are translated into RDD transformations rather than MapReduce jobs. These RDD transformations are then executed by the Spark execution engine. During the second step of scoring, a Java program loads the trained MLlib decision tree model from HDFS and classifies the processed new data. The results are saved to HDFS.

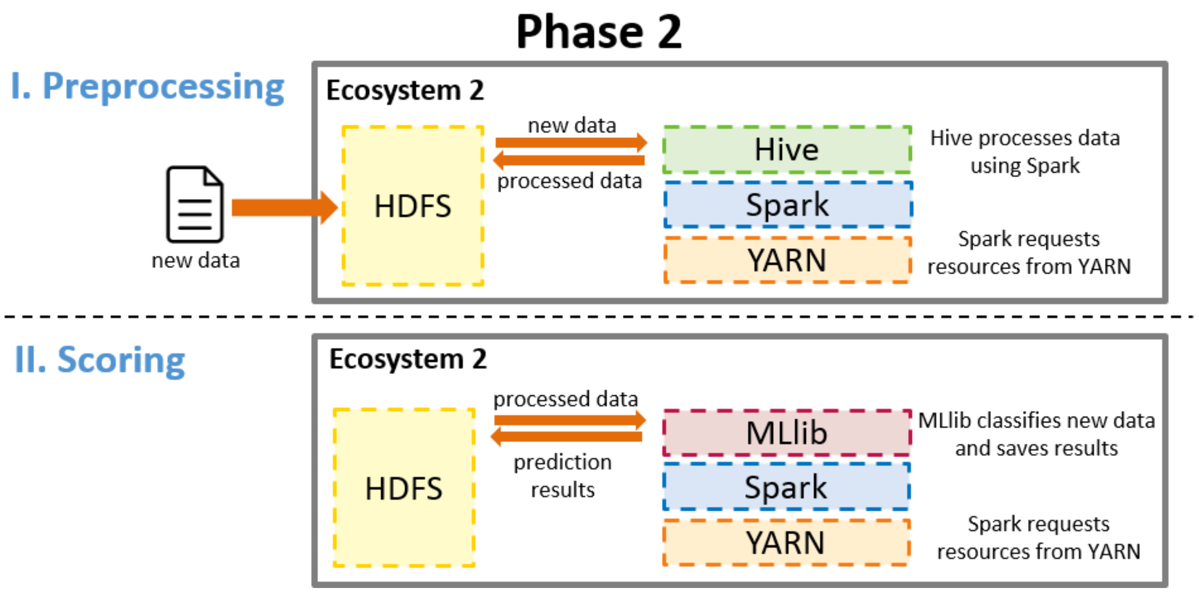


Figure 70: Implementation of Phase 2 of the PMB on Ecosystem 2.

Similar to the PMB implementation in Ecosystem 1, the execution time of each step of Phase 1 and Phase 2 is measured. Where possible, the execution time is further divided into subcategories such as time for initializing a framework, loading data, and processing it. The results of the PMB implementation are shown below.

Ecosystem 2: PMB Results

The performance of Ecosystem 2 is measured by timing the execution time of the individual steps of Phase 1 and Phase 2. Figure 71 shows the PMB performance results of Ecosystem 2 in seconds. As before, the results are divided into Phase 1 (blue) and Phase 2 (green). Data, data size, and request size are identical to the PMB implementation in Ecosystem 1.

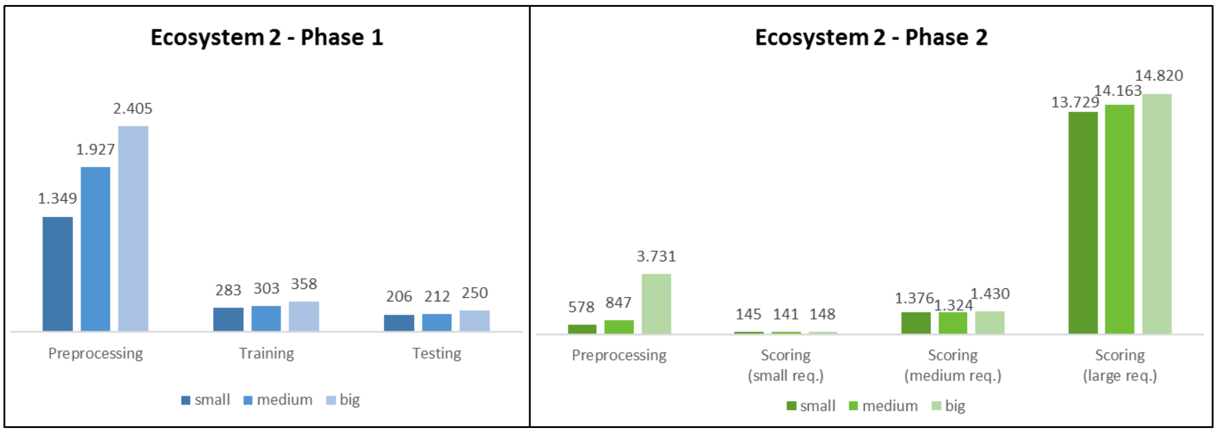


Figure 71: PMB results of Ecosystem 2 in seconds.

As before, most of the time during Phase 1 is used for preprocessing the data (Figure 71, blue bar graph, left). The processing of the medium data size takes 578 additional seconds, while the processing of the big data set only takes 478 additional seconds. Preprocessing therefore scales well with increasing data size. For training (blue bar graph, centre) and testing (blue bar graph, left) an increase in data size results in a decrease in processing time per machine. For example, training with the small data set takes 283 seconds, amounting to 8,6 seconds / machine. Adding 33 machines (medium data set) results in 4,6 seconds / machine and adding another 33 machines leads to 3,6 seconds / machine.

Preprocessing during Phase 2 (Figure 71, green bar graph, left) shows good scalability when increasing the data size. The processing time/item decreases from 57,8 seconds (small) to 8,47 seconds (medium) to 3,7 seconds (big). The data size has only minimal effect on the execution time during scoring. For instance, when scoring the small request (green bar graph, centre left), 10 items (145 seconds) are scored almost as fast as 1.000 items (148 seconds). The execution time scales linear with the request size. Increasing the request size 10 times results in approximately 10 times longer execution time. For instance, scoring the small data size increases from 145 seconds (small request), to 1.378 seconds (medium request), to 13.729 seconds (big request).

Figure 72 shows a breakdown of Phase 1(blue) and Phase 2(green) discussed above.

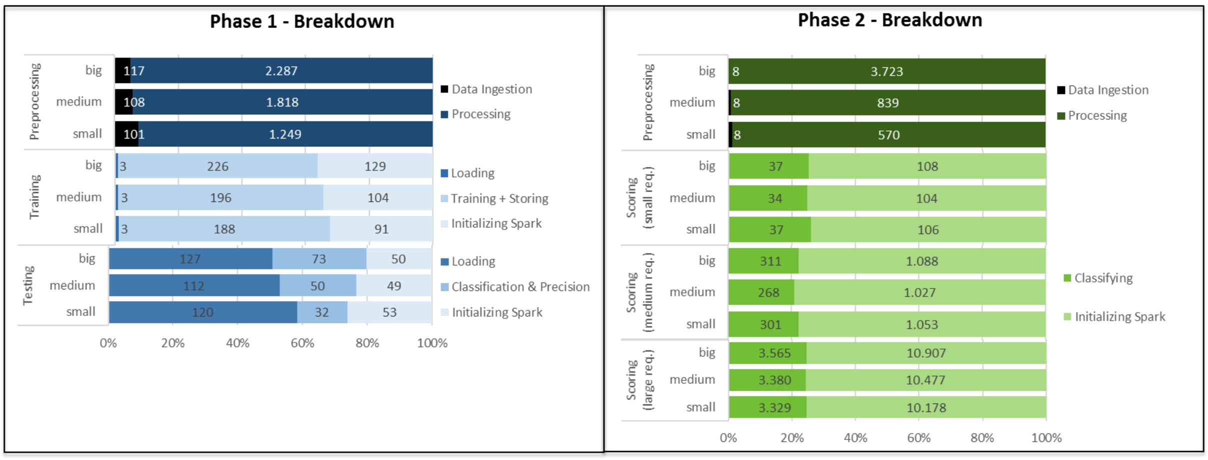


Figure 72: Ecosystem 2: Performance breakdown of Phase 1 and Phase 2 in seconds.

In Phase 1, preprocessing (Figure 72, blue bar graph, top) is separated into the two tasks of data ingestion and processing with processing being the major contributor to execution time. The breakdown of training (blue bar graph, centre) shows that loading data is not affected by file size. While counterintuitive at first, this can be explained by Sparks lazy loading approach. Data is only really loaded if used. During loading, Spark only checks if the data is available for later use but does not keep it in memory. This means, in Phase 1, data is first loaded during actual training of the model and not before. Training, and initializing Spark however, are affected by data size with increased execution times. During testing (blue bar graph, bottom) a similar phenomenon to training can be observed. Data size has almost no effect on data loading time but increases the time of classification. The time for initializing Spark is similar across all data sizes with 49-53 seconds.

Preprocessing of Phase 2 (Figure 72, green bar graph, top) is dominated by processing the data (>98,5%) with only 8 seconds spent for data ingestion. Across all three request sizes of scoring (green bar graph, bottom three), the execution time ratio between classifying data and initializing Spark remains similar at roughly 25%/75%.

6.5 PMB Analysis and Findings

This section presents the evaluation of the PMB. First, the performance results of the two PMB implementations are analysed and used to compare the two ecosystems with each other. The results of the comparison are then matched against conclusions drawn by other benchmarks, analysing Hadoop and Spark based environments. Finally, key findings are presented and used to evaluate the PMB.

6.5.1 Comparison between Ecosystem 1 and Ecosystem 2

During Phase 1 of the PMB, Ecosystem 1 and Ecosystem 2 differ in two main aspects. Firstly, although both ecosystems run the same Hive preprocessing scripts, Ecosystem 1 runs Hive on MapReduce while Ecosystem 2 runs Hive on Spark. Secondly, Ecosystem 1 uses Mahout for training and testing the model whereas Ecosystem 2 uses MLlib. Figure 73 shows a comparison between the PMB performances of Ecosystem 1 and Ecosystem 2 during Phase 1 of the PMB. The measurements are divided into the three steps of Preprocessing (left), Training (centre), and Testing (right).

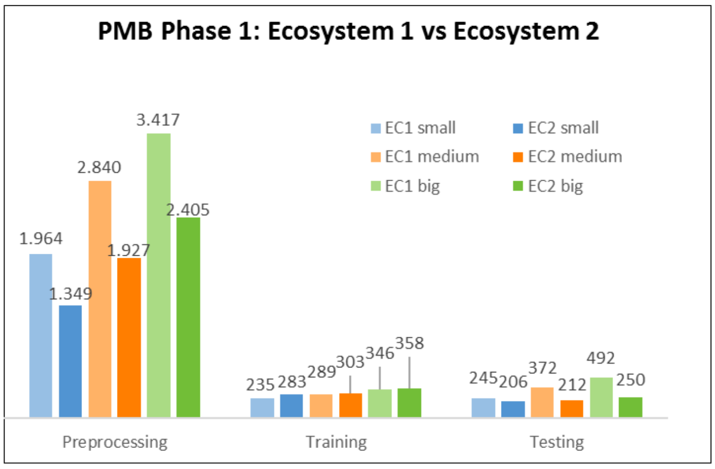


Figure 73: Comparison between Ecosystem 1 and Ecosystem 2 during Phase 1 of the PMB in seconds.

In comparison to Ecosystem 1, Ecosystem 2 shows significantly better performance during preprocessing (Figure 73, left) with a speed advantage of roughly 30%. Both preprocessing tasks are done with Hive but using the different execution engines Hadoop MapReduce and Spark. The better performance of Ecosystem 2 suggests that Hive on Spark is significantly faster than Hive on MapReduce.

For training (Figure 73, centre) with a small data set, Ecosystem 1 is more than 20% faster than Ecosystem 2. However, this advantage decreases with growing data sizes to 4,8% with the medium data set and 3,5% with the big data set. This shows that while Hadoop and Mahout are better with smaller data sets, Spark and MLlib are catching up with increasing data size.

Considering testing (Figure 73, right), Ecosystem 2 shows better performance across all three data sets. Spark and MLlib outperform Hadoop and Mahout especially with growing data sizes and are almost twice as fast when testing the large data set.

In Phase 2, Ecosystem 1 and Ecosystem 2 again use different execution engines for the same Hive preprocessing scripts (MapReduce om Ecosystem 1, Spark in Ecosystem 2). While Ecosystem 1 uses Mahout on MapReduce for scoring, Ecosystem 2 utilizes MLlib on Spark. Figure 74 displays the performance of Ecosystem 1 and Ecosystem 2 during Phase 2 of the PMB.

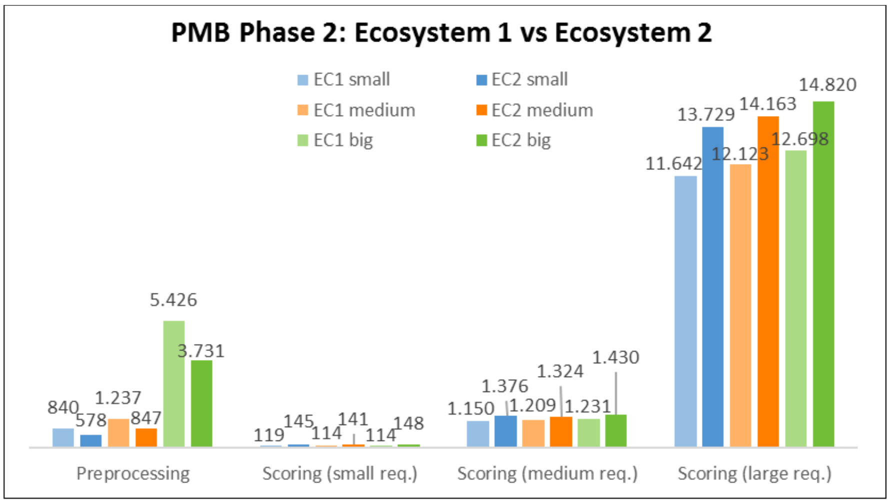


Figure 74: Comparison between Ecosystem 1 and Ecosystem 2 during Phase 2 of the PMB in seconds.

Similar to the observations during Phase 1, Ecosystem 2 shows roughly 30% better performance during Preprocessing across all data sizes in Phase 2. This reinforces the proposition that hive runs faster on the Spark execution engine than on MapReduce.

Throughout scoring, Ecosystem 1 outperforms Ecosystem 2. The main reason is the long initialization time of Spark. Starting a Spark job takes almost twice as long as starting a Hadoop MapReduce job. Due to repeated initialization with each request, Spark shows lower performance across all request sizes.

6.5.2 Validating PMB Results by Analysing other Performance Evaluation Studies

This section looks at the results from performance evaluation studies of Hadoop and Spark and compares them with the results of the PMB implementations. The analysis this comparison is used to assess the validity of the PMB.

Zaharia et al. [24] compare Spark to Hadoop by implementing two machine learning applications, logistic regression and k-means, and measuring their execution time. The experiment is done with 100GB of data on a 100 node cluster. For both applications, Spark shows significantly better performance: Spark outperforms Hadoop by up to 3x during k-means application and up to 20x in logistic regression. The PMB also shows a performance advantage of Spark, although not as high as reported by Zaharia et al. [24]. One possible explanation is the difference in measured tasks. While the PMB covers the entire process of data ingestion, data processing and data storage, Zaharia et al. [24] focus on data processing. Another possible explanation is the bigger cluster used by Zaharia et al. [24]. The significantly higher available RAM benefits Spark because it can store the entire data set in-memory at all time.

Samadi et al. [198] use the HiBench benchmark suite to analyze the performance of Spark and Hadoop. HiBench covers workloads from the four categories of micro benchmarks, web search, SQL, and machine learning. Their results show that Spark is up to 18x faster in the category of web search tasks, up to 6.7x faster in SQL tasks, 1.8x faster in micro-benchmarks, and 1.6x faster in machine learning tasks. Overall, the findings of Samadi et al. [198] correspond with the results of the PMB, where Spark also outperforms Hadoop.

Mavridis and Karatza [199] analyze the performance of Hadoop and Spark when analyzing web server log files. The analysis consists of tasks such as counting requests per day, finding possible DoS (Denial of Service) attacks, identifying DoS attackers, counting errors, and finding most frequent errors. All tasks are done on log files of three different sizes (1.1GB, 5.5GB, and 11GB) The experiment runs on a 6 node cluster with 8 GB RAM, and 40GB disk space each. Similar to the results of the PMB, the comparison of Mavridis and Karatza [199] shows that Spark is faster than Hadoop in every analyzed case.

Poggi et al. [200] compare the performance of Hive, Mahout, and MLlib using BigBench [14]. Their experiments show that Hive and MLlib is up to 2.2x faster than Hive and Mahout. These results are similar to the ones produced by the PMB, although in the latter, the performance advantage of MLlib is not as significant. This can be explained by the difference in tasks and data size. Due to the smaller data size used in PMB, initialization has a higher impact on overall execution time. Therefore, the higher initialization time of MLlib compromises its advantage during data processing, resulting in a smaller performance advantage then measured by Poggi et al. [200].

Overall, the PMB implementations on Ecosystem 1 and Ecosystem 2 produce comparable results to performance evaluation studies analysing Hadoop and Spark.

6.5.3 Key Findings and Evaluation of the PMB

The development and implementation of the PMB resulted in key findings, which are presented below:

Representability of the PMB

The PMB is developed based on in-depth research in the field of predictive maintenance (see chapter 6). The PMB data model (see section 5.2) represents typical data sources and data types encountered during predictive maintenance. It combines periodic condition monitoring data with event based maintenance-, error-, and failure logs. The data set was created by monitoring machines over the period of one year and thus represents real world data. The workload of the PMB (see section 5.3) is based on typical tasks of PMB, covering the three steps of predictive maintenance data acquisition, data processing, and maintenance decision making (see section 4.2-4.4).

Feasibility of the PMB

The PMB was implemented on the two most popular Big Data analytic ecosystems. The structure of these ecosystems is based on the 6-Pillar of Kahilfa et al. [18] (see section 3.3). The selection of the frameworks underlying the two ecosystems is based on a scientific literature research (see section 6.1). The successful implementation and execution of the PMB on two Big Data analytic ecosystem showed its feasibility. The metric of the PMB enables a structured comparison of the performance of both ecosystems.

Validity of the PMB

The PMB implementations on both ecosystems show similar results as performance evaluation studies comparing Hadoop and Spark based systems (see section 6.5.2). The PMB results indicate Spark outperforming Hadoop throughout almost every workload. This indication corresponds to the results of multiple studies analysing Hadoop and Spark.

7 Conclusion and Future Work

7.1 Conclusion

Benchmarks for Big Data analytic ecosystems need be designed for specific application areas. Most available Big Data benchmarks focus on tasks originating from e-commerce, retail, search engines, or social media. There are currently no benchmarks available for the field of predictive maintenance. This thesis introduced the PMB, a technology-agnostic benchmark to test Big Data analytic ecosystems in the application area of predictive maintenance.

The first step was to identify the requirements of a Big Data analytic ecosystem hosting a predictive maintenance system. For this purpose, extensive research was undertaken to establish a theoretical understanding of Big Data analytic ecosystems and their application in the field of predictive maintenance. Based on this research, general requirements for a benchmark covering predictive maintenance were identified. In the second step, the acquired knowledge was used to develop the PMB. After planning the benchmark, the data model and the workloads of the PMB were defined. For evaluation, the PMB was implemented using two different Big Data ecosystems and executed on a 5 node Raspberry Pi cluster. The frameworks forming the respective ecosystems were determined through a scientific literature research. Ecosystem 1 is based on Hadoop, Mahout, and Hive. Ecosystem 2 consists of Hadoop, Spark, MLlib, and Hive. The PMB results indicate that Ecosystem 2 outperforms Ecosystem 1 in almost every workload. Furthermore, the performance of Ecosystem 2 scales faster with growing data sizes than the performance of Ecosystem 1. The main driver behind this performance advantage is assumed to be the in-memory technology of the Spark execution engine.

The evaluation of the PMB showed that it is a viable tool for comparing Big Data analytic ecosystems in the field of predictive maintenance. The data model and workloads represent typical predictive maintenance workflows and the individual measurements provide a detailed view of the performance of the ecosystem. The results of the PMB implementations are comparable to results of other Big Data benchmark implementations, that analyse Hadoop and Spark based ecosystems.

7.2 Future Work

The PMB enables comparison of Big Data analytic ecosystems in the application area of predictive maintenance. Although this benchmark covers a typical data model and typical workloads faced during predictive maintenance, there are still open issues. These issues are presented below.

The PMB data model consists of periodic measurements (telemetry data), event based data (i.e. failure logs) and static information (machine meta data). All considered data types are value based. As described in chapter 4.2, many condition indicators such as vibration or acoustic data is measured in waveform. Therefore, incorporating wave data and analysing it presents a promising extension to the PMB.

PMB uses a decision tree as classification algorithm to predict future failures. As shown in chapter 4.4.2, multiple other machine learning algorithms are available for predicting failures. Hence, overall coverage of the PMB can be increased by including multiple machine learning algorithms for classification as well as prediction of remaining useful life.

For evaluation, the PMB was implemented on two different Big Data analytic ecosystems using a 5 node Raspberry Pi cluster. The testing environment is restricted by the limited capabilities of the Raspberry Pis. The next step is to execute the PMB implementations on an improved cluster of more capable machines, enabling insights in how both ecosystems scale with improved hardware.

Finally, the PMB can be implemented on other ecosystems. Analysis of these implementations can provide feedback on which areas of the PMB need further development. This iterative procedure improves the benchmark with each implementation.

Bibliography

[1] M. Macchi, I. Roda, and L. Fumagalli. On the Advancement of Maintenance Management: Towards Smart Maintenance in Manufacturing. In H. Lödding, R. Riedel, K. . Thoben, G. Cieminski, and D. Kiritsis, editors, *Advances in Production Management Systems. The Path to Intelligent, Collaborative and Sustainable Manufacturing. APMS 2017*, volume 513 of *IFIP Advances in Information and Communication Technology*, pages 383–390. Springer Cham., 2017.

[2] L. M. Pintelon and L. F. Gelders. Maintenance management decision making, *European Journal of Operational Research*, 58(3): 301–317, 1992.

[3] R. K. Mobley. *An Introduction to Predictive Maintenance*. Elsevier B.V.: Amsterdam, 2nd edition, 2002.

[4] R. Roy, R. Stark, K. Tracht, S. Takata, and M. Mori. Continuous maintenance and the future - Foundations and technological challenges, *CIRP Annals - Manufacturing Technology*, 65(2): 667–688, 2016.

[5] D. Laney. *3D Data Management: Controlling Data Volume, Velocity, and Variety*, META Group, 2001.

[6] I. Anagnostopoulos, S. Zeadally, and E. Exposito. Handling big data: research challenges and future directions, *Journal of Supercomputing*, 72(4): 1494–1516, 2016.

[7] E. Rahm and H. H. Do. Data Cleaning: Problems and Current Approaches, *Bulletin of the Technical Committee on Data Engineering*, 23(4): 3–13, 2000.

[8] I. Yaqoob, I. A. T. Hashem, A. Gani, S. Mokhtar, E. Ahmed, N. B. Anuar, and A. V. Vasilakos. Big data: From beginning to future, *International Journal of Information Management*, 36(6): 1231–1247, 2016.

[9] P. Pääkkönen and D. Pakkala. Reference Architecture and Classification of Technologies, Products and Services for Big Data Systems, *Big Data Research*, 2(4): 166–186, 2015.

[10] M. Turck and J. Hao. Firing on All Cylinders: The 2017 Big Data Landscape, 2017. http://mattturck.com/bigdata2017/, Accessed: 10.05.2017.

[11] T. Ivanov, T. Rabl, M. Poess, A. Queralt, J. Poelman, N. Poggi, and J. Buell. Big data benchmark compendium. In R. Nambiar and M. Poess, editors, *Performance Evaluation and Benchmarking: Traditional to Big Data to Internet of Things. TPCTC 2015*, volume 9508 of *Lecture Notes in Computer Science*, pages 135–155. Springer Cham., 2016.

[12] R. Han, L. Xiaoyi, and X. Jiangtao. On big data benchmarking. In J. Zhan, R. Han, and C. Weng, editors, *Big Data Benchmarks, Performance Optimization, and Emerging Hardware. BPOE 2014*, volume 8807 of *Lecture Notes in Computer Science*, pages 3–18. Springer Cham., 2014.

[13] C. Baru, M. Bhandarkar, R. Nambiar, M. Poess, and T. Rabl. Setting the direction for big data benchmark standards. In R. Nambiar and M. Poess, editors, *Selected Topics in Performance Evaluation and Benchmarking. TPCTC 2012*, volume 7755 of *Lecture Notes in Computer Science*, pages 197–208. Springer Berlin Heidelberg., 2013.

[14] A. Ghazal, T. Rabl, M. Hu, F. Raab, M. Poess, A. Crolotte, and H.-A. Jacobsen. BigBench: Towards an Industry Standard Benchmark for Big Data Analytics, In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD '13)*, pages 1197–1208. ACM New York, 2013.

[15] L. Wang, J. Zhan, C. Luo, Y. Zhu, Q. Yang, Y. He, W. Gao, Z. Jia, Y. Shi, S. Zhang, C. Zheng, G. Lu, K. Zhan, X. Li, and B. Qiu. BigDataBench: A big data benchmark suite from internet services, In *Proceedings of the 20th International Symposium on High Performance Computer Architecture (HPCA)*, pages 488–499. IEEE, Orlando, 2014.

[16] B. Kitchenham, O. P. Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman. Systematic literature reviews in software engineering – A systematic literature review, *Information and Software Technology*, 51(1): 7–15, 2009.

[17] A. R. Hevner, S. T. March, J. Park, and S. Ram. Design Science in Information Systems Research, *MIS Quarterly*, 28(1): 75–105, 2004.

[18] S. Khalifa, Y. Elshater, K. Sundaravarathan, A. Bhat, P. Martin, F. Imam, D. Rope, M. Mcroberts, and C. Statchuk. The Six Pillars for Building Big Data Analytics Ecosystems, *ACM Computing Surveys*, 49(2): 1–36, 2016.

[19] S. Huang, J. Huang, Y. Liu, L. Yi, and J. Dai. HiBench: A Representative and Comprehensive Hadoop Benchmark Suite, 2012.

[20] A. Sangroya, D. Serrano, and S. Bouchenak. *MRBS: A Comprehensive MapReduce Benchmark Suite, Research Report RR-LIG-024*, LIG Grenoble France, 2012.

[21] M. Li, J. Tan, Y. Wang, L. Zhang, and V. Salapura. SparkBench, In *Proceedings of the 12th ACM International Conference on Computing Frontiers - CF '15*, pages 1–8. ACM New York, 2015.

[22] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica. Discretized Streams: Fault-Tolerant

Streaming Computation at Scale, In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 423–438. ACM New York, 2013.

[23] J. Dean and S. Ghemawat. MapReduce: Simplied Data Processing on Large Clusters, In *Proceedings of 6th Symposium on Operating Systems Design and Implementation*, pages 137–149. USENIX Association Berkeley, 2004.

[24] M. Zaharia, C. Mosharaf, D. Tathagata, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoicam. Resilient Distributed Datasets- A Fault-Tolerant Abstraction for In-Memory Cluster Computing, In *Proceedings of the Ninth USENIX NSDI Symposium on Networked Systems Design and Implementation*, pages 2–14. USENIX Association Berkeley, 2012.

[25] S. Shahrivari. Beyond Batch Processing: Towards Real-Time and Streaming Big Data, *Computers*, 3(4): 117–129, 2014.

[26] A. Shukla and Y. Simmhan. Benchmarking distributed stream processing platforms for IoT applications. In R. Nambiar and M. Poess, editors, *Performance Evaluation and Benchmarking. Traditional - Big Data - Internet of Things. TPCTC 2016*, volume 10080 of *Lecture Notes in Computer Science*, pages 90–106. Springer Cham., 2017.

[27] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker. A comparison of approaches to large-scale data analysis, In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 165–178. ACM New York, 2009.

[28] J. Ferrarons, M. Adhana, C. Colmenares, S. Pietrowska, F. Bentayeb, and J. Darmont. PRIMEBALL: A Parallel Processing Framework Benchmark for Big Data Applications in the Cloud. In R. Nambiar and M. Poess, editors, *Performance Characterization and Benchmarking. TPCTC 2013*, volume 8391 of *Lecture Notes in Computer Science*, pages 109–124. Springer Cham., 2014.

[29] P. Cao, B. Gowda, S. Lakshmi, C. Narasimhadevara, P. Nguyen, J. Poelman, M. Poess, and T. Rabl. From BigBench to TPCx-BB: Standardization of a Big Data Benchmark. In R. Nambiar and M. Poess, editors, *Performance Evaluation and Benchmarking. Traditional - Big Data - Internet of Things. TPCTC 2016*, volume 10080 of *Lecture Notes in Computer Science*, pages 24–44. Springer Cham., 2017.

[30] Transaction Processing Performance Council (TPC). *Tpc benchmark H: Standard Specification*, San Francisco, 2011. http://www.tpc.org/TPC_Documents_Current_Versions/pdf/tpc-h_v2.17.3.pdf, Accessed: 10.04.2018.

[31] Standard Performance Evaluation Corporation (SPEC). *SFS 2014 SP2 Users Guide*, 2017. https://www.spec.org/sfs2014/docs/usersguide.pdf, Accessed: 10.04.2018.

[32] Storage Performance Council (SPC). *SPC-1 Specification*, 2017. http://spcresults.org/sites/default/files/files/specifications/SPC1_v340_final.pdf, Accessed: 10.04.2018.

[33] Storage Performance Council (SPC). *SPC-2 Specification*, 2017. http://spcresults.org/sites/default/files/files/specifications/SPC-2_SPC-2E_v1.6.pdf, Accessed: 10.04.2018.

[34] G. Press. A Very Short History Of Big Data, *Forbes*, 2013. http://www.forbes.com/sites/gilpress/2013/05/09/a-very-short-history-of-big-data/, Accessed: 08.04.2018.

[35] N. Khan, I. Yaqoob, I. A. T. Hashem, Z. Inayat, W. K. Mahmoud Ali, M. Alam, M. Shiraz, and A. Gani. Big Data: Survey, Technologies, Opportunities, and Challenges, *The Scientific World Journal*, 2014: 1–18, 2014.

[36] N. Henke, J. Bughin, M. Chui, J. Manyika, T. Saleh, B. Wiseman, and G. Sethupathy. *The Age of Analytics : Competing in a Data-Driven World*, McKinsey Global Institute, 2016.

[37] Y. Demchenko, C. De Laat, and P. Membrey. Defining architecture components of the Big Data Ecosystem, In *Poceedings of the 2014 International Conference on Collaboration Technologies and Systems (CTS)*, pages 104–112. IEEE, 2014.

[38] A. Gandomi and M. Haider. Beyond the hype: Big data concepts, methods, and analytics, *International Journal of Information Management*, 35(2): 137–144, 2015.

[39] Google. Google Trends, 2018. https://trends.google.de/, Accessed: 25.01.2018.

[40] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. H. Byers. *Big data: The next frontier for innovation, competition, and productivity*, McKinsey Global Institute, 2011.

[41] I. Lee. Big data: Dimensions, evolution, impacts, and challenges, *Business Horizons*, 60(3): 293–303, 2017.

[42] J. Gantz and D. Reinsel. State of the Universe: An Executive Summary, *IDC iView*, 2011(June): 1–12, 2011.

[43] T. Shafer. The 42 V's of Big Data and Data Science, 2017. https://www.elderresearch.com/company/blog/42-v-of-big-data, Accessed: 15.02.2018.

[44] H. Chen, R. H. L. Chiang, and V. C. Storey. Business Intelligence and Analytics: From Big Data to Big Impact, *MIS Quaterly*, 36(4): 1165–1188, 2013.

[45] S. Kaisler, F. Armour, J. A. Espinosa, and W. Money. Big Data: Issues and Challenges Moving Forward, In *Proceedings of the 46th Hawaii International Conference on System Sciences*, pages 995–1004. IEEE, 2013.

[46] S. Achari. *Hadoop Essentials*. Packt Publishing Ltd.: Birmingham, 2015.

[47] H. Hussain, S. U. R. Malik, A. Hameed, S. U. Khan, G. Bickler, N. Min-Allah, M. B. Qureshi, L. Zhang, W. Yongji, N. Ghani, J. Kolodziej, A. Y. Zomaya, C. Z. Xu, P. Balaji, A. Vishnu, F. Pinel, J. E. Pecero, D. Kliazovich, P. Bouvry, H. Li, L. Wang, D. Chen, and A. Rayes. A survey on resource allocation in high performance distributed computing systems, *Parallel Computing*, 39(11): 709–736, 2013.

[48] R. Appuswamy, C. Gkantsidis, D. Narayanan, O. Hodson, and A. Rowstron. Scale-up vs scale-out for Hadoop, In *Proceedings of the 4th annual Symposium on Cloud Computing - SOCC '13*, pages 1–13. ACM New York, 2013.

[49] H. Zhang, G. Chen, B. C. Ooi, K. L. Tan, and M. Zhang. In-Memory Big Data Management and Processing: A Survey, *IEEE Transactions on Knowledge and Data Engineering*, 27(7): 1920–1948, 2015.

[50] R. Bärenfänger, B. Otto, and H. Österle. Business value of in-memory technology-Multiple-case study insights, *Industrial Management and Data Systems*, 114(9): 1396–1414, 2014.

[51] J. vom Brocke, S. Debortoli, O. Müller, and N. Reuter. How In-memory Technology Can Create Business Value: Insights from the Hilti Case, *Communications of the Association for Information Systems*, 34(1): 151–168, 2014.

[52] J. Bosch. From Software Product Lines to Software Ecosystems, In *Proceedings of the 13th International Software Product Line Conference (SPLC 09)*, pages 111–119. Carnegie Mellon University Pittsburgh, 2009.

[53] J. Chen, Y. Chen, X. Du, C. Li, J. Lu, S. Zhao, and X. Zhou. Big data challenge: A data management perspective, *Frontiers of Computer Science*, 7(2): 157–164, 2013.

[54] H. Fang, Z. Zhang, C. J. Wang, M. Daneshmand, C. Wang, and H. Wang. A survey of big data research, *IEEE Network*, 29(5): 6–9, 2015.

[55] E. F. Codd. A relational model of Data for Large Shared Data Banks, *Communications of the ACM*, 13(6): 377–387, 1970.

[56] T. Haerder and A. Reuter. Principles of transaction-oriented database recovery, *ACM Computing Surveys*, 15(4): 287–317, 1983.

[57] DB-Engines. DB-Engines Ranking, 2017. https://db-engines.com/en/ranking, Accessed: 13.11.2017.

[58] Progress Software Corporation. *2017 Data Connectivity Outlook*, 2017. https://www.progress.com/campaigns/datadirect/whitepapers/2017-data-connectivity-outlook-survey, Accessed: 02.12.2017.

[59] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google file system, *ACM SIGOPS Operating Systems Review*, 37(5): 29, 2003.

[60] K. Shvachko. The Hadoop Distributed File System, In *Proceedings of the 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10. IEEE, 2010.

[61] E. Brewer. CAP twelve years later: How the "rules" have changed, *Computer*, 45(2): 23–29, 2012.

[62] Y. Huang and T. J. Luo. NoSQL database: A scalable, availability, high performance storage for big data. In Q. Zu, M. Vargas-Vera, and M. Hu, editors, *Pervasive Computing and the Networked World. ICPCA/SWS 2013*, volume 8351 of *Lecture Notes in Computer Science*, pages 172–183. Springer Cham., 2014.

[63] L. Liu. Computing infrastructure for big data processing, *Frontiers of Computer Science*, 7(2): 165–170, 2013.

[64] W. Wingerath, F. Gessert, S. Friedrich, and N. Ritter. Real-time stream processing for Big Data, *it - Information Technology*, 58(4): 186–194, 2016.

[65] Y. Chen, S. Alspaugh, and R. Katz. Interactive analytical processing in big data systems: a cross-industry study of MapReduce workloads, *Proceedings of the VLDB Endowment*, 5(12): 1802–1813, 2012.

[66] R. Singh and P. J. Kaur. Analyzing performance of Apache Tez and MapReduce with hadoop multinode cluster on Amazon cloud, *Journal of Big Data*, 3(1): 19, 2016.

[67] M. Armbrust, A. Ghodsi, M. Zaharia, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, and M. J. Franklin. Spark SQL: Relational Data Processing in Spark, In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1383–1394. ACM New York, 2015.

[68] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis. Dremel: Interactive Analysis of Web-Scale Datasets, In *Poceedings of the 36th International Conference on*

Very Large Data Bases, pages 330–339. Google, 2010.

[69] S. Agarwal, A. Panda, B. Mozafari, S. Madden, and I. Stoica. BlinkDB: Queries with Bounded Errors and Bounded Response Times on Very Large Data, In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 29–42. ACM New York, 2012.

[70] N. Laptev, K. Zeng, and C. Zaniolo. Early accurate results for advanced analytics on MapReduce, *Proceedings of the VLDB Endowment*, 5(10): 1028–1039, 2012.

[71] I. Goiri, R. Bianchini, S. Nagarakatte, and T. D. Nguyen. ApproxHadoop: Bringing Approximations to MapReduce Frameworks, In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 383–397. ACM New York, 2015.

[72] R. Nair. Big data needs approximate computing: technical perspective, *Communications of the ACM*, 58(1): 105–115, 2015.

[73] V. Kumar Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O 'malley, S. Radia, B. Reed, and E. Baldeschwieler. Apache Hadoop YARN: Yet Another Resource Negotiator, In *SOCC '13 Proceedings of the 4th annual Symposium on Cloud Computing*, page article 5. ACM New York, 2013.

[74] D. Fisher, R. DeLine, M. Czerwinski, and S. Drucker. Interactions with big data analytics, *Interactions*, 19: 50–59, 2012.

[75] G. Piatetsky. KDnuggets 2016 Software Poll Results. http://www.kdnuggets.com/2016/06/r-python-top-analytics-data-mining-data-science-software.html, Accessed: 10.03.2017.

[76] R. S. Barga, J. Ekanayake, and W. Lu. Project Daytona: Data analytics as a cloud service, In *Proceedings of the 28th International Conference on Data Engineerin*, pages 1317–1320. IEEE, 2012.

[77] Z. P. Ak, G. Makrai, T. Henk, and C. Gar-Papanek. Radoop : Analyzing Big Data with RapidMiner and Hadoop, In *Proceedings of the 2nd RapidMiner Community Meeting and Conference*, pages 1–12. , 2011.

[78] Y. Gil, V. Ratnakar, J. Kim, P. Gonzalez-Calero, P. Groth, J. Moody, and E. Deelman. Wings: Intelligent Workflow-Based Design of Computational Experiments, *IEEE Intelligent Systems*, 26(1): 62–72, 2011.

[79] P. Godfrey, J. Gryz, and P. Lasek. Interactive Visualization of Large Data Sets, *IEEE Transactions on Knowledge and Data Engineering*, 28(8): 2142–2157, 2015.

[80] A. S. Syed Fiaz, N. Asha, D. Sumathi, and A. S. Syed Navaz. Visualization: Enhancing big data more adaptable and valuable, *International Journal of Applied Engineering Research*, 11(4): 2801–2804, 2016.

[81] S. M. Ali, N. Gupta, G. K. Nayak, and R. K. Lenka. Big data visualization: Tools and challenges, In *Proceedings of the 2nd International Conference on Contemporary Computing and Informatics (IC3I)*, pages 656–660. IEEE, 2016.

[82] S. A. Hirve, A. Kunjir, B. Shaikh, and K. Shah. An approach towards data visualization based on AR principles, In *Proceedings of the 2017 International Conference on Big Data Analytics and Computational Intelligence (ICBDAC)*, pages 128–133. IEEE, 2017.

[83] C. Bermejo, Z. Huang, T. Braud, and P. Hui. When Augmented Reality meets Big Data, In *Proceedings of the 37th International Conference on Distributed Computing Systems Workshops, ICDCSW 2017*, pages 169–174. IEEE, 2017.

[84] E. Olshannikova, A. Ometov, Y. Koucheryavy, and T. Olsson. Visualizing Big Data with augmented and virtual reality: challenges and research agenda, *Journal of Big Data*, 2(1): 22, 2015.

[85] C. Donalek, S. G. Djorgovski, A. Cioc, A. Wang, J. Zhang, E. Lawler, S. Yeh, A. Mahabal, M. Graham, A. Drake, S. Davidoff, J. S. Norris, and G. Longo. Immersive and Collaborative Data Visualization Using Virtual Reality Platforms, In *Proceedings of 2014 IEEE International Conference on Big Data*, pages 609–614. IEEE, 2014.

[86] D. D. Chamberlin and R. F. Boyce. SEQUEL: A Structured English Query Language, In *Proceedings of the 1976 ACM SIGFIDET (now SIGMOD) workshop on Data description, access and control - FIDET '76*, pages 249–264. ACM New York, New York, New York, USA, 1976.

[87] G. Piatetsky. Python overtakes R, becomes the leader in Data Science, Machine Learning platforms, 2017. https://www.kdnuggets.com/2017/08/python-overtakes-r-leader-analytics-data-science.html, Accessed: 15.11.2017.

[88] B. Oancea and R. M. Dragoescu. Integrating R and Hadoop for Big Data Analysis, *Romanian Statistical Review*, (2): 83–94, 2014.

[89] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: A Not-So-Foreign Language for Data Processing, In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data - SIGMOD '08*, pages 1099–1110. ACM New York, 2008.

[90] K. . Beyer, V. Ercegovac, R. Gemulla, A. Balmin, M. Eltabakh, C.-C. Kanne, F. Ozcan, and E. . Shekita. Jaql: A scripting language for large scale semistructured data analysis, *Proceedings of the VLDB*

Endowment, 4(12): 1272–1283, 2011.

[91] X. Meng, J. Bradley, S. Street, S. Francisco, E. Sparks, U. C. Berkeley, S. Hall, S. Street, S. Francisco, D. Xin, R. Xin, M. J. Franklin, U. C. Berkeley, and S. Hall. MLlib : Machine Learning in Apache Spark, *The Journal of Machine Learning Research*, 17(1): 1235–1241, 2016.

[92] T. Kraska, A. Talwalkar, J. Duchi, R. Griffith, M. Franklin, and M. Jordan. MLbase : A Distributed Machine-learning System, In *Proceedings of the 6th Biennial Conference on Innovative Data Systems Research (CIDR'13)*, 2013.

[93] S. Schelter, A. Palumbo, S. Quinn, and A. Musselman. Samsara : Declarative Machine Learning on Distributed Dataflow Systems, In *Proceedings of the 30th Conference on Neural Information Processing Systems (NIPS 2016)*, 2016.

[94] S. Landset, T. M. Khoshgoftaar, A. N. Richter, and T. Hasanin. A survey of open source tools for machine learning with big data in the Hadoop ecosystem, *Journal of Big Data*, 2(1): 24, 2015.

[95] F. Serban, J. Vanschoren, J.-U. Kietz, and A. Bernstein. A survey of intelligent assistants for data analysis, *ACM Computing Surveys*, 45(3): 1–35, 2013.

[96] A. Bernstein, F. Provost, and S. Hill. Toward intelligent assistance for a data mining process: An ontology-based approach for cost-sensitive classification, *IEEE Transactions on Knowledge and Data Engineering*, 17(4): 503–518, 2005.

[97] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From Data Mining to Knowledge Discovery in Databases, *AI Magazine*, 17(3): 37, 1996.

[98] K. Ebner, T. Bühnen, and N. Urbach. Think big with big data: Identifying suitable big data strategies in corporate environments, In *Proceedings of the 47th Hawaii International Conference on System Sciences*, pages 3748–3757. IEEE, 2014.

[99] M. Kavis. *Architecting the cloud: Design decisions for cloud computing service models (SaaS, PaaS, AND IaaS)*. John Wiley & Sons: Hoboken, 2014.

[100] B. Al-Najjar and I. Alsyouf. Enhancing a company's profitability and competitiveness using integrated vibration-based maintenance: A case study, *European Journal of Operational Research*, 157(3): 643–657, 2004.

[101] I. Alsyouf. The role of maintenance in improving companies' productivity and profitability, *International Journal of Production Economics*, 105(1): 70–78, 2007.

[102] R. A. Carter. Shovel maintenance gains from improved designs, tools, and techniques, *Engineering and Mining Journal*, 202(8): 7–10, 2001.

[103] A. Bousdekis, B. Magoutas, D. Apostolou, and G. Mentzas. A proactive decision making framework for condition-based maintenance, *Industrial Management & Data Systems*, 115(7): 1225–1250, 2015.

[104] A. K. S. Jardine, D. Lin, and D. Banjevic. A review on machinery diagnostics and prognostics implementing condition-based maintenance, *Mechanical Systems and Signal Processing*, 20(7): 1483–1510, 2006.

[105] R. Ahmad and S. Kamaruddin. An overview of time-based and condition-based maintenance in industrial application, *Computers and Industrial Engineering*, 63(1): 135–149, 2012.

[106] A. H. C. Tsang. Strategic dimensions of maintenance management, *Journal of Quality in Maintenance Engineering*, 8(1): 7–39, 2002.

[107] D. J. Edwards, G. D. Holt, and F.C.Harris. Predictive maintenance techniques and their relevance to construction plant, *Journal of Quality in Maintenance Engineering*, 4(1): 25, 1998.

[108] M. C. Carnera. Selection of diagnostic techniques and instrumentation in a predictive maintenance program. A case study, *Decision Support Systems*, 38(4): 539–555, 2005.

[109] G. E. Aitor. Sound-based predictive maintenance : a cost-effective approach, *Hydrocarbon Processing*, 87(5): 37–40, 2008.

[110] J. R. González, J. Velayos, and M. Comamala. Predictive Maintenance of Cogeneration Engines Through the Analysis of the Circulating Fluids, In *Proceedings of the 2002 International Joint Power Generation Conference*, pages 389–396. ASME, Phoenix, 2002.

[111] M. Lukas and D. P. Anderson. Lubricant Analysis for Gas Turbine Condition Monitoring, *Journal of Engineering for GAs Turbines and Power*, 119(October): 863–869, 1997.

[112] S. Kalligeros. Predictive Maintenance of Hydraulic Lifts through Lubricating Oil Analysis, *Machines*, 2(1): 1–12, 2013.

[113] M. M. Najafabadi, F. Villanustre, T. M. Khoshgoftaar, N. Seliya, R. Wald, and E. Muharemagic. Deep learning applications and challenges in big data analytics, *Journal of Big Data*, 2(1): 1, 2015.

[114] N. Tang. Big data cleaning. In L. Chen, Y. Jia, T. Sellis, and G. Liu, editors, *Web Technologies and Applications. APWeb 2014*, volume 8709 of *Lecture Notes in Computer Science*, pages 13–24. Springer Cham., 2014.

[115] R. Xu and C. Kwan. Robust Isolation Of Sensor Failures, *Asian Journal of Control*, 5(1): 12–23, 2008.

[116] F. Koushanfar, M. Potkonjak, and A. Sangiovanni-Vincentelli. On-line fault detection of sensor measurements, In *Proceedings of IEEE Sensors 2003 (IEEE Cat. No.03CH37498)*, pages 974–979. IEEE.
[117] M. H. Gorelick. Bias arising from missing data in predictive models, *Journal of Clinical Epidemiology*, 59(10): 1115–1123, 2006.
[118] A. Breitwieser and K. Wick. What we miss by missing data: Aid effectiveness revisited, *World Development*, 78(C): 554–571, 2016.
[119] P. Louridas and C. Ebert. Machine Learning, *IEEE Software*, 33(5): 110–115, 2016.
[120] A. Malhi and R. X. Gao. PCA-based feature selection scheme for machine defect classification, *IEEE Transactions on Instrumentation and Measurement*, 53(6): 1517–1525, 2004.
[121] A. K. Jain, R. P. W. Duin, and J. Mao. Statistical pattern recognition: a review, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1): 4–37, 2000.
[122] I. Lopez and N. Sarigul-Klijn. Effects of Dimensional Reduction Techniques on Structural Damage Assessment Under Uncertainty, *Journal of Vibration and Acoustics*, 133(6): 61008, 2011.
[123] T. Oikawa, M. Tomizawa, and S. Degawa. New monitoring system for thermal power plants using digital image processing and sound analysis, *Control Engineering Practice*, 5(1): 75–78, 1997.
[124] C. Demant, B. Streicher-Abel, and C. Garnica. *Industrial Image Processing. visual quality control in manufacturing*, 2013.
[125] C. Connolly. The use of infrared imaging in industry, *Assembly Automation*, 25(3): 191–195, 2005.
[126] A. Pandian and A. Ali. A review of recent trends in machine diagnosis and prognosis algorithms, In *Proceedings of the 2009 World Congress on Nature & Biologically Inspired Computing (NaBIC)*, pages 1731–1736. IEEE, 2009.
[127] M. Nyberg. A general framework for fault diagnosis based on statistical hypothesis testing, In *Proceedings of the International workshop on principles of diagnosis*, 2001.
[128] V. a. A. Skormin, L. J. J. Popyack, V. I. I. Gorodetski, M. L. L. Araiza, and J. D. D. Michel. Applications of cluster analysis in diagnostics-related problems, In *Proceedings of the 1999 IEEE Aerospace Conference (Cat. No.99TH8403)*, pages 161–168 vols.3-161–168 3. IEEE, 1999.
[129] J. Ying, T. Kirubarajan, and K. R. Pattipati. A hidden Markov model-based algorithm for fault diagnosis with partial and imperfect tests, *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, 30(4): 463–473, 2000.
[130] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm, *Journal of the Royal Statistical Society Series B Methodological*, 39(1): 1–38, 1977.
[131] A. H. Tai, W. K. Ching, and L. Y. Chan. Detection of machine failure: Hidden Markov Model approach, *Computers and Industrial Engineering*, 57(2): 608–619, 2009.
[132] Z. Li, Z. Wu, Y. He, and C. Fulei. Hidden Markov model-based fault diagnostics method in speed-up and speed-down process for rotating machinery, *Mechanical Systems and Signal Processing*, 19(2): 329–339, 2005.
[133] H. Wu, Z. Yu, and Y. Wang. Real-time FDM machine condition monitoring and diagnosis based on acoustic emission and hidden semi-Markov model, *International Journal of Advanced Manufacturing Technology*, 90(5–8): 2027–2036, 2017.
[134] C. (Chris) Aldrich and L. Auret. *Unsupervised process monitoring and fault diagnosis with machine learning methods*, 2013.
[135] W. S. Mcculloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity, *Bulletin of Mathematical Biophysics*, 5(4): 115–133, 1943.
[136] S. He and X. Li. Application of a group search optimization based Artificial Neural Network to machine condition monitoring, In *Proceedings of the 2008 IEEE International Conference on Emerging Technologies and Factory Automation*, pages 1260–1266. IEEE, 2008.
[137] N. K. Verma, R. K. Sevakula, S. Dixit, and A. Salour. Intelligent Condition Based Monitoring Using Acoustic Signals for Air Compressors, *IEEE Transactions on Reliability*, 65(1): 291–309, 2016.
[138] H. Soliman, H. Wang, B. Gadalla, and F. Blaabjer. Artificial Neural Network Algorithm for Condition Monitoring of DC-link Capacitors Based on Capacitance Estimation, *Journal of Renewable Energy and Sustainable Development (RESD)*, 1(2): 294–299, 2015.
[139] S. Gowid, R. Dixon, and S. Ghani. Performance Comparison Between Fast Fourier Transform-Based Segmentation, Feature Selection, and Fault Identification Algorithm and Neural Network for the Condition Monitoring of Centrifugal Equipment, *Journal of Dynamic Systems, Measurement, and Control*, 139(6): 61013, 2017.
[140] X. Deng, Q. Gao, C. Zhang, D. Hu, and T. Yang. Rule - based Fault Diagnosis Expert System for Wind Turbine, In *Proceedings of the ITM Web of Conferences 11*, page 7005. , 2017.
[141] L. Gao, L. Wu, Y. Wang, H. Wei, and H. Ye. Intelligent fault diagnostic system based on RBR for the gearbox of rolling mills, *Frontiers of Mechanical Engineering in China*, 5(4): 483–490, 2010.

[142] Z. Wen, J. Crossman, J. Cardillo, and Y. L. Murphey. Case Base Reasoning in Vehicle Fault Diagnostics, In *Proceedings of the International Joint Conference in Neural Networks*, pages 2679–2684. IEEE, 2003.

[143] M. Stanek, M. Morari, and K. Fröhlich. Model-aided diagnosis: An inexpensive combination of model-based and case-based condition assessment, *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, 31(2): 137–145, 2001.

[144] L. A. Zadeh. Fuzzy sets, *Information and Control*, 8(3): 338–353, 1965.

[145] C. K. Mechefske. Objective machinery fault diagnosis using fuzzy logic, *Mechanical Systems and Signal Processing*, 12(6): 855–862, 1998.

[146] T. Noreesuwan and B. Suksawat. Propose of Unsealed Deep Groove Ball Bearing Condition Monitoring Using Sound Analysis and Fuzzy Logic, In *Proceedings of the International Conference on Control, Automation and Systems*, pages 409–413. IEEE, 2010.

[147] M. Hichem, D. Djalel, and A. Salim. Monitoring of Stator windings Faults in Induction Machine Using Fuzzy Logic, *Journal of Electrical and Electronics Engineering*, 9(2): 49–54, 2016.

[148] Y. Peng, M. Dong, and M. J. Zuo. Current status of machine prognostics in condition-based maintenance: A review, *International Journal of Advanced Manufacturing Technology*, 50(1–4): 297–313, 2010.

[149] W. BARTELMUS. Mathematical Modelling and Computer Simulations As an Aid To Gearbox Diagnostics, *Mechanical Systems and Signal Processing*, 15(5): 855–871, 2001.

[150] K. B. Goode, J. Moore, and B. J. Roylance. Plant machinery working life prediction method utilizing reliability and condition-monitoring data, *Proceedings of the Institution of Mechanical Engineers*, 214(2): 109–122, 2000.

[151] Q. Li and S. Y. Liang. Degradation trend prognostics for rolling bearing using improved R/S statistic model and fractional Brownian motion approach, *IEEE Access*, 2017.

[152] J. C. R. Pacheco, D. T. Román, and L. E. Vargas. R/S Statistic: Accuracy and Implementations, In *Proceedings of 18th International Conference on Electronics, Communications and Computers (conielecomp 2008)*, pages 17–22. IEEE, 2008.

[153] X. S. Si, W. Wang, C. H. Hu, and D. H. Zhou. Remaining useful life estimation - A review on the statistical data driven approaches, *European Journal of Operational Research*, 213(1): 1–14, 2011.

[154] D. Banjevic and A. K. S. Jardine. Calculation of reliability function and remaining useful life for a Markov failure time process, *IMA Journal of Management Mathematics*, 17(2): 115–130, 2006.

[155] M. D. Pandey, X. X. Yuan, and J. M. van Noortwijk. The influence of temporal uncertainty of deterioration on life-cycle management of structures, *Structure and Infrastructure Engineering*, 5(2): 145–156, 2009.

[156] W. Caesarendra, A. Widodo, and B. S. Yang. Application of relevance vector machine and logistic regression for machine degradation assessment, *Mechanical Systems and Signal Processing*, 24(4): 1161–1171, 2010.

[157] J. Yan, M. Koç, and J. Lee. A prognostic algorithm for machine performance assessment and its application, *Production Planning & Control*, 15(8): 796–801, 2004.

[158] R. Khelif, B. Chebel-Morello, S. Malinowski, E. Laajili, F. Fnaiech, and N. Zerhouni. Direct Remaining Useful Life Estimation Based on Support Vector Regression, *IEEE Transactions on Industrial Electronics*, 64(3): 2276–2285, 2017.

[159] G. J. Vachtsevanos and P. Wang. Fault prognosis using dynamic wavelet neural networks, In *Proceedings of the 2001. IEEE Systems Readiness Technology Conference*, pages 857–870. IEEE, 2001.

[160] S. A. Asmai, A. S. H. Basari, A. S. Shibghatullah, N. K. Ibrahim, and B. Hussin. Neural network prognostics model for industrial equipment maintenance, In *Proceedings of the 2011 11th International Conference on Hybrid Intelligent Systems, HIS 2011*, pages 635–640. IEEE, 2011.

[161] Z. Tian. An artificial neural network method for remaining useful life prediction of equipment subject to condition monitoring, *Journal of Intelligent Manufacturing*, 23(2): 227–237, 2012.

[162] Z. Yang, P. Baraldi, and E. Zio. A comparison between extreme learning machine and artificial neural network for remaining useful life prediction, In *Proceedings of the 2016 Prognostics and System Health Management Conference (PHM-Chengdu)*, pages 1–7. IEEE, 2016.

[163] A. Ray and S. Tangirala. Stochastic modeling of fatigue crack dynamics for on-line failure prognostics, *IEEE Transactions on Control Systems Technology*, 4(4): 443–451, 1996.

[164] Y. Li, T. R. Kurfess, and S. Y. Liang. Stochastic prognostics for rolling element bearings, *Mechanical Systems and Signal Processing*, 14(5): 747–762, 2000.

[165] Y. Li, S. Billington, C. Zhang, T. Kurfess, S. Danyluk, and S. Liang. Adaptive prognostics for rolling element bearing condition, *Mechanical Systems and Signal Processing*, 13(1): 103–113, 1999.

[166] S. R. Safavian and D. Landgrebe. A Survey of Decision Tree Classifier Methodology, *IEEE*

Transactions on Systems, Man and Cybernetics, 21(3): 660–674, 1991.

[167] M. N. Anyanwu and S. G. Shiva. Comparative Analysis of Serial Decision Tree Classification Algorithms, *International Journal of Computer Science and Security*, 3(3): 230–240, 2009.

[168] E. B. Hunt, J. Marin, and P. J. Stone. *Experiments in induction*. Academic Press, 1966.

[169] S. Letourneau, F. Famili, and S. Matwin. Data mining to predict aircraft component replacement, *IEEE Intelligent Systems*, 14(6): 59–66, 1999.

[170] P. P. Bonissone and K. Goebel. Soft computing applications in equipment maintenance and service, In *Proceedings Joint 9th IFSA World Congress and 20th NAFIPS International Conference (Cat. No. 01TH8569)*, pages 2752–2757. IEEE, 2001.

[171] Q. Guan, Z. Zhang, and S. Fu. Ensemble of Bayesian predictors and decision trees for proactive failure management in cloud computing systems, *Journal of Communications*, 7(1): 52–61, 2012.

[172] C. Cortes and V. Vapnik. Support-Vector Networks, *Machine Learning*, 20(3): 273–297, 1995.

[173] G. A. Susto, A. Schirru, S. Pampuri, S. McLoone, and A. Beghi. Machine learning for predictive maintenance: A multiple classifier approach, *IEEE Transactions on Industrial Informatics*, 11(3): 812–820, 2015.

[174] A. Verma and A. Kusiak. Predictive analysis of wind turbine faults: A data mining approach, In *Proceedings of the 2011 Industrial Engineering Research Conference*, 2011.

[175] P. Cheeseman, J. Stutz, and Cheeseman. Bayesian Classification(AutoClass):Theory and Results. *Advances in Knowledge Discovery and Data Mining*, pages 153–180. American Association for Artificial Intelligence., 1996.

[176] H. Zhang. The Optimality of Naive Bayes, In *Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference FLAIRS 2004*, American Association for Artificial Intelligence, 2004.

[177] F. Di Maio, S. Ng, K.-L. Tsui, and E. Zio. Naïve Bayesian Classifier for On-line Remaining Useful Life Prediction of Degrading Bearings, In *Proceedings of the MMR 2011*, pages 1–14. , 2011.

[178] P. Bangalore and L. B. Tjernberg. An artificial neural network approach for early fault detection of gearbox bearings, *IEEE Transactions on Smart Grid*, 6(2): 980–987, 2015.

[179] F. Z. Benjelloun, A. A. Lahcen, and S. Belfkih. An overview of big data opportunities, applications and tools, In *Proceedings of the 2015 Intelligent Systems and Computer Vision (ISCV)*, pages 1–6. IEEE, 2015.

[180] L. Wang, J. Zhan, C. Luo, Y. Zhu, Q. Yang, Y. He, W. Gao, Z. Jia, Y. Shi, S. Zhang, C. Zheng, G. Lu, K. Zhan, X. Li, and B. Qiu. BigDataBench: A big data benchmark suite from internet services, In *Proceedings of the 20th International Symposium on High Performance Computer Architecture (HPCA)*, pages 488–499. IEEE, 2014.

[181] F. B. Uz. Predictive Maintenance Modelling Guide, 2016. https://gallery.cortanaintelligence.com/Collection/Predictive-Maintenance-Implementation-Guide-1, Accessed: 30.06.2017.

[182] B. Yadranjiaghdam, N. Pool, and N. Tabrizi. A Survey on Real-time Big Data Analytics : Applications and Tools, In *Proceedings of the 2016 International Conference on Computational Science and Computational Intelligence (CSCI)*, IEEE, 2016.

[183] E. Karydi and K. Margaritis. Parallel and Distributed Collaborative Filtering, *ACM Computing Surveys*, 49(2): 1–41, 2016.

[184] C.-W. Tsai, C.-F. Lai, H.-C. Chao, and A. V. Vasilakos. Big data analytics: a survey, *Journal of Big Data*, 2(21), 2015.

[185] D. Singh and C. K. Reddy. A survey on platforms for big data analytics, *Journal of Big Data*, 2(8), 2015.

[186] M. Chen, S. Mao, and Y. Liu. Big data: A survey, *Mobile Networks and Applications*, 19(2): 171–209, 2014.

[187] H. Hu, Y. Wen, T. S. Chua, and X. Li. Toward scalable systems for big data analytics: A technology tutorial, *IEEE Access*, 2: 652–687, 2014.

[188] M. G. Huddar and M. M. Ramannavar. A Survey on Big Data Analytical Tools, *International Journal of Latest Trends in Engineering and Technology*, : 85–91, 2010.

[189] S. Sakr. *Big Data 2.0 Processing Systems*. Springer: Sydney, 2016.

[190] A. Loganathan, A. Sinha, V. Muthuramakrishnan, and S. Natarajan. A Systematic Approach to Big Data Exploration of the Hadoop Framework, *International Journal of Information & Computation Technology*, 4(9): 869–878, 2014.

[191] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark : Cluster Computing with Working Sets, *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, : 1–10, 2010.

[192] R. P. Foundation. *Annual Review 2017*, 2017.

https://static.raspberrypi.org/files/about/RaspberryPiFoundationReview2017.pdf.
[193] S. J. Cox, J. T. Cox, R. P. Boardman, S. J. Johnston, M. Scott, and N. S. O'Brien. Iridis-pi: A low-cost, compact demonstration cluster, *Cluster Computing*, 17(2): 349–358, 2014.
[194] F. P. Tso, D. R. White, S. Jouet, J. Singer, and D. P. Pezaros. The Glasgow raspberry Pi cloud: A scale model for cloud computing infrastructures, In *Proceedings of the 33rd International Conference on Distributed Computing Systems Workshops*, pages 108–112. IEEE, 2013.
[195] P. Abrahamsson, S. Helmer, N. Phaphoom, L. Nicolodi, N. Preda, L. Miori, M. Angriman, J. Rikkilä, X. Wang, K. Hamily, and S. Bugoloni. Affordable and energy-efficient cloud computing clusters: The Bolzano Raspberry Pi cloud cluster experiment, In *Proceedings of the 5th International Conference on Cloud Computing Technology and Science*, pages 170–175. IEEE, 2013.
[196] A. Ashari and M. Riasetiawan. High performance computing on cluster and multicore architecture, *Telkomnika (Telecommunication Computing Electronics and Control)*, 13(4): 1408–1413, 2015.
[197] N. Schot. Feasibility of Raspberry Pi 2 based Micro Data Centers in Big Data Applications, In *Proceedings of the 23th Twente Student Conference on IT*, University of Twente, 2015.
[198] Y. Samadi, M. Zbakh, and C. Tadonki. Comparative study between Hadoop and Spark based on Hibench benchmarks, In *Proceedings of the 2nd International Conference on Cloud Computing Technologies and Applications (CloudTech)*, pages 267–275. IEEE, 2017.
[199] I. Mavridis and H. Karatza. Performance evaluation of cloud-based log file analysis with Apache Hadoop and Apache Spark, *Journal of Systems and Software*, 125(C): 133–151, 2017.
[200] N. Poggi, A. Montero, and D. Carrera. Characterizing BigBench Queries, Hive, and Spark in Multi-cloud Environments. In R. Nambiar and M. Poess, editors, *Performance Evaluation and Benchmarking for the Analytics Era. TPCTC 2017*, volume 10661 of *Lecture Notes in Computer Science*, pages 55–74. Springer Cham., 2018.