University of Applied Sciences
HOCHSCHULE EMDEN·LEER

Big Data Processing mit Apache Spark

vorgelegt von

Sascha P. Lorenz

Matrikel-Nr.: 501 63 21

dem Fachbereich Technik
der Hochschule Emden-Leer
und der Beuth Hochschule für Technik Berlin
vorgelegte Masterarbeit
zur Erlangung des akademischen Grades
Master of Science (M.Sc.)
im Studiengang
Medieninformatik (Master)

Tag der Abgabe 07. März 2015

1. Betreuer	Prof. Dr. Stefan Edlich	Beuth Hochschule für Technik Berlin
2. Betreuer	Prof. Dr. Schiemann-Lillie	Hochschule Emden-Leer

Kurzfassung

Gegenstand dieser Arbeit sind die Grundlagen der Verarbeitung und Analyse großer Datenmengen (Big Data) am konkreten Beispiel von Apache Spark. Zunächst sollen verschiedene Ansätze mit Ihren Funktionsweisen sowie den Vor- und Nachteilen diskutiert werdem. Hier werden zuerst allgemeine Grundlagen zu Big Data erarbeitet. Was ist Big Data, was unterscheidet die Verarbeitung von strukturierten und unstrukturierten Daten, Relationale Datenbanken vs. noSQL, wie müssen die Quelldaten für die jeweiligen Verarbeitungen beschaffen sein, welche besonderen Herausforderungen stellen gestreamte Daten an die Verarbeitung. Besonders wird hier auf Hadoop und den Map/Reduce-Algorithmus eingegangen, um das bisher etablierte Vorgehen zu beschreiben und ein grundsätzliches Verständnis für die Domäne "Big Data Processing" zu schaffen. In diesem Kontext wird das gesamte Ökosystem rund um Hadoop vorgestellt.

Nachdem eine Einführung in das Thema "Big Data Processing" erfolgt ist und ein entsprechend quantitativ und qualitativ brauchbarer Datensatz zur Verfügung steht, werden die Next-Generation Data-Processing Technologien betrachtet. Kernthema ist hier Apache Spark und der gesamte BDAS (Berkeley Data Analytics Stack), der von den den AMP-Labs innerhalb von Apache-Projekten um Spark herum aufgebaut wurde. Zu praktisch jeder "offiziellen"BDAS-Implementierung existieren noch Alternativen. Besonders Apache flink wird hier als Alternative näher untersucht. Auch Applikationen, die auf dem eigentlichen Stack aufsetzen, werden näher betrachtet und entsprechenden Praxistests unterzogen (beispielsweise H2O für statistische Analysen).

Danach wird die API von Spark und deren Möglichkeiten mit Scala, Java und Clojure näher betrachtet und durch jeweils eigene Implementierungen untersucht.

Die Arbeit schließt mit durch verschiedene Versuchsreihen fundierte Empfehlungen für die unterschiedlichen Anforderungen im Bereich des Big-Data-Processing.

Abstract

Inhaltsverzeichnis

Kapitel 1

Einführung

Big Data ist insbesondere in den letzten Jahren immer stärker in den verschiedensten Zusammenhängen in den allgemeinen Sprachgebrauch vorgedrungen und ist hier einem ständigen Bedeutungswandel ausgesetzt. Besonders in letzter Zeit wird dieses Thema auch verstärkt kontrovers diskutiert.

Im ersten Kapitel soll der Begriff *Big Data* jenseits von Management-Hype und Skepsis rational definiert werden. Des Weiteren werden einige grundlegende Konzepte des Umgangs mit sehr großen und unstrukturierten Datensätzen diskutiert und im Speziellen die Motivation hinter den Apache Frameworks Hadoop und Spark vorgestellt.

Das zweite Kapitel beschäftigt sich mit dem Berkeley Data Analytics Stack (BDAS), mit dem von der UCLA Berkeley rund um Hadoop ein leistungsfähiger Infrastruktur-Stack für die Einsatzbereiche von Big Data Analytics geschaffen wurde.

Innerhalb vom BDAS etabliert sich langsam auch eine schnellere und flexiblere Alternative zu Hadoop: Apache Spark. Im dritten Kapitel wird diese neue Kerntechnologie vorgestellt, die zugleich auch den Hauptteil dieses Wissenschaftlichen Projektes darstellt.

Im vierten Teil dieser Ausarbeitung wird Spark in der praktischen Anwendung gezeigt inklusive Installation und ersten kleineren Beispielen sowohl direkt in Spark, als auch aus darüber liegenden Schichten aus dem Stack.

1.1 Was versteht man unter Big Data?

Der Begriff *Big Data* wurde vermutlich zum ersten Mal Ende des 20. Jahrhunderts von John R. Marshey, damals Chefwissenschaftler bei Silicon Graphics, im Rahmen einer Usenix-Konferenz öffentlich erwähnt [Tur14]. Mittlerweile ziert dieser Begriff gefühlt jedes zweite Cover von IT-Zeitschriften mit Business-Fokus und auch Manager und *Sales-Professionals* werten Ihre Produktpräsentationen gerne mit diesem Buzzword auf. Aber dieser Begriff ist nicht nur positiv

assoziiert. Besonders seit Bekanntwerden der Tätigkeiten des Amerikanischen Auslandsgeheimdienstes weckt die Vorstellung des Datensammelns in großen Dimensionen auch Misstrauen.

Im Rahmen dieser Arbeit soll jedoch ausschließlich die technische Betrachtung und die exemplarische Darstellung von möglichen Anwendungsgebieten diskutiert werden.

Wie lässt sich der Begriff *Big Data* abgrenzen? Es existiert keine abschließend eindeutige Definition, jedoch gibt es einige Attribute, die sich in einem Großteil der Fachliteratur etabliert haben. Der Artikel aus dem O'Reilly Radar zum Thema [Dum14] fasst dies folgendermaßen zusammen:

„Big data is data that exceeds the processing capacity of conventional database systems. The data is too big, moves too fast, or doesn't fit the structures of your database architectures."

Neben der reinen Menge spielt also offensichtlich auch die mangelnde oder fehlende Strukturierung und unter Umständen die Flüchtigkeit der Daten eine nicht unerhebliche Rolle. Dies können beispielsweise Daten aus Social-Media-Quellen sein, die aus allen möglichen verschiedenen Einzeldaten bestehen, Daten von Sensoren, die permanent überwacht werden müssen, oder Datenströme (Video, Audio, Bilder, Text), die nach einheitlichen Kriterien gefiltert werden sollen, um hier nur einige Beispiele zu nennen. Auch die temporäre Komponente ist ein Einsatzgebiet für *Big Data*, und auch hier ist wieder das Beispiel der Datenströme heranzuziehen.

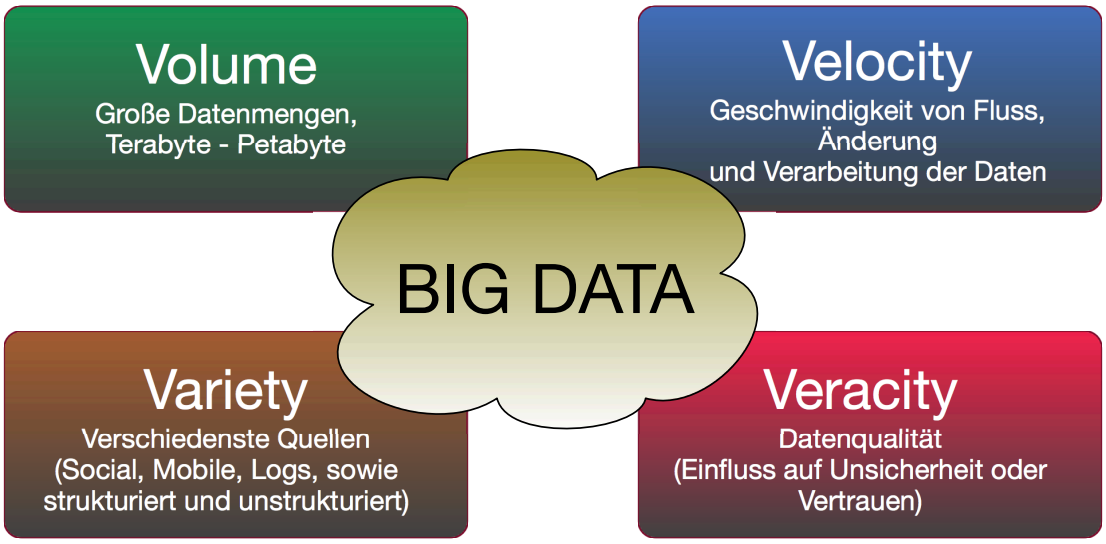

Abbildung 1.1: Darstellung der vier Säulen von Big Data: The Four V's of Big Data

Bei der Definition von *Big Data* werden laut [Tee14] auch immer wieder die „Four V's"[1] angeführt. Dies sind *Volume*, also die Datenmenge, *Variety*, die Datenvielfalt, *Velocity*, die

[1] The four V's: Es exisiteren verschiedene Versionen der Säulen von Big Data. Deshalb war lange von nur drei Säulen die Rede (ohne Veracity), aber es existieren auch Definitionen von bis zu sieben V's (zusätzlich Variability, Visualisation, Value - Vergleich [McN14]). Mittlerweile hat sich die Darstellung der Four V's jedoch etabliert.

Geschwindigkeit der Auswertung, sowie *Veracity*, die häufig stark schwankende Datenqualität. In Abbildung 1.1 wird dieser Zusammenhang dargestellt.

Die sinnvolle Analyse dieser Daten kann Unternehmen oder anderen Organisationen wichtige Informationen z.B. über Marktentwicklungen, bestimmte Kundenbedürfnisse, Epedemie-Ausbreitungen oder andere wichtige Sachverhalte liefern. Diese Analyse inklusive der dazu verwendeten Werkzeuge wird allgemein *Big Data Analytics* genannt.

1.2 Ansätze für Big Data Analytics

Die Disziplin *Big Data Analytics* umfasst Methoden und Werkzeuge zur automatisierten oder interaktiven Erkennung und daraufhin auch Verwendung von bestimmten Mustern und Assoziationen. Dies sind unter anderem:

- Prediction-Models zur Vorhersage bestimmter Sachverhalte
- statistische Verfahren, wie beispielsweise *Logistic Regression* oder *k-means-Algorithmen*
- Optimierungs- und Filteralgorithmen
- Werkzeuge zum Datamining
- Textanalyse
- Bild- und Tonanalyse
- Datenstromanalysen

Nach dem BITKOM-Leitfaden [BIT14] besteht die Taxonomie der Big-Data-Technologien grundsätzlich aus vier Schichten:

- Daten-Haltung
- Daten-Zugriff
- Analytische Verarbeitung
- Visualisierung

Diese werden durch *Daten-Integration* und *Daten-Governance*, sowie Daten-Sicherheit flankiert, um den Weg von Rohdaten bis zu nutzbaren Erkenntnissen in existierende Standards einzubetten.

Zahlreiche Hersteller herkömmlicher relationaler Datenbanksysteme versuchen derzeit, ihre bestehenden Lösungen mit dem Label *Big Data* zu versehen und diese so weiterhin in diesen

sich verändernden Marktsegmenten zu positionieren. Wenn *Big Data* jedoch jenseits der Datengröße definiert wird und auch unstrukturierte und temporäre Daten-Stacks oder –ströme zu verarbeiten oder zu analysieren sind, stoßen RDBMS [2] sehr schnell an ihre Grenzen. Doch auch was die Skalierbarkeit angeht, sind relationale Datenbanken meist nicht hinreichend flexibel [Lou14].

Für die Anforderungen an dedizierte Aufgaben im Bereich *Big-Data-Analytics* sind seit einigen Jahren einige *Frameworks* auf dem Markt, die in allen drei oben genannten Aspekten besser geeignet sind, als RDBMS. Der Ansatz ist hier primär, die Verarbeitung zu dezentralisieren, also auf unabhängige Knoten in einem Rechner-Cluster zu verteilen und nur Referenzen auf die Clusterknoten zentral zu verwalten.

Es existieren mittlerweile Lösungen am Markt, die speziell diese Aufgaben für derartige Aufgaben entwickelt wurden. Hier wären unter anderem Hadoop, Spark, HPCC, GPMR, Mincmeat, Sphere, Bashreduce und R3 zu nennen. Bis auf HPCC setzen alle eben genannten Implementierungen generell oder in Teilen auf das Programmiermodell MapReduce.

Der zweifellose De-facto-Standard in diesen Bereichen ist bereits seit einiger Zeit das Open-Source-Framework Apache Hadoop. Auf Hadoop basierend existieren etliche Derivate. Unter anderem sind hier Cloudera, Amazon Elastic MapReduce, Apache BigTop, Datameer, Apache Mahout, MapR und IBM PureData System zu nennen.

1.3 Motivation für Apache Hadoop/Spark

Anfang des 21. Jahrhunderts wurde das Bedürfnis für Möglichkeiten, sehr große Datenmengen effizient verarbeiten zu können, stetig größer. Nicht zuletzt durch die zu dieser Zeit exponentiell steigende Menge von Inhalten im World Wide Web und deren Indexierung durch Suchmaschinen wie Google. Davon motiviert wurde 2002 das Projekt *Nutch* mit dem Ziel gestartet, ein geeignetes *Such- und Crawlersystem* frei verfügbar zu machen. Die ersten Versuche skalierten sehr schlecht, bis Google 2003 die Funktionsweise ihres verteilten Dateisystem GFS (Google File System) veröffentlichte. Somit konnten die sehr großen Dateien, die durch die Indexierung entstanden, effizient auf verschiedene Knoten verteilt gespeichert werden und die Verwaltung dieser Knoten und Dateien aus dem eigentlichen Indexierungs- und Suchprozess ausgelagert werden.

Im Jahre 2004 publizierte Google den *MapReduce-Algorithmus*, der unter anderem die Indexierungs- und Analysefunktionen parallelisieren, delegieren und sinnvoll bündeln kann. In Nutch wurden daraufhin sämtliche wichtige Algorithmen auf MapReduce umgestellt, nachdem zuvor auch GFS unter dem Namen NDFS (Nutch Distributed File System) integriert wurde. Die möglichen Anwendungsgebiete von Nutch waren damit auch weit über das reine Suchen und

[2] RDBMS = Relational Database Management System, also ein relationales Datenbanksystem (im Gegensatz zu Objekt- oder Graphdatenbanken).

Indexieren von Webseiten hinaus gewachsen. 2006 wurde aus Nutch ein Unterprojekt mit dem Namen Hadoop ausgegliedert, das im Jahre 2008 zum *Apache Top-Level-Projec*t ernannt wurde. Zu dieser Zeit nutzten bereits Firmen wie Yahoo!, Facebook oder die New York Times Hadoop. Ein exemplarischer Anwendungsfall bei der NY Times war, mit Hilfe der Hadoop-basierten EC2-Cloud von Amazon ca. vier Terabyte gescannter Archivdateien in PDF-Dateien umzuwandeln und dies in weniger als 24 Stunden auf 100 Knoten. Auch beim Sortieren von sehr großen Datenmengen stellten Hadoop-basierte Systeme nach und nach sämtliche Rekorde ein [Whi13].

Hadoop und Hadoop-basierte System gelten mittlerweile als Industriestandard für Big-Data-Analytics-Anwendungen. Jedoch ist Hadoop nicht für alle Anwendungsgebiete gleichermaßen geeignet. Aufgrund der Charakterisierung der Paradigmen für Big Data Analytics im Paper „Frontiers in Massive Data Analysis“ der National Academic Press [Cou13], lassen sich die Einsatzgebiete und Schwächen für Hadoop ermitteln [Agn14].

So lassen sich mit Hadoop einfachere statistische Aufgabenstellungen sehr gut umsetzen. Dazu gehören Mittelwert, Median, Varianz und allgemein abzählende sowie ordnende Statistikaufgaben. Dies sind in der Regel Anwendungen mit einer Laufzeitkomplexität von O(n) für n Betrachtungswerte. Sie sind meist auch sehr gut parallelisierbar und somit sehr gut für Hadoop geeignet.

Für linear-algebraische Berechnungen (lineare Regression, Eigenwertproblem, Hauptkomponentenanalyse), generalisierte n-Körper-Probleme (mit einer Komplexität von $O(n^2)$ oder $O(n^3)$), Graphentheorie, Optimierungsprobleme (Verlust-, Kosten- oder Energiefunktionen, sowie Integrations- und Ausrichtungsfunktionen ist *Hadoop* nur in jeweils einfacher Problemausprägung einsetzbar. Auch für Interaktive Abfragen ist *Hadoop* nur bedingt geeignet, da es ursprünglich für die *Batch-Verarbeitung* entwickelt wurde.

Aus diesem Grund wurde am *AMPLab* der University of California in Berkeley nach Alternativen geforscht, die auch für komplexe linear-algebraische Probleme, generalisierte n-Körper-Probleme und diverse Optimierungsprobleme geeignet sind. Das Ergebnis ist *Spark*, mittlerweile *Apache Top-Level-Projekt* und dazu geeignet, die Nachfolge von Hadoop als *Big-Data-Analytics-Framework* anzutreten.

1.4 Ziel und Aufbau dieser Arbeit

Die vorliegende Arbeit beschäftigt sich mit *Apache Spark* und dem dazugehörigen Ökosystem bestehend aus aus einem Applikationsstack mit verschiedenen Bibliotheken, dem *Berkeley Data Anaytics Stack (BDAS)*. Aufgrund der Konzepte von Spark ist dieses unter anderem prädestiniert für Machine-Learning-Anwendungen. Diese bilden, neben der eigentlichen Kernimplementierung von Spark, einen zentralen Teil dieser Arbeit. Auf dem Markt befindet sich

bislang noch verhältnismäßig wenig Literatur zu diesem Thema und wenn, dann werden zumeist Teilaspekte für bestimmte Anwendungsbereiche gekapselt betrachtet.

Diese Arbeit soll, nachdem einige Grundlagen zum Thema *Big Data Analytics* im Allgemeinen diskutiert werden, zunächst einen ganzheitlichen Überblick über den BDAS bieten. Hier werden die einzelnen Schichten des Stacks kurz beschrieben und gegebenenfalls Alternativlösungen zu den jeweiligen Implementierungen vorgestellt.

Insbesondere wird in den darauffolgenden Grundlagenkapiteln die Machine-Learning-Bibliothek *MLLib* betrachtet. Um ein Verständnis für die Funktionen dieser Bibliothek zu vermitteln, wird zunächst eine Einführung in die Grundlagen in die Themengebiete Data-Mining und Machine-Learning vermittelt. Anschließend werden die elementaren Machine-Learning-Algorithmen vorgestellt, welche in MLLib implementiert sind. Auch die übrigen Elemente des BDAS wie das Caching-Frameworks Tachyon, sowie die Streaming-Bibliothek Spark Streaming und die Graphenanwendung GraphX werden vorgestellt. Anschließend werden die APIs der einzelnen Bibliotheken gezeigt. Diese bilden die Grundlage für eigene Anwendungen mit Spark.

Des weiteren wird die BDAS-Bibliothek *MLLib* mit der Alternativbibliothek *H2O* verglichen. Außerdem wird ein Vergleich zwischen der Kernimplementierung von Spark wird mit dem neueren *Big Data Analytics Framework Apache Flink*[3] vorgestellt.

Im letzten Teil dieser Arbeit wird der Aufbau von lokalen BDAS-Stacks für verschiedene Anwendungsbereiche gezeigt. Darauf folgt die Beschreibung von Prototypen die zum Test der Spark-Infrastruktur im Rahmen dieser Arbeit implementiert wurden. Zum Schluss werden für die jeweiligen Anwendungsbereiche des BDAS Metriken definiert, welche für Messungen der Prototypen benutzt werden.

In einem Ausblick werden die Erkenntnisse aggregiert und mögliche weitere Entwicklungen prognostiziert.

[3] Entwickelt von der Technischen Universität Berlin zunächst unter dem Namen *Stratosphere* und mittlerweile (Stand Ende 2014) *Apache Incubator* Projekt

Kapitel 2

Allgemeine Grundlagen

Das nachfolgende Kapitel behandelt die Grundlagen, die für ein Verständnis der Anwendungsbereiche von Apache Spark, dem Berkeley Data Analytics Stack und im Allgemeinen des Themenkomplexes Big Data Analytics und insbesondere für Machine-Learning-Anwendungen nötig sind. Im ersten Unterkapitel werden die grundsätzlichen Eigenschaften eines verteilten Systems beschrieben um die Basis für die in der Arbeit beschriebenen Besonderheiten von Verarbeitungen im Clusterbetrieb zu legen. Hier wird ein exemplarischer Clusteraufbau skizziert, Probleme mit Concurrency und Netzwerkverkehr beschrieben und welche Möglichkeiten es hier gibt. Im darauf folgenden Unterkapitel werden grundlegende Problemstellungen und Technologien beschrieben, die im Rahmen von Big Data Analytics im Allgemeinen vorkommen. Unter anderem werden hier Grundlagen und Begriffe aus den Themengebieten Anwendungen von Big Data Analytics, Machine Learning, Klassifikation, Vorhersagen, statistische Analysen, Graph-Suchen und Streaming-Frameworks in erklärt. Besonders die Algorithmen, die in den Machine-Learning-Implementierungen MLibs und H20 zum Einsatz kommen, werden hier detaillierter vorgestellt. In einer Zusammenfassung werde diese Grundlagen nochmals auf einen Blick dargestellt.

2.1 Cluster Computing

Die Nachfrage nach immer mehr Rechenleistung hat in dein letzten Jahren dazu geführt, dass verstärkt Rechnercluster eingesetzt werden. Alternativ gibt es den Ansatz, Mainframes[1] mit immer mehr Rechenleistung auszustatten, diese jedoch ausdrücklich autonom zu betreiben[2]. Je nach Aufgabenspektrum ist die eine oder andere Infrastruktur besser geeignet. In der Regel wird ein geclustertes System dort eingesetzt, wo hohe Verfügbarkeit oder gut parallelisierbare

[1] Unter Mainframe wird hier ein sehr leistungsfähiges Rechnersystem verstanden, das einen oder beliebig viele Prozessoren in einer physischen Einheit, also einem logischen Mainboard verbindet.

[2] In diesem Kontext kann durchaus ein Failover-Cluster vorhanden sein, also eine Mainframe wird zur Ausfallsicherheit repliziert. Dies wird an dieser Stelle jedoch nicht als Cluster im eigentlichen Sinn bezeichnet.

Aufgaben vorherrschen. Bei netzwerkintensiven Aufgaben, wie z.B. als Webserver oder Datenbanksystem sollten in der Regel besser Installationen auf einem autonomen System eingesetzt werden [TR10].

Ein Rechner-Cluster besteht in der Regel aus mehr oder weniger eng miteinander verbundenen Computern, wobei hier im Gegensatz zu Mainframes jeder Rechner über eigene Ressourcen wie Hauptspeicher, Massenspeicher, etc. verfügt. Ein Cluster, bzw. ein Verteiltes System ist nach Andrew S. Tanenbaum [AST07] folgendermaßen definert:

„A distributed system is a collection of independent computers that appears to its users as a single coherent system.“

Bei der Verwendung eines Clusters sind einige Besonderheiten zu beachten, die bei der Ausführung auf gewöhnlichen Systemen nicht ins Gewicht fallen [AST07]. Unter Anderem sind die Tasks so gestalten, dass möglichst wenig Wartezeit durch Abhängigkeiten entsteht und diese möglichst autonom verarbeitet werden können. Außerdem muss beachtet werden, dass die einzelnen Knoten eines Clusters über Messaging-Mechanismen miteinander kommunizieren und dies insbesondere hohe Anforderungen an die Netzwerkinfrastruktur stellt. Eine typische Clustertopologie besteht aus mehreren Worker-Knoten und einem Masterknoten. Der Masterknoten deligiert die Tasks an die einzelnen Worker-Knoten und stellt das gesamte Cluster nach außen hin als ein geschlossenes System dar. Sämtliche Kommunikation mit dem Cluster findet grundsätzlich nur über den Masterknoten statt.

2.2 Anwendungen für Big Data Analytics

Im folgenden Unterkapitel werden exemplarisch einige Anwendungsfälle für *Big Data Analytics* (auch *Data Mining*) dargestellt, um zu klären, für welche Einsatzbereiche *Frameworks* wie *Apache Spark* und die darauf aufbauenden Bibliotheken in der Praxis benötigt werden.

Laut Arvind Sathi [Sat12] zeichnet sich *Big Data* unter Anderem durch ein mögliches Vorkommen von unstrukturierten Daten aus. Die Autoren Chakraborty und Pagolu gehen in ihrem Artikel [DGC14] davon aus, dass mittlerweile mehr als 80% der gesamten Daten im digitalen Raum in unstrukturierter Form vorliegen. In der Vergangenheit mussten für Analysetätigkeiten in aller Regel strukturierte Datensätze vorliegen. Grundsätzlich ist es mit den gängigen *Data Analytics Frameworks* nach wie vor möglich, beispielsweise quantitative Analysen auf strukturierten Datensätzen durchzuführen oder unstrukturierte Datensätze nachträglich zu strukturieren, um wiederum quantitative Analysen darauf anwenden zu können.

Das Potential dieser *Frameworks* zeigt sich jedoch dann in vollem Umfang, wenn auf unstrukturierten Daten diverse Analysemethoden oder Verarbeitungen angewendet werden. In jüngerer Vergangenheit ist das Aufkommen unstrukturierter Daten, wie bereits erwähnt, erheblich gestiegen. Dies wird nicht zuletzt durch die massive Verbreitung von Sensoren aller

Art verursacht. Dies können *Logdaten*, Bewegungsdaten, Sensorwerte zur Überwachung von technischen Einrichtungen, Messwerte aus Wetterstationen und unzähligen weiteren Quellen sein. Auch viele Internetanwendungen, besonders wenn es sich um laufende Datenströme handelt, verursachen erhebliche Datenmengen, die entweder *persistiert* oder sogar zur Laufzeit analysiert werden können.

Data Mining[3] ist laut [JC14] ein analytischer Prozess mit dem Zweck, große Datenmengen nach konsistenten Mustern oder systematischen Beziehungen zu untersuchen. Die Ergebnisse werden in der Regel validiert, in dem gefundene Muster oder Ähnlichkeiten auf einer Teilmenge der ermittelten Daten angewendet werden. Ein weiteres Ziel von *Data-Mining-Prozessen* sind Vorhersagen von Ereignissen mittels geeigneter Algorithmen (Vergleich [JC14] und 2.3).

Der Prozess des Data Mining setzt sich nach [UF96] im Wesentlichen aus einem oder mehreren der folgenden Aufgabenbereiche zusammen:

- **Klassenbeschreibung:** Eine knappe Beschreibung der Charakterisierung der Datensätze, um sie eindeutig von anderen Daten unterscheiden zu können.
- **Assoziation:** Die Untersuchung der Daten nach assoziativen Verbindungen oder Korrelationen zwischen einzelnen Daten oder Datengruppen.
- **Klassifizierung:** Hier wird ein definierter Satz von Trainingsdaten[4] analysiert und anhand deren Beschaffenheit ein Modell generiert. Durch die Klassifizierung werden Entscheidungsbäume (Vergleich Kapitel 2.3) oder Klassifizierungsregeln generiert, die schließlich für die Klassifizierung folgender Daten verwendet werden [SW97].
- **Vorhersage:** Die Vorhersage bezieht sich auf mögliche Werte von nicht-vorhandenen Daten oder Datenspektren, die wiederum durch *Approximation* einer Funktion mittels Beispielen durchgeführt wird [JC14]. Dies sind ebenfalls Trainingsdaten, die aus Datensätzen mit den dazugehörigen berechneten Funktionswerten bestehen.
- **Cluster-Analyse:** Diese dient dazu, *Cluster* innerhalb von Datensätzen zu ermitteln. Dies sind Daten, die definierte Ähnlichkeiten zueinander aufweisen.
- **Zeitreihenanalysen:** Hier werden in der Regel große Mengen an Zeitreihendaten analysiert, um nach Ähnlichkeiten oder Mustern innerhalb der Daten zu suchen.

[3] Ein Großteil der Literatur verwendet die Begriffe *Big Data Analytics* und *Data Mining* synonym. *Machine Learning* wird jedoch von *Data Mining* abgegrenzt, da letzteres eine explorative Datenanalyse darstellt.

[4] Trainingsdaten können beispielsweise Daten sein, die bereits im Vorfeld manuell klassifiziert wurden, deren Klassenzugehörigkeit also bekannt ist.

2.3 Machine Learning

„Learning denotes changes in the system that are adaptive in the sense that they enable the system to do the same task (or tasks drawn from a population of similar tasks) more effectively the next time.“ [HS83]

Unter *Machine Learning* wird ein interdisziplinärer Teilbereich der Informatik und der Statistik verstanden. Ziel ist die Erstellung von Algorithmen, die in der Lage sind, selbstständig auf Grund von Daten iterativ zu lernen gemäß der oben zitierten Definition. Um dies zu erreichen, erstellen diese Algorithmen basierend auf den jeweiligen Eingabedaten Modelle, die Entscheidungen oder Vorhersagen treffen können [GJ13]. In den Abbildungen 2.1 und 2.2 wird dies veranschaulicht.

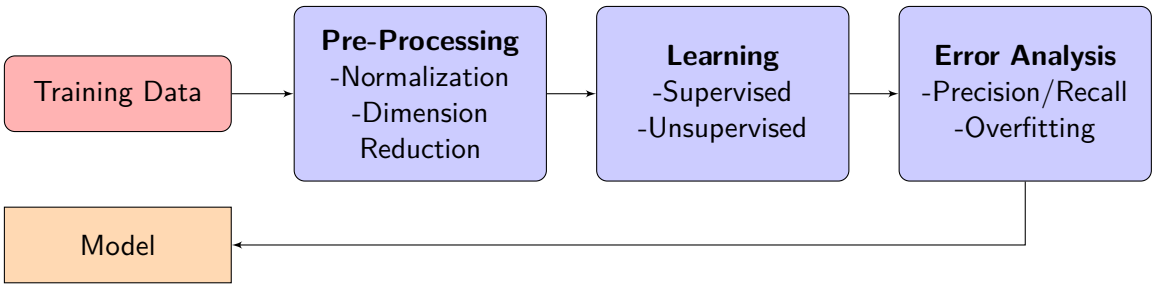

Abbildung 2.1: Der Machine-Learning-Prozess: Phase 1 - Lernphase

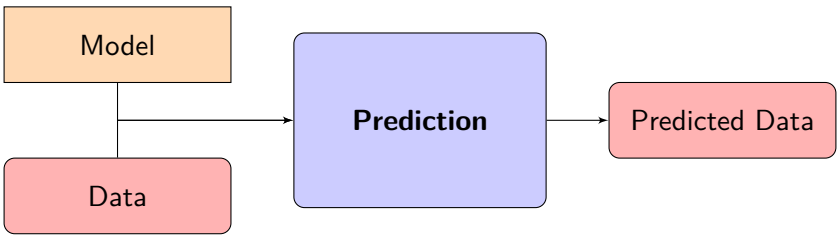

Abbildung 2.2: Der Machine-Learning-Prozess: Phase 2 - Prediction-Phase (Vorhersage)

Das erste Diagramm 2.1 zeigt den Lernprozess. Zunächst werden definierte Trainingsdaten einem *Pre-Processing*[5] zugeführt (Vergleich Unterkapitel 2.3.1).

Danach folgt der eigentliche Lernprozess, der mit oder ohne *Überwachung*[6], als *Minimalisierungsfunktion* oder mit anderen Lernalgorithmen durchgeführt wird (Vergleich Unterpunkte Clusterverfahren und Klassifikationsalgorithmen). Der Lernprozess kann entweder mit einem bestimmten Lernalogrithmus durchgeführt werden, oder mit sogenannten *Ensemble-Learning-Methods*. Dies ist die Zusammenfassung von unterschiedlichen Lernalgorithmen zu einem Lernprozess, um die Vorhersageleistung zu steigern. Nach dem Lernprozess werden das erzeugte

[5] Das Pre-Processing kann aus *Normalisierung, Dimensionsreduktion, Bildverarbeitung* oder anderen Vorarbeiten bestehen.

[6] Beim überwachten Lernen (Supervised Learning) liegen der Analyse vorher angelegte Trainingsdaten zugrunde, das unüberwachte Lernen (Unsupervised Learning) erzeugt vorher gänzlich unbekannte Modelle aus gefundenen Mustern [Gha15]

Modell einem Validierungsprozess zugeführt. Dies können *Precision/Recall*[7], *Overfitting*[8], oder andere Validierungsfunktionen sein. Am Ende der Prozesse entsteht ein Modell, das für verschiedenste Aufgaben eingesetzt werden kann.

In Abbildung 2.2 wird das erzeugte Modell zusammen mit produktiven Daten genutzt, um mittels geeigneter Vorhersagealgorithmen (Vergleich Unterkapitel 2.3.1) bisher noch nicht vorhandene Datensätze erzeugen zu können.

Die Hauptanwendungsgebiete für *Machine Learning* sind sämtliche Bereiche, in denen eine Anwendung von strikten, regelbasierten Algorithmen nicht in Frage kommt [SW97]. Beispiele für diese Bereiche sind laut [Bis06] Suchmaschinen, Sprach- und Musikerkennung, Handschriftenerkennung, Spamfilter, Umgebungserkennungen und viele mehr.

2.3.1 Klassifizierung von Daten

Im Gegensatz zu herkömmlichen *Business-Intelligence-Anwendungen*[9] zeichnen sich Big-Data-Anwendungen in erster Linie durch Unterschiede in der Herangehensweise und in der Formulierung der primären Fragestellungen aus. In der Vergangenheit wurden zu Beginn einer Analyse zunächst Problemstellungen formuliert und aufgrund dessen Prozesse und vor allen Dingen die zu sammelnden Daten definiert [NG12]. Mit der steigenden Etablierung von Big-Data-Anwendungen hat sich dieser Ansatz gewandelt. Hier werden zunächst sämtliche anfallenden Daten gespeichert. Auf diese Datensätze, deren Größe in relativ kurzer Zeit massiv anwachsen kann, werden erweiterte Analyseprozesse[10] angewendet, die unter anderem auch Vorhersagen erlauben [Rus11]. So liegen also bei Big-Data-Analytics große Datenmengen vor, ohne dass das Ziel der Analyse im Vorfeld bekannt ist.

Clusterverfahren

Nun gibt es verschiedene Herangehensweisen, wie sich relevante Fragestellungen aus den vorliegenden Daten ableiten lassen. Ein sinnvoller Ansatz besteht darin, sogenannte Cluster innerhalb der Daten zu ermitteln, also ähnliche Datensätze zu entsprechenden Gruppen zusammenzufassen. Aus diesen Gruppen werden Klassen gebildet, in dem ihnen aussagekräftige Namen zugeordnet werden[JC14]. Cluster werden beispielsweise benötigt, um Kunden nach bestimmten

[7] *Precision:* Wie viele der ausgewählten Datensätze sind relevant. *Recall:* Wieviele relevante Datensätze wurden ausgewählt.

[8] *Overfitting* bezeichnet einen Zustand, in dem bekannte Daten (Trainingsdaten) von einem Algorithmus generell akkurater erkannt werden, als Daten, die von dem Algorithmus erkannt werden sollen (Predicitions).

[9] Laut [NG12] versteht man unter *Business Intelligence (BI)* die Sammlung, Auswertung und Darstellung von Daten anhand definierter Prozesse.

[10] Unter dem Begriff *Advanced Analytics* sind verschiedene Werkzeugtypen aus der vorhersagenden Analyse (*Predictive Analytics*), aus dem Data Mining, aus der Statistik, der Künstlichen Intelligenz, der Sprachverarbeitung und weiteren Disziplinen zusammengefasst [Rus11].

Interessen zu gruppieren, um Zeichen auf Ähnlichkeiten für die automatischen Zeichenerkennung zu untersuchen, oder Bilder auf vergleichbare Bildpunktanordnungen oder Farbspektren, also generell überall, wo nach Ähnlichkeiten gesucht wird.

Ein Cluster zeichnet sich dadurch aus, dass die Objekte innerhalb eines Clusters möglichst ähnlich sind, die Cluster untereinander jedoch möglichst unähnliche Inhalte besitzen. Um Ähnlichkeiten zwischen Datensätzen oder Clustern feststellen zu können, bedarf es einer Distanzfunktion, zur Ermittlung und Sicherstellung der Clustergüte einer Qualitätsfunktion (Vergleich [JC14]).

Laut [JC14] sind für Distanz- und Qualitätsfunktionen folgende Annahmen zu treffen:

Vorbedingungen

X (Instanzenraum)

$E \subseteq X$ (Instanzenmenge)

$X \times X \rightarrow \mathbb{R}^+$ (Abstandsfunktion)

$2^{2^x} \rightarrow \mathbb{R}$ (Qualitätsfunktion)

Gesucht wird eine Clustermenge $C = \{C_1, ..., C_k\}$ mit folgenden Eigenschaften:

$C_i \subseteq E$
$quality(C) \rightarrow max$
$C_i \cap C_j = \varnothing$
$C_1 \cup ... \cup C_k = E$

Die Abstandsfunktion $dist$, die gegebene Objekte so in Teilmengen zerlegt, dass der Abstand der Objekte innerhalb einer Teilmenge (Cluster) kleiner ist, als der Abstand zu Objekten anderer Teilmengen:

$\forall C_i, C_j \in C (i \neq j) : \forall x_k, x_l \in C_i, x_m \in C_j : dist(x_k, x_l) < dist(x_k, x_m)$

Die Qualitätsfunktion $quality$ beschreibt die Qualität des Clusterings basierend auf der Abstandsfunktion $dist$:

$quality(C = \{C_1 \subseteq E, ..., C_k \subseteq E\}) \rightarrow \mathbb{R}$

Exemplarisch werden im Folgenden einige Distanzfunktionen dargestellt, die für die Clusteranalyse wichtig sind (Vergleich [CV14], [Bro98]):

- **Hamming-Distanz:** Hier werden die Vektorelemente der Position i zweier Vektoren miteinander verglichen. Wenn sich diese unterscheiden, wird die Distanz auf den Wert 1 gesetzt, bei gleichen Werten 0. Nachdem alle Elemente der Vektoren miteinander verglichen wurden, werden diese summiert. Ein Vorteil der Hamming-Distanz ist, dass sich Abstände zwischen metrischen, ordinalen und sogar nominalen Daten berechnen lassen. Der Nachteil dieser Distanzfunktion ist, dass Abstandsverhältnisse nicht erkannt werden, also ist beispielsweise der Abstand zwischen 2 und 3 für diese Funktion gleichwertig zum Abstand von 2000 zu 2100.

  $dist_H(x, y) = count_i(x_i \neq y_i)$

- **Manhattan-Distanz:** Diese Distanzfunktion hat ihren Namen aufgrund der blockweisen Anordnung der Straßenzüge in Manhattan. Distanzen werden, wie auf einem Schachbrett, durch vertikale und horizontale Bewegungen und Richtungsänderungen beschrieben. Um die Ergebnisse der verschiedenen Distanzfunktionen untereinander vergleichbar zu machen, wird der Betrag der Distanzwerte verwendet. Die Manhattan-Distanz ist nur auf metrische Daten anwendbar. Diese Distanzfunktion ist auch unter dem Namen *Block-Distanz* bekannt.

  $dist_M(x, y) = \sum_i \mid x_i - y_i \mid$

- **Euklidische Distanz:** Hier wird der direkte Abstand zwischen zwei Vektoren beschrieben, also zweier Punkte im n-dimensionalen Raum. Diese Funktion lässt sich nur auf metrische Daten anwenden. Auch hier wird durch Quadrieren und der Quadratwurzel der Differenzen eine Normalisierung erreicht, die die Distanzwerte vergleichbar macht.

  $dist_E(x, y) = \sqrt{\sum_i (x_i - y_i)^2}$

- **Tschebyscheff-Distanz:** Auch hier wird der Abstand zwischen zwei Punkten im n-dimensionalen Raum betrachtet. Im Unterschied zu den beiden vorhergehenden Verfahren wird hier jedoch der größtmögliche absolute Abstand zwischen allen Attributen ermittelt. Deshalb wird diese Distanzfunktion auch als *Maximum-Distanz* bezeichnet.

  $dist_T(x, y) = max_i(\mid x_i - y_i \mid)$

- **Minkowski-Distanz:** Diese Distanz-Funktion ist ähnlich zur Euklidischen Distanz. Allerdings entspricht hier der Abstand der n-ten Wurzel der Summe der n-ten Potenzen der Differenzen. Auch diese Funktion lässt sich nur auf metrische Daten anwenden.

  $dist_K(x, y) = \sqrt[n]{\sum_i (x_i - y_i)^n}$

Darüber hinaus existieren noch zahlreiche weitere Distanzfunktionen, die je nach Einsatzzweck besser oder schlechter geeignet sein können, als die dargestellten.

Die Clusteranalyse zählt zu den *unüberwachten Lernalgorithmen (Unsupervised Learning)*. Zoubin Ghahramani beschreibt dies in seinem Artikel *Unsupervised Learning* [Gha15] wie folgt:

„[...]in unsupervised learning the machine simply receives inputs x1, x2,..., but obtains neither supervised target outputs, nor rewards from its environment. It may seem somewhat mysterious to imagine what the machine could possibly learn given that it doesn't get any feedback from its environment. However, it is possible to develop of formal framework for unsupervised learning based on the notion that the machine's goal is to build representations of the input that can be used for decision making, predicting future inputs, efficiently communicating the inputs to another machine, etc. In a sense, unsupervised learning can be thought of as finding patterns in the data above and beyond what would be considered pure unstructured noise.“

Prinzipiell wird zwischen vier verschiedenen Clusterarten unterschieden [JC14]:

- **Partitionierendes Clustering:** Eine Menge von Datenobjekten wird in k Cluster, die um einen Medoid[11] oder einen Centroid[12] gebildet werden, zerlegt (Vergleich [Mir15]).
- **Hierarchisches Clustering:** Hier werden die Cluster hierarchisch aufgebaut, indem jeweils die Cluster mit der größten Ähnlichkeit, also mit der geringsten Distanz, miteinander vereinigt werden. Aus diesen vereinigten Clustern können in der Hierarchie übergeordnete Ebenen entstehen. Beim *Agglomerativen Clustering* wird mit den einzelnen Datenobjekten selbst begonnen, in dem jedes Cluster aus genau einem Datenobjekt besteht. Die jeweils ähnlichsten Cluster werden vereinigt und somit zu einer neuen Hierarchieebene. Dies geschieht so lange, bis in der obersten Ebene ein Cluster mit allen Datenobjekten entstanden ist. Beim *Divisiven Clustering* wird die Hierarchie in umgekehrter Reihenfolge aufgebaut, also zunächst werden alle Datenobjekte zu einem großen Cluster zusammengenommen und dieser dann iterativ anhand der Ähnlichkeiten der Datenobjekte untereinander geteilt.
- **Dichtebasiertes Clustering:** Die Cluster werden so gebildet, dass in der Umgebung eines Datenobjektes innerhalb eines Clusters möglichst viele weitere Datenobjekte liegen, die Dichte also besonders hoch ist. Die geforderte Dichte wird über Schwellenwerte definiert.
- **Clustering mit Neuronalen Netzen:** Auch über neuronale Netze können Cluster gebildet werden. Die Netze werden in diesem Fall nicht überwacht trainiert (Vergleich Einleitung Kapitel 2.3) und nutzen das Verfahren des *Wettbewerbslernens* (Vergleich [JC14], beispielsweise *Vonoroi-Mengen*). Das Neuron, welches dem Eingabewert (dem Datenobjekt) am ähnlichsten ist, ist im jeweiligen Wettbewerb im Vorteil.

[11] Medoide sind Stellvertreterobjekte eines Clusters, deren durchschnittliche Ähnlichkeit zu allen Datenobjekten des Clusters maximal ist. Ein Medoid ist immer ein Datensatz aus dem Cluster.

[12] Centroide sind Vektoren der Mittelwerte der Attribute aus den Datenobjekten. Im zweidimensionalen Raum entspricht dies der Mittelwerte der x- und y-Koordinaten aller Datenobjekte.

Zu den Cluster-Verfahren zählen laut [WKH12] und [JC14] unter Anderem folgende Algorithmen:

- **klassischer k-Means-Algorithmus (partitionierendes Verfahren):** Wie bei allen partitionierenden Verfahren werden zunächst die Clusterzahl k und je Cluster ein Zentrum definiert. Die Zentrumsdefinition, und somit die Clusterbildung findet zufällig statt. Die Repräsentation des Clusters erfolgt durch den Centroid, also den Schwerpunkt. Eine Fehlerfunktion wird iterativ minimiert, indem die euklidischen quadrierten Abstände der Inhaltsobjekte zu den Centroiden hin verschoben werden. Nach jedem Iterationsschritt werden durch Mittelwertbildung die Centroiden neu definiert. Dieses Verfahren wird so lange iterativ durchgeführt, bis kein Datenobjekt mehr sein Cluster wechselt. Der k-Means-Algorithmus setzt metrische Werte[13] voraus. Die Umwandlung von ordinalen Daten in metrische ist meist einfach umzusetzen, nominale Daten können mittels *Binarisierung* in numerische Werte gewandelt werden. Alternativ kann auch ein modifiziertes *k-Means-Verfahren* eingesetzt werden (Vergleich [JC14], Kapitel 8).
- **k-Medoid-Verfahren (partitionierende Verfahren):** Im Unterschied zum k-Means-Algorithmus wird hier nicht der Centroid, sondern der Medoid als Stellvertreter für ein Cluster definiert. Ein Medoid muss immer ein Element der Datenobjekte sein. Durch Vertauschung wird nun iterativ nach qualitativ hochwertigeren Medoiden gesucht, also solchen, deren mittlere Distanz zu den anderen Datenobjekten im Cluster geringer ist. Dies wird so lange durchgeführt, bis kein Tausch der Medoiden mehr stattfindet, also jedes Cluster seinen qualitativ hochwertigsten Medoid besitzt. Die beiden wichtigsten k-Mediod-Verfahren sind PAM (Partitoning Around Medoids) und CLARANS (Clustering Large Applications based on RANdomized Search) (Vergleich [Ng02]). Bei PAM wird jeweils nach dem besten neuen Mediod mit Hilfe einer Qualitätsverbesserung durch graphische Repräsentation der Datenobjektverteilung und -häufigkeit gesucht. Allerdings ist PAM nur für kleine Datenmengen zu verwenden. CLARANS ist für große Datenmengen besser geeignet, da hier nicht sämtliche Datenobjekte durchsucht werden. Statt dessen beschränkt sich die Suche auf Teilmengen.
- **k-Median-Algorithmus (partitionierendes Verfahren)** Im Gegensatz zu den k-Means-Verfahren wird beim k-Median nicht mit der Euklidischen Distanz der Abstand zu den Zentren ermittelt, sondern die Summen der Manhattan-Distanzen (1-Norm-Distanzen) werden iterativ minimiert [BJA06].

[13] *Metrisch*: Merkmal, das aus einer Zahl besteht und Dimension, sowie Nullpunkt besitzt (z.B. Geschwindigkeit, Einkommen, Alter), *ordinal*: Merkmal mit natürlicher Ordnung (z.B. Schulnoten sehr gut, gut, befriedigend,...), *nominal*: Merkmal ohne natürliche Ordnung (z.B. Geschlecht, Name)

- **EM-Clustering (partitionierendes Verfahren):** Dieses Verfahren basiert auf dem Expectation-Maximization-Alogrithmus. Dieser arbeitet nach dem Prinzip, zunächst mit einem zufällig gewählten Modell zu starten. In der Expectation-Phase werden die Zuteilungen der Datenobjekte zum Modell verbessert, in Maximization-Phase werden die Modellparameter entsprechend der aktuellen Zuteilung verbessert, also das Modell wird analog der Datenobjekte angepasst [CBD08]. Das EM-Clustering ist im Gegensatz zum k-Means-Algorithmus in seiner Zuordnung unscharf, da prinzipiell jedes Datenobjekt jedem Cluster zugehörig sein könnte, sowie jedes Datenobjekt jeden Parameter verändern könnte.

- **Fuzzy C-Means (partitionierendes Verfahren):** Die Entfernung vom Clusterzentrum bestimmt über den Zugehörigkeitsgrad eines Datenobjekts. Der Zugehörigkeitsgrad wird mittels eines Intervalls zwischen 0 und 1 angegeben, wobei 1 eine totale Zugehörigkeit zum jeweiligen Clusterzentrum bedeutet. Die Verschiebung der Zentren findet im Gegensatz zum k-Means-Algorithmus jedoch abhängig vom Zuordnungsgrad ab. So werden Datenobjekte mit dem Zuordnungsgrad 1 vollständig bei der Berechnung berücksichtigt, während kleinere Zugehörigkeitsgrade den Einfluss entsprechen minimieren [NRP05].

- **divisive Clusterverfahren (hierarchisches Verfahren):** Wie bei der Vorstellung der prinzipiellen Clusterverfahren beschrieben, wird beim divisiven Clustering zunächst mit einem Cluster begonnen, das sämtliche Datenobjekte beinhaltet. In diesem wird nun nach genau dem Datenobjekt gesucht, bei dem die Ähnlichkeit zu den anderen Datenobjekten im Cluster minimal, also die mittlere Distanz maximal ist. In jeder Iteration wird um das so ermittelte Zentrum ein neues Cluster aus den Datenobjekten gebildet, die eine geringe Distanz zum jeweiligen Clusterzentrum aufweisen. Die Iteration läuft genau so lange, bis jedes Cluster nur noch ein Datenobjekt enthält.

- **agglomerative Clusterverfahren (hierarchisches Verfahren):** Dieses ist analog zu den *divisiven Clusterverfahren*, allerdings umgekehrt. Zunächst ist jedes Datenobjekt zugleich ein Cluster. Die Cluster mit einer geringen Distanz werden so lange iterativ zusammengefasst, bis alle Datenobjekte in einem Cluster vorhanden sind. In einem Alternativansatz werden die Cluster zusammengefasst, deren *totale Varianz* (Streuung) die geringste Steigerung aufweist.

- **DBSCAN (dichtebasiertes Clusterverfahren):** Cluster werden hier als Areale behandelt, in denen Datenobjekte nah voneinander entfernt lokalisiert sind. Die Cluster werden getrennt durch Areale, in denen die Datenobjekte eine größere Entfernung aufweisen. Ein Cluster wird gebildet, sobald die Dichte der Datenobjekte einen definierten Schwellenwert überschreitet. Kernobjekte sind Datenobjekte, die innerhalb eines definierten Abstandes mindestens k Nachbardatenobjekte besitzen. Randobjekte sind Datenobjekte, die einem Cluster nahe sind und werden in dieses eingeschlossen, alle anderen Datenobjekte werden als *Rauschen* nicht berücksichtigt [JA03].

Klassifikations- und Regressionsalgorithmen

Die Klassifikations- und Ressionsalgorithmen haben zum Ziel, die Daten in Klassen einzuteilen (Vergleich Abbildung 2.1). Beides sind gebräuchliche Formen des überwachten Lernens (supervised Learning) [JWS90]. Das Training findet also anhand von Beispielen statt, bei denen die Einordnung der Daten in Klassen bereits manuell vorgenommen wurde. Der Unterschied zwischen Klassifikation und Regression liegt in erster Linie in der Art der Variablen, die bestimmt wird. Bei der Regression werden kontinuierliche Werte für die Variablen bestimmt, bei der Klassifikation werden dagegen diskrete Werte erwartet.

Grundsätzlich existieren zwei Arten der Klassifikation (Vergleich [JC14]). Die *instanzenbasierten* Klassifikationsverfahren führen eine direkte Klassifizierung auf Grund von Beispieldaten auf den noch zu klassifizierenden Daten durch. Diese Verfahren sind verhältnismäßig einfach aufgebaut, da ein zu untersuchendes Datenobjekt mit den vorhandenen Testobjekten anhand einer Distanzfunktion verglichen und der Klasse mit der größten Ähnlichkeit zugeordnet wird.

Im Gegensatz dazu erstellt das *modellbasierte* Klassifikationsverfahren aus den zur Verfügung gestellten Testdaten ein Modell als Metaebene. Die Testdaten werden nach Erstellung und Verifizierung des Modells für die Durchführung des Klassifikationsprozesses nicht mehr benötigt.

Im folgenden Abschnitt werden einige wichtige Klassifikations- und Regressionsverfahren vorgestellt, die auch Teil der Implementierungen der Machine-Learning-Bibliotheken MLlib und H2O sind.

- **Linear Regression:** Bei den Linear-Regression-Algorithmen handelt es sich um eine der gebräuchlichsten Regressionsmethoden. Hier wird eine statistisch abhängige Variable durch 1 bis n unabhängige Variablen beschrieben [DML15]. Die Regressionskoeffizienten werden in erster Potenz im Modell berücksichtigt. Deshalb wird dieses Verfahren als *Linear Regression* bezeichnet.

  Zusammengefasst durch die *Simple Linear Regression* lässt sich das Verfahren wie folgt darstellen [Enz15]:

  Die Ausprägung einer Variablen ist bekannt, es soll die Ausprägung einer anderen Variablen vorhergesagt werden. Die vorherzusagende Variable (abhängige Variable) ist das *Kriterium*. Die zur Vorhersage genutzte Variable (unabhängige Variable) ist der *Prädiktor*. Ziel der Regression ist nun, die lineare Funktion zu finden, welche die Abhängigkeit zwischen Pädiktor und Kriterium beschreibt.

  Als einfachste Vorhersagefunktion wird hier ein lineares Gleichungssytem betrachtet. Hier werden die Werte von Y geschätzt, indem die Werte von X linear transformiert werden:

  $\hat{y} = a_{yx} + b_{yx} * x$

  mit $\hat{y} =$ Schätzwert des Kriteriums
  $a_{yx} =$ Schätzwert des Kriteriums wenn der Prädiktor den Wert 0 hat (*Intercept*)

b_{yx} = Regressionskoeffizient, Steigung der Linie zum vorhergesagten Wert des Kriteriums (*Slope*)
x = Wert des Kriteriums zur Schätzung des Prädiktors.

So kann durch die Koeffizienten die Änderung in den abhängigen Variablen durch die Änderung um eine Einheit in der unabhängigen Variablen vorausgesagt werden.

Nun wird noch ein Maß für die Schätzfehler benötigt. Hierzu wird bei den *Linear-Regression-Algorithmen* auf die *Least-Squares* zurückgegriffen, also die summierten, quadrierten Abweichungen der tatsächlichen von den vorhergesagten Werten. *Intercept* und *Slope* werden so bestimmt, dass die *Least-Squares* minimal sind.

$\sum(y - \hat{y})^2 = min$ (*Ordinary Least Squares - OLS*)

So wird die Linie für das Modell bestimmt, die aufgrund der geringen Schätzfehlers am Besten passt.

Im Gegensatz zu den *Ordinary Least Squares* kommen insbesondere im Kontext der Bilbliotheken von MLLibs und H2O auch die *Alternating Least Squares Methoden* (ALS) zum Einsatz. Diese Methode basiert auf dem Grundsatz, dass eine algebraische Lösung einer Optimierung erheblich vereinfacht wird, wenn die Hälfte der Funktionsparameter in einem Optimierungsschritt unverändert verbleibt [YZ09]. Bei den ALS werden also in den Optimierungsläufen alternierend je eine Hälfte der Attribute angepasst, die andere Hälfte wird nicht verändert [TH14].

Die Funktion für einen *Alternating Least Squares Algorithmus* kann sich folgendermaßen darstellen lassen [Dat15]:

$f[i] = argmin \sum(r_{ij} - w^T f[j])^2 + \lambda \parallel w \parallel_2^2$ mit $w \in \mathbb{R}^d$ und $j \in Nbrs(i)$

Jedoch existieren auch eine ganze Reihe von verfeinerten und veränderten ALS-Methoden, welche die Modellleistung teils erheblich steigern können oder hier zumindest vielversprechende Ansätze liefern.

Die Least-Squares-Algorithmen zählen zu den *kollaborativen Filtermethoden*. Diese finden unter Anderem in Empfehlungssystemen Verwendung.

- **Logisitic Regression:** Die *Logistic Regression* ist in Ihrem prinzipiellen Zweck der *Linear Regression* sehr ähnlich. Auch hier wird die Beziehung zwischen einer oder mehreren unabhängigen und einer abhängigen Variablen modelliert. Im Gegensatz zur *Linear Regression* wird bei der *Logistic Regression* jedoch die Eintrittswahrscheinlichkeit eines Ereignisses berechnet. Somit wird also kein linearer Zusammenhang zwischen den abhängigen und den unabhängigen Variablen hergestellt, in dem die abhängige Variable einen präzisen numerischen Wert erhält. Statt dessen wird mittels der *Logistic Regression* ein Wahrscheinlichkeitswert zwischen 0 und 1 erwartet [BG15]. Die entsprechende Funktion stellt sich folgendermaßen dar:

$P = \frac{e^{\alpha+\beta x}}{1+e^{\alpha+\beta x}}$

mit P als Eintrittswahrscheinlichkeit für das Ereignis
α, β als Koeffizienten des Modells (Vergleich *Linear Regression*), die von x abhängig sind. Wenn x den Wert 0 hat, erreicht α den Wert von P. β zeigt die Wahrscheinlichkeit, dass sich eine 1 ändert, wenn x sich um eine Einheit ändert.

Ein weiterer Bestandteil der Logistic Regression sind die *Odds*, also die Verhältnisse der Wahrscheinlichkeit zu ihrer jeweiligen Gegenwahrscheinlichkeit. Diese werden im Logistic-Regression-Alogrithmus als *Logit* (Logarithmus eines Odds) bezeichnet und stellen hier die abhängige Variable dar. Ein Logit ist folgendermaßen definiert:

$logit(p) = ln(\frac{P}{1-P})$

- **Support Vector Machines:** Eine *Support Vector Machine* (SVM) ist ein Verfahren zur Klassifikation und Regression. Ziel ist, eine Menge von zu kassifizierenden Datenobjekten derart aufzuteilen, dass um die Trennlinie herum ein möglichst breiter Korridor frei von Datenobjekten bleibt. Einen solchen Korridor nennt man deshalb *Large Margin Classifier* [Mar15].

  Um die Klassen voneinander trennen zu können, wird eine sogenannte Hyper Plane (Hyperebene) konstruiert. diese ist definiert durch den Normalenvektor w und die Verschiebung b. Die Definition der Hyperebene ist somit:

  $H = x \mid \langle w, x \rangle + b = 0$

  Support Vectors sind die Punkte, die zu der generierten *Hyper Plane* den geringsten Abstand aufweisen. Diese Stützungsvektoren haben exklusiv Einfluss auf die Lage der *Hyper Plane*, Punkte die nicht als Stützungsvektoren definiert wurden, sind für deren Lage irrelevant.

  Die Vorhersage der Klasse für ein neues Objekt findet statt, indem zunächst für einen Trainingslauf w und b so gewählt werden, dass die Hyper Pane die Trainingsdatensätze trennt. Im Vorhersageschritt wird nun untersucht, auf welcher Seite des Hyper Pane das Objekt liegt. Objekte, die in Richtung des Normalenvektors der Hyper Pane liegen, werden als positiv klassifiziert, alle anderen als negativ. Fehler, also Daten, die sich nicht trennen lassen, lassen den Abstand zur Hyperebene mit dem Fehlergewicht C multiplizieren [TH09].

  Der optimale Stützvektor wird konstruiert, indem die quadratische Norm $\| w \|$ minimal wird unter der Bedingung, dass für alle $i = 1...m$ gilt:

  $k_i * (w^T * x_i + b) \geq 1 - C_i$ mit $1 \leq i \leq m$, wobei C hier als Fehlergewicht eine positive Schlupfvariable[14] darstellt.

  Mit dem sogenannten Kernel-Verfahren wird aus einer Trennung, deren Realisierung im zweidimensionalen Raum komplex ist, eine dreidimensionale Darstellung. Eine Fläche besitzt umfangreichere Trennmöglichkeiten, als eine Gerade. Die Kernel-Funktion dient

[14] Schlupfvariablen werden für die für die Lösung eines Problems eingeführt, ihr Wert ist aber nicht von Interesse. Die eingeführte Schlupfvariable führt ein Problem auf ein einfacheres Problem zurück.

als Maß der Ähnlichkeit. Somit muss keine tatsächliche Transformation in die dritte Dimension vollzogen werden.

- **Naive Bayes:** Hierbei handelt es sich um einen wahrscheinlichkeitsbasierten Algorithmus, dessen Zweck die Vorhersage der Klasse mit der höchsten Wahrscheinlichkeit ist. Da es sich um ein instanzenbasiertes Klassifikationsverfahren handelt, ist hier keine Modellentwicklung durch die Trainingsdaten nötig. Beim Naive-Bayes-Verfahren wird von der Annahme ausgegangen, dass sämtliche Datenattribute untereinander statistisch unabhängig sind. Dieser Algorithmus basiert auf der Bayesschen Formel (Vergleich: [Mit15]):
  $P(X \mid Y) = \frac{P(Y|X)*P(X)}{P(Y)}$
  Laut [DJF15] lässt sich die Naive-Bayes-Klassifikation für mittlere bis große Trainingsmengen für Diagnose oder Klassifikation für Dokumente wie folgt anwenden:

  Das Ziel besteht darin, jede Instanz X in der Funktion $f : X \to V$ durch die Attributwerte $[a_1, a_2...a_n]$ zu beschreiben. Hierzu wird der wahrscheinlichste Wert von $f(x)$ ermittelt durch:
  $f(x) = argmax P(a_1, a_2...a_n) \mid v_i) P(v_i)$ mit $v_i \in V$
  Beim Naive-Bayes-Algorithmus wird folgende Annahme getroffen:
  $f(x) = P(a_1, a_2...a_n) \mid v_i) = \prod_i P(a_i \mid v_i)$
  Somit wird die Naive-Bayes-Klassifikation zu:
  $v_{NB} = argmax P(v_i) \prod_i P(a_i \mid v_i)$ mit $v_i \in V$

  Also wird für jeden der Klassifikationswerte v_i die Wahrscheinlichkeit $P(v_i)$ berechnet und für jeden Attributwert a_i die von v_i abhängige Wahrscheinlichkeit $P(a_i \mid v_i)$.

- **Decision Trees:** Ein Decision Tree (Entscheidungsbaum) ist ein Klassifikationsverfahren, das als hierarchischer Baum für den Merkmalsraum X und den Klassenraum K aufgebaut ist [AL15]. An jedem Knoten wird den Ästen jeweils eindeutig ein Element $x \in X$ zugeteilt. Jedes Blatt ist einer Klasse aus K zugeteilt. [ST15]

  Ein Entscheidungsbaum wird generiert, indem man aus einer Attributmenge A und einer Beipielmenge B ein Attribut $a \in A$ als Wurzel des Entscheidungsbaumes bestimmt. Wenn nun x_a die Menge aller Werte von a repräsentiert, so wird für jede Ausprägung dieses Attributs die Menge $B^x \subseteq B$ gebildet [JC14]. Für diese gilt:

  $\forall b \in B^x : x_a(b) = x$

  Danach wird eine so markierte Kante an die entsprechende Wurzel gelegt. Wenn $B^x = \varnothing$, wird die entsprechende Kante beendet. Wenn alle Elemente aus B in der gleichen Klasse sind, endet die Kante mit einem entsprechenden Blatt.

  Für ein Objekt wird eine Klasse vorhergesagt, indem der Baum von der Wurzel beginnend traversiert wird, jedem Knoten ein Ast angehängt wird, der die Zuordnung des Objekts x erhält und schließlich bei den erreichten Blättern die jeweilige Klasse abgelesen wird. Die Anzahl der Blätter entspricht der Anzahl von Entscheidungsregeln.

Entscheidungsbäume können als Klassifikationsverfahren in Betracht gezogen werden, wenn die Instanzen durch Attribut-Wertpaare beschreibbar sind, die Zielfunktion einen diskreten Wert besitzt, *disjunkte Hypothesen* zur Klassifikation erstellt werden sollen, oder wenn die Trainingsdaten möglicherweise starkes Rauschen aufweisen. *Decision Trees* neigen leicht zu einem *Overfitting* der Testdaten.

- **Random Forests:** Bei den *Random Forests* handelt es sich um eine Form der *Ensemble-Learning-Methoden* [LB15]. Diese bauen während des Trainings eine Menge unterschiedlicher *Decision Trees* auf, um aufgrund verschiedener Teile der Trainingsdaten Durchschnitte auf verschieden großen Decision Trees zu bilden. Ziel ist, die Varianz zu reduzieren und damit die Modellleistung erheblich zu steigern. Allerdings steigen gegenüber einfachen *Decision Trees* der *Bias*[15] und durch gesteigerte Komplexität sind diese oft nicht mehr einfach interpretierbar.

2.4 Das MapReduce-Paradigma

Bei *MapReduce* handelt es sich um ein Programmierparadigma, das von *Google, Inc.* entwickelt wurde, um große Datensätze mittels eines auf einem Cluster verteilten und nebenläufig ausführbaren Algorithmus analysieren und verarbeiten zu können [JD04]. Dieses Paradigma eignet sich sowohl für strukturierte, als auch unstrukturierte Daten, Voraussetzung für die erfolgreiche Verarbeitung ist allerdings, dass die *Tasks* einen hohen Parallelisierbarkeitsgrad aufweisen (Vergleich [JD04]).

MapReduce nimmt einen Satz von Eingabe Key/Value-Pairs entgegen und produziert daraus einen Satz von Ausgabe *Key/Value-Pairs*. Das Paradigma besteht aus drei jeweils parallelisierbaren Einzelschritten, map, shuffle und reduce von denen map und reduce durch den Anwender selbst definiert werden müssen [AP09].

Diese Schritte von MapReduce in der Übersicht:

- **Map:** Hier wird aus den Eingabedaten mittels durchgehend gleicher Berechnungen ein temporär relevantes *Key/Value-Pair* erzeugt. Für jedes Wertepaar aus der Eingabeliste wird die *Map-Funktion* separat und unabhängig aufgerufen. Die Map-Funktion besteht aus Filter- und Sortierregeln. Die Ausführung erfolgt nebenläufig und verteilt.

- **Shuffle:** In diesem Schritt werden die Ergebnisse vor dem Aufruf der *Reduce-Funktion* nach ihrem duch *Map* erzeugten neuen Schlüssel in *Collections* gruppiert. Da in diesem Schritt bei verteilten Systemen und großen Datenmengen gegebenenfalls extreme Last im Netzwerk erzeugt werden kann, kommt hier oft vorher noch ein *Combine-Schritt*

[15] Ein Bias (auch Trend) bezeichnet einen systematischen Störeffekt, der bei Messungen auftreten kann und eine steigende oder fallende Tendenz aufweist [LB15].

zur Anwendung. Dieser sorgt mittels einer Vorreduktion dafür, dass die Datenmenge, die beim Shuffle über das Netzwerk transferiert wird, bereits auf den jeweiligen Nodes möglichst stark reduziert wird.

- **Reduce:** Auf jede im *Shuffle-Schritt* erzeugten *Collection* wird hier eine beliebige, durch den Anwender in Form eines Clojure definierte Funktion ausgeführt. Ziel dieser Funktion ist die Reduktion der vorhandenen Datensätze. Diese Funktion erstellt eine Ausgabe-Collection mit den Ergebnissen und ist ebenfalls je Collection nebenläufig und verteilt anwendbar.

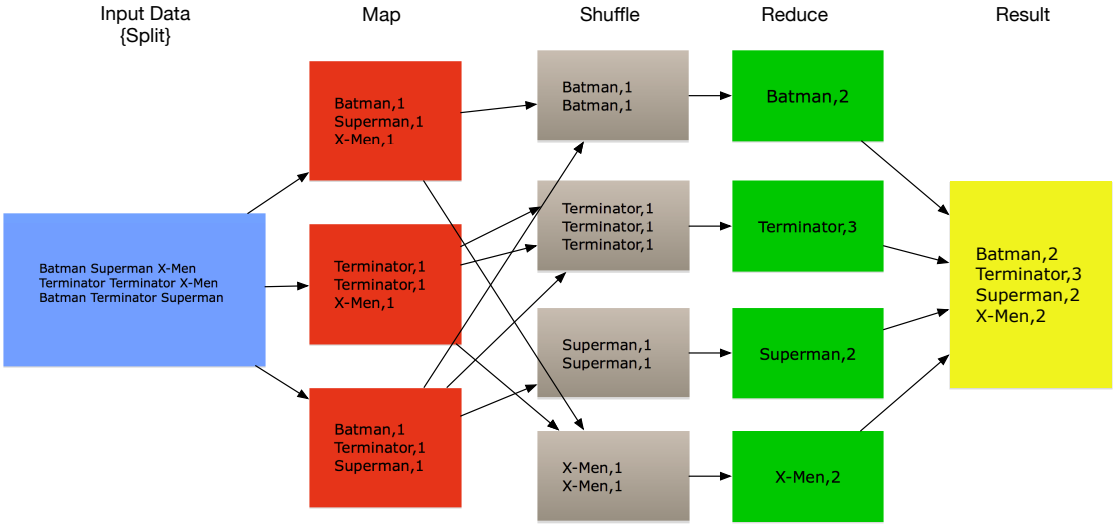

Abbildung 2.3: Ein konkretes Word-Count-Beispiel für MapReduce

In Abbildung 2.3 wird das *MapReduce-Modell* an einem praktischen Beispiel zur Wortzählung gezeigt. Bei *MapReduce* wird zunächst eine problemspezifische *Map-Funktion* definiert. Diese teilt die Ursprungsmenge mittels einer internen *Split-Funktion* in gleich große Partitionen und versieht unstrukturierte Werte in einem ersten Schritt mit Standardwerten, um so aus jedem unstrukturierten Datensatz ein Key-Value-Pair zu generieren [Xia15]. Im gezeigten Beispiel wird hier jedem Wort die Zahl Eins als Key zugeordnet. Im nächsten Schritt werden diese so gewonnenen Zwischen-Key-Value-Paare in einem *Shuffling-Prozess* klassifiziert und die so homogenisierten Pakete auf die Knoten im Cluster verteilt. Nun enthält jede Ausführungseinheit nur noch gleiche Wörter. Im Reduce-Schritt schließlich, werden die gleichartigen Wörter zu Gesamtmengen zusammengefasst, addiert und im Endergebnis aggregiert.

An dem gezeigten Beispiel ist deutlich erkennbar, dass die einzelnen Ausführungsschritte jeweils parallelisiert und auf separate Knoten im Cluster verteilt werden können. Das Laufzeitsystem übernimmt die Details der Partitionierung der Eingabedaten, die Ressourcenverwaltung innerhalb des Clusters, das Behandeln von Fehlern und die Kommunikation zwischen den einzelnen Knoten.

2.5 Streaming Frameworks

asdfasdf

2.6 Anwendungen von Graphen

Graphen sind geeignet, um Beziehungsstrukturen innerhalb von Daten zu modellieren. Graphen können in Baum oder Netzform vorliegen und mittels verschiedener Strategien traversiert werden. Dies kann mittels Tiefensuche, Breitensuche oder anhand von priorisierten Wegen durchgeführt werden. Prinzipiell werden Daten und deren Beziehungen in Form von Kanten, Knoten und Blättern dargestellt. Den Kanten können Gewichte zugeordnet werden, um Relevanz zwischen Beziehungen oder Wegen umzusetzen.

2.7 Zusammenfassung

Blablba

Kapitel 3

Der Berkeley Data Analytics Stack (BDAS)

Rund um *Hadoop* beziehungsweise *Spark* wurde an der University of California in Berkeley ein ganzer Infrastruktur-Stack für Big-Data-Analytics aufgebaut, der BDAS. Im folgenden Kapitel wird dieser Stack und die Bibliotheken, aus dem dieser besteht, vorgestellt. Im ersten Unterkapitel wird zunächst ein kurzer Überblick über den gesamten Stack gegeben. In den darauffolgenden Unterkapiteln werden die einzelnen Bestandteile im Einzelnen oberflächlich betrachtet, um dem Leser einen Einblick in die Nutzungsmöglichkeiten zu bieten. Eine Detailbetrachtung der jeweiligen Grundlagen, der praktischen Anwendung, Messungen und Vergleichsbetrachtungen der Alternativimplementierungen folgen im weiteren Verlauf dieser Ausarbeitung. Das Kapitel schließt mit einer Zusammenfassung, die auf einen Blick die Zusammensetzung des BDAS rund um Apache Spark zeigt.

3.1 Die Schichten des BDAS

Im Folgenden wird der *Berkeley Data Analytics Stack (BDAS)* näher vorgestellt, der wie in der Einführung bereits erwähnt um Hadoop, bzw. Spark als Hauptbestandteile herum aufgebaut ist. Der BDAS wurde von den *AMPLabs* [1] von der University of California in Berkeley aufgrund von Forschungsergebnissen im Bereich der Analyse sehr großer Datenmengen ins Leben gerufen.

Der Einsatz des BDAS kann laut Vijay Agneeswaran [Agn14] dabei helfen, beispielsweise konkrete praktische Fragen wie die folgenden zu beantworten:

- Wie segmentiert man am besten eine Menge von Nutzern und kann herausfinden, welche Nutzersegmente an bestimmten Kampagnen interessiert sein könnten?

[1] kurz für „algorithms, machines and people“

- Wie kann man richtige Metriken für Nutzer-Engagement in Social-Media-Applikationen herausfinden?
- Wie kann ein Video-Streaming-Dienst für jeden Nutzer dynamisch ein optimales *Content-Delivery-Network (CDN)* [2] basierend auf Daten wie Bandbreite, Auslastung, Pufferrate, etc auswählen?

Prinzipiell sind die in der Einführung in Kürze beschriebenen Einschränkungen von Hadoop und die damit verbundene Motivation für Spark auch die Motivation für den BDAS. Besonders für Aufgaben, die iterative Datenzugriffe und -manipulationen erfordern, wie beispielsweise Machine-Learning Algorithmen oder interaktive Abfragen, ist Hadoop auf Grund seiner starken festspeicherabhängigkeit nur bedingt zu empfehlen. Für diese Aufgaben ist Spark mit seinen In-Memory-Primitives prädestiniert. Um Hadoop herum ist in den letzten Jahren ein umfangreiches Ökosystem von Bibliotheken und Frameworks gewachsen, so dass die meisten Aufgaben im Umfeld von Big-Data-Analytics mit einer Hadoop-Infrastruktur gelöst werden können. Dagegen spricht laut [Kun14], dass für jeden Nutzungsfall ein eigener, auf Hadoop basierender Technologiestack eingesetzt werden muss, große Expertise in mehreren Technologien für produktives Arbeiten nötig ist und die Architektur vor allem für Aufgaben, bei denen schneller Datenaustausch zwischen parallel bearbeiteten Tasks notwendig ist, unpassend ist.

Im Gegensatz dazu bietet der BDAS auf allen Ebenen Standard APIs für Java, Scala, Python und SQL an und mit MLLibs stehen etliche Machine-Learning-Alogrithmen direkt in Spark zur Verfügung. Der Stack obehalb von Spark ist flexibel konfigurierbar und sämtliche Bibliotheken lassen sich parallel betreiben.

In Abbildung 3.1 wird eine Übersicht über die Hauptschichten des BDAS gezeigt, welche die jeweils von den AMPLabs empfohlene Implementierung zeigt.

In der untersten Schicht befindet sich das Cluster-Management-System Mesos, darüber das verteilte Filesystem HDFS, die Caching-Schicht Tachyon und der eigentliche Spark Kernel. Auf diesem setzen die Bibliotheken MLLib für Machine-Learning, Spark Streaming für Streaming-Anwendungen, GraphX für Graphenanwendungen und für datenbankähnliche Abfragen auf dem BDAS Spark SQL auf.

In den folgenden Unterkapiteln werden die einzelnen Elemente des BDAS detailliert vorgestellt.

[2] Ein Content Delivery Network (auch Content Distribution Network) ist ein Netzwerk verteilter und über das World Wide Web verbundener Server, das den Zweck hat, grosse Dateien bereitzustellen und auszuliefern.

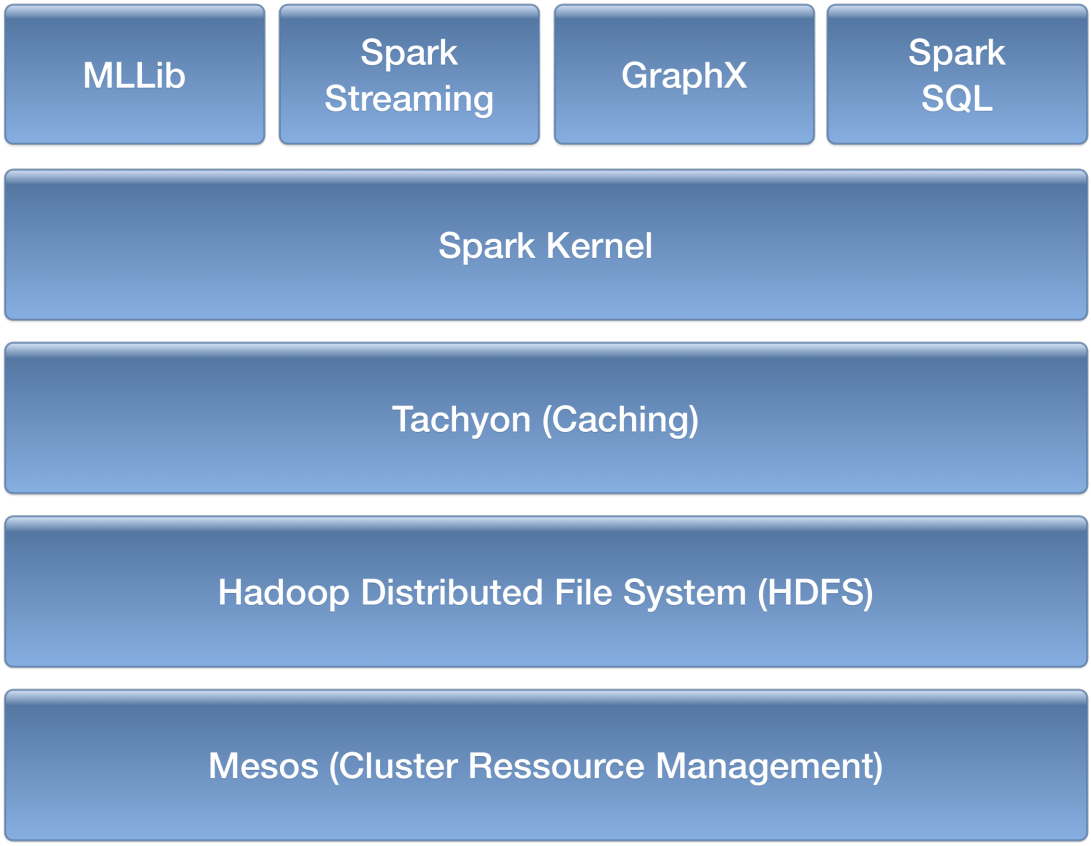

Abbildung 3.1: Übersicht des BDAS mit den vom AMPLab empfohlenen Bibliotheken.

3.2 Apache Mesos

Bei *Apache Mesos* handelt es sich um ein *Cluster-Management-Framework* für Anwendungen, die in verteilten Serverpools laufen sollen. Bestandteil von Mesos ist wiederum *Apache ZooKeeper*, das für Konfigurationsinformationen, Naming-Services und die Synchronisation von verteilten Anwendungen zuständig ist.

Mesos wird im BDAS eingesetzt, um die Prozesse von Hadoop/Spark effizient auf die einzelnen Knoten im Cluster zu verteilen. Besonders das Ressourcen-Management und –Monitoring innerhalb des Clusters ist ein wichtiger Faktor, um Jobs performant auf verteilten Systemen ausführen zu können. Auch das Fehlerhandling für Knoten, Prozesse und im Netzwerk wird im Berkeley-Stack von Mesos übernommen.

Ein besonderer Vorteil von Mesos gegenüber Yarn oder anderen Alternativen, wie dem Cloudera Cluster Manager oder Ambari von Hortonworks ist die Möglichkeit, verschiedene Frameworks gleichzeitig und isoliert in einem Cluster betreiben zu können. So kann beispielsweise Hadoop mit Spark in einer gemeinsamen Infrastruktur koexistieren.

3.3 Hadoop Distributed File System (HDFS) und Tachyon

Das Hadoop Distributed File System basiert ideologisch auf dem GoogleFileSystem (GFS) und hat zum Zweck, zuverlässig und fehlertolerant sehr große Dateien über verschiedene Maschinen hinweg in verteilten Umgebungen zu speichern. In entsprechenden Veröffentlichungen von Hortonworks [Hor14] wird von Produktivsystemen berichtet, die bis zu 200 PetaByte an Datenvolumen in einem Cluster von 4500 Servern basierend auf HDFS verwalten.

HDFS wurde speziell für den Einsatz mit MapReduce entwickelt, ist also auf geringe Datenbewegungen ausgelegt, da MR die Berechnungsprozesse jeweils zu den physischen Datensätzen selbst bringt und nicht, wie herkömmlich, die Daten zu den Prozessen geliefert werden müssen. So wird massiv Netzwerkverkehr innerhalb des Clusters eingespart und letztlich werden nur Prozesse und Prozessergebnisse verschickt.

Die Hauptbestandteile von HDFS sind der sogenannte NameNode, der die Metadaten des Clusters verwaltet und die DataNodes, die die eigentlichen Daten halten. Dateien und Verzeichnisse werden vom NameNode durch inodes repräsentiert. Diese wiederum enthalten Informationen über Zugriffsrechte, Zugriffszeiten oder Größenangaben der Dateien.

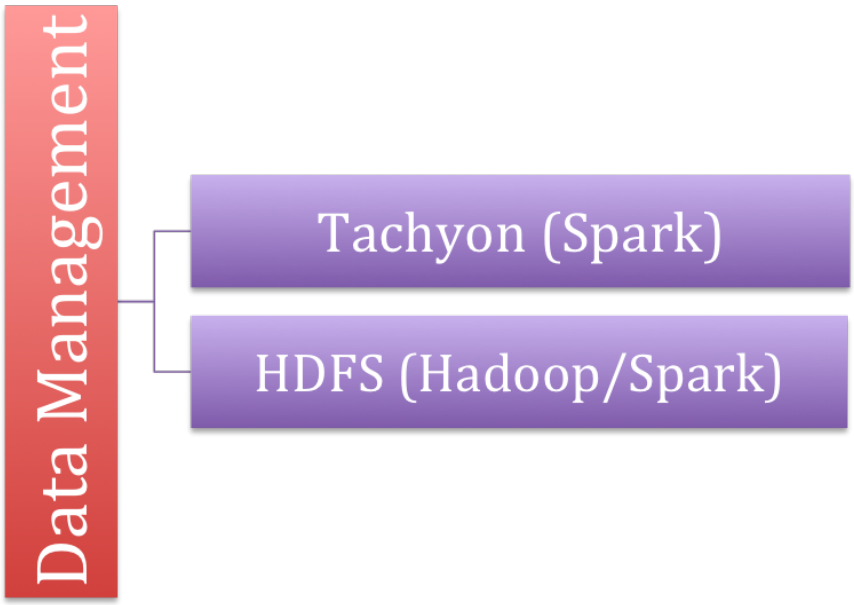

Abbildung 3.2: Der Datamanagement-Layer im BDAS mit HDFS und Tachyon

In Abbildung 3.2 wird die Datenmanagementschicht des BDAS nochmals detaillierter dargestellt. Hier ist erkennbar, dass reine Hadoop-Implementierungen direkt auf dem HDFS aufsetzen, da das HDFS für *MapReduce* optimiert ist.

Kommt hingegen Spark zum Einsatz, lässt sich wahlweise direkt das *HDFS* ansprechen oder alternativ eine Zwischenschicht nutzen, die auf das In-Memory-Modell von Spark zugeschnitten ist. Dies ist innerhalb des BDAS das verteilte Dateisystem Tachyon. Hier werden die zu verarbeitenden oder zu analysierenden Datensätze direkt in den Hauptspeicher des jeweiligen

Knoten in Form eines Cache gehalten. Somit werden Lade- und Speicheroperationen auf Massenspeicher minimiert und eine massiv höhere Ausführungsgeschwindigkeit erreicht. Unterhalb von Tachyon ist nach wie vor ein HDFS für die persistente Datenhaltung notwendig. Alternativ kann auch das Amazon S3-File-System eingesetzt werden. Tachyon wurde direkt innerhalb der AMPLabs entwickelt und ist mittlerweile fester Bestandteil des BDAS.

3.4 Apache Spark

Spark ist das Herzstück des BDAS. Bei Spark handelt es sich um ein open-source Data-Analytics-Framework, das, wie Hadoop, speziell für die Bedürfnisse im Rechner-Cluster konzipiert ist. Auch Spark nutzt das HDFS entweder direkt, oder indirekt über Tachyon. Im Gegensatz zu Hadoop bietet Spark jedoch Funktionen für In-Memory-Cluster-Berechnungen und ist nicht zwingend an MapReduce gebunden. Besonders interaktive Analyse oder Verarbeitung der Daten, Abfragen über verteilte Dateien und iterative

Lernalgorithmen erfahren so laut AMPLab eine bis zu hundertfache Ausführungs-geschwindigkeit im Gegensatz zu Hadoop. Auch die im ersten Kapitel angesprochenen Schwächen von Hadoop bei Berechnungen von komplexen linear-algebraischen Problemen, generalisierten n-Körper-Problemen, diversen Optimierungsproblemen und diversen anderen Aufgaben, treten bei Spark auf Grund der offenen Architektur und der Zerlegung von Datensätzen in die sogenannten Resilient Distributed Datasets (RDD) nicht mehr auf.

Spark wurde komplett in Scala entwickelt und bietet APIs für Scala, Java (inklusive Lambda-Expressions von Java 8) und Python. Im Labor existieren bereits Spark-Installationen mit bis zu 2000 Knoten, in Produktivsystemen sind bisher Systeme mit bis zu 1000 Knoten im Einsatz [Met14]. Durch die Möglichkeit, die Datensätze im Speicher für interaktive Analyseaufgaben zu cachen und iterativ abzufragen, ist eine direkte Kommandozeileninteraktion über das integrierte Scala REPL (alternativ auch in Python) möglich.

Für Spark existieren dedizierte Bibliotheken für Verarbeitung von Datenströmen, Machine-Learning und Graphenverarbeitung. Ähnliche Artefakte existieren auch für Hadoop (Mahout, Vowpal Wabbit, etc.), jedoch ist die Architektur von Spark wesentlich besser für derartige Anwendungsbereiche zugeschnitten.

3.5 Spark Streaming

Spark Streaming ist eine der oben genannten Bibliotheken, die Spark um dedizierte Anwendungsbereich erweitert. Hierbei handelt es sich um eine Erweiterung, um die integrierte API von Spark für Anwendungen auf Datenströmen nutzen zu können. Das Programmiermodell

unterscheidet nicht zwischen Batch- und Streaming-Anwendungen. So lassen sich beispielsweise Datenströme zur Laufzeit mit Archivdaten vergleichen und direkt Ad-hoc-Abfragen auf die Ströme formulieren. Im Fehlerfall ermöglicht Streaming zahlreiche Wiederherstellungsoptionen, sowohl von verlorenen Datenströmen, als auch von Einstellungen. Ein Anwendungsbeispiel ist die Echtzeitanalyse von Twitter-Meldungen.

3.6 GraphX

GraphX ist eine Erweiterung für Spark, die verteilte, flexible Graphen-Anwendungen in einem Spark-Cluster ermöglicht [Xin14]. Besonders in den Disziplinen „Machine Learning“ und „Data Mining“ ist die Anwendung komplexer Graphen unerlässlich. Graph-datenbanken kommen immer dann zum Einsatz, wenn stark vernetzte Informationen und ihre Beziehungen zueinander interessant sind. Hier werden die Entitäten als Knoten behandelt, die Beziehungsart definiert die Kanten. Die Kanten können auch gewichtet

sein. Ein konkretes Beispiel sind die Mitglieder eines sozialen Netzwerks mit ihrem jeweiligen Beziehungsgeflecht. Je nach Kontaktintensität können diese Beziehungen auch priorisiert werden, was hier dem Kantengewicht entspricht.

GraphX nutzt hier die Vorteile der darunterliegenden Spark-Infrastruktur, in dem durch eine tabellarische Anordnung der Datenstrukturen eine massive Parallelisierung möglich ist und auch der Verarbeitung in RDDs voll unterstützt wird. So sind auch interaktive Operationen auf den Graphen jederzeit über REPL möglich.

3.7 MLbase/MLLib

MLbase ist eine Sammlung von Bibliotheken und Werkzeugen für Machine-Learning-Anwendungen mit Spark. Sie besteht grundsätzlich aus den drei Teilen MLlib, MLI und ML-Optimizer und ist oberhalb der Spark-Installation angesiedelt, wie auf Abbildung 3.3 zu erkennen ist.

Die MLlib ist eine verteilte Machine-Learning-Bibliothek die für die Spark-Laufzeitumgebung entwickelt wurde und die bekannten Algorithmen für Probleme wie Klassifikation, Regression, Clustering und kollaboratives Filtern enthält.

Bei MLI handelt es sich um eine API, die es ermöglicht, selbst ML-Features zu entwickeln und in erster Linie für komplexere Problemstellungen geeignet ist. Mit MLI lassen sich die Funktionen direkt gegen Spark entwickeln, gegebenenfalls unter Zuhilfenahme der Bibliotheken der MLlib.

Der ML-Optimizer soll ML-Probleme für Endnutzer vereinfachen, in dem Modellauswahlen automatisiert werden. Hierzu werden Features aus der MLlib und der MLI extrahiert und zur Hilfe genommen.

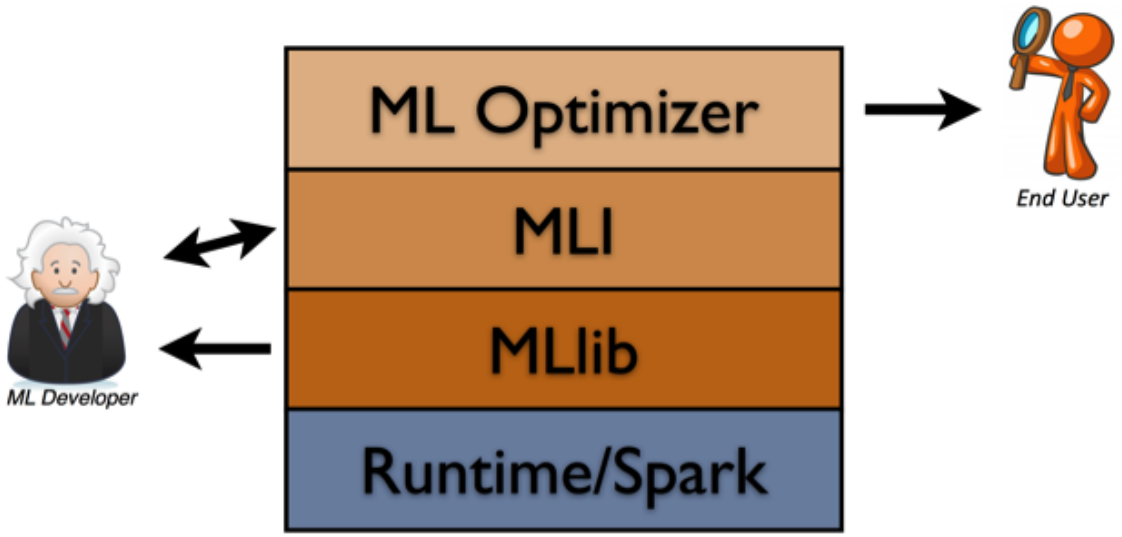

Abbildung 3.3: Die Bestandteile der MLbase [Lou14]

3.8 Spark SQL

Im Ökosystem von Hadoop ist Hive *Hive* ist eine *SQL-Query-Engine* , die sich großer Beliebtheit in der Community erfreut und trotz zahlreich Spark SQL[3] ist eine Portierung dieser Engine für Spark, um alle Vorteile der BDAS-Architektur nutzen zu können und ist kompatibel mit sämtlichen Hive-Daten, -Metastores und –Queries. Im Gegensatz zu Hive, das aus Datensätzen zur Laufzeit Java-Objekte generiert, nutzt Spark SQL eine zeilenorientierte Speicherung mittels Arrays primitiver Datentypen und ist somit selbst in einer Hadoop-Infrastruktur im Mittel bis zu fünfmal schneller als Hive.

Eine Besonderheit von Spark SQL ist neben seinem SQL-Interface die Möglichkeit, auch Machine-Learning-Funktionen als Abfragen formulieren zu können.

Für die Anwendung von Spark SQL hat sich die Architektur von Spark mit seinen RDDs als sehr vorteilhaft erwiesen, da Abfragen auf Fehlerhaften RDDs nach dem Neuaufbau des entsprechenden Datasets direkt erneut ausgeführt werden können.

Ein weiterer Unterschied zu Hive ist die sogenannte Partial-DAG-Execution (PDE). Dies bedeutet, dass logische Abfragepläne in Spark SQL aufgrund gesammelter Statistiken zur Laufzeit flexibel erstellt werden im Gegensatz zu Hive oder herkömmlichen relationalen Datenbanksystemen, wo bereits zur Kompilierungszeit starre physische Abfragepläne generiert werden. Besonders die Machine-Learning- und Failover-Funktionen wären mit einer Planerstellung zu Kompilierzeit nicht umsetzbar.

[3] Ursprünglich war die für den BDAS-Stack empfohlene Implementierung einer SQL-Query-Engine unter dem Namen *Shark* bekannt. Im Juli 2014 wurde jedoch bekanntgegeben, dass die Entwicklung von Shark zugunsten von Spark SQL eingestellt wurde und die vorhandenen Shark-Implementierungen voll in Spark SQL integriert werden. Deshalb zeigen Schaubilder des BDAS von vor Juli 2014 Shark als Query-Engine.

3.8.1 Zusammenfassung

Kapitel 4

Alternative Implementierungen der Bibliotheken und Frameworks des BDAS

Wie in den Abbildungen 3.1 und 6.1 ersichtlich ist, existieren auf jeder Ebene des BDAS auch alternative Implementierungen. Einige davon werden im Folgenden kurz vorgestellt.

4.1 Alternative zu Spark: Apache Flink

lkjasdlfkjsaödlkfj aöksdjfölksajdfl äöaskdjfölaskdfj

4.2 Alternative zu Spark Streaming: Storm

Storm ist, wie Apache Streaming, ein Framework für Hadoop, bzw. Spark für verteilte Streaming-Anwendungen. Wo Spark ganz klar eine Verbesserung gegenüber Hadoop darstellt und Shark dementsprechend für Hive, ist die Situation bei Storm und Apache Streaming dagegen nicht so klar determinierbar.

Storm und Spark Streaming unterscheiden sich fundamental in ihren Verarbeitungsmodellen [Agn14]. Das erstgenannte Framework verarbeitet eintreffende Events nacheinander, immer genau eines pro Zeitraum. Spark Streaming sammelt im Gegensatz dazu die Events in Mini-Batch-Jobs und verarbeitet sie paketweise zu definierten Zeiträumen nach wenigen Sekunden. Deshalb kann Storm Latenzzeiten von deutlich unter einer Sekunde erreichen, während Spark Streaming eine Latenzzeit von einigen Sekunden aufweist. Diesen Nachteil macht Spark Streaming durch eine sehr gute Fehlertoleranz wett, da die Mini-Batches nach aufgetretenen Fehlern

einfach nochmals bearbeitet werden können und die zuvor fehlerhaft ausgeführte Verarbeitung verworfen wird. Treten hingegen bei Storm Fehler auf, wird genau dieser Datensatz nochmals verarbeitet. Dies bedeutet, dass dieser auch mehrfach verarbeitet werden kann. Durch dieses Verhalten lassen sich die beiden Frameworks grob in zwei Einsatzgebiete verteilen:

Storm ist das Framework der Wahl, wenn Wert auf sehr kurze Latenzzeiten gelegt werden muss, hingegen ist es für statusbehaftete Anwendungen durch die Möglichkeit der Mehrfachverarbeitung ungeeignet. Im Umkehrschluss ist Spark Streaming eine gute Wahl, wenn aufgrund der gestreamten Daten eine Statusmaschine aufgebaut werden soll. Dafür müssen hier höhere Latenzzeiten in Kauf genommen werden.

4.3 Alternative zu MLLibs: H2O - Sparkling Water

BluBlaBlubb

4.4 Alternative zu MLLibs: Dato GraphLab Create™

BluBlaBlubb

4.4.1 Zusammenfassung

Kapitel 5

Funktionsweise von Spark

Im vohergehenden Kapitel wurde der Berkeley Data Analytics Stack vorgestellt. Es wurde gezeigt, dass dieser aus einer Reihe von Bibliotheken, Infrastrukturkomponenten und dem eigentlich Kern, Apache Spark, besteht.

In diesem Kapitel werden die grundlegenden Konzepte von Spark vorgestellt und dessen Funktionsweise betrachtet. Einleitend wird gezeigt, wie eine Spark-Infrastruktur aufgebaut sein kann, wie diese intern Abfragen und eigene Spark-Programme verarbeitet und wie der *Spark-Context* sich als Cluster-Repräsentant gegenüber dem Anwender und der API exponiert. Im nächsten Unterkapitel wird die eigentliche Basis von Apache Spark vorgestellt. Spark basiert im Wesentlichen auf einer verteilten Datenstruktur, den *Resilient Distributed Datasets*. Deren Konzept wird sowohl theoretisch, als auch im Anwendungskontext dargestellt.

Ein weiteres Kernelement der Spark-Implementierung bildet das *In-Memory-Processing* der Daten. Spark ist in der Lage, je nach Konfiguration des Host-Systems, große Teile der Analysen und Verarbeitungen äußerst flexibel im Hauptspeicher durchzuführen und so massive Performanceverbesserungen gegenüber festspeicherbasierter Verarbeitung zu generieren. Hierzu bietet Spark spezielle *In-Memory-Primitives* an. In einem weiteren Unterkapitel werden dieses detailliert vorgestellt.

5.1 Spark im Cluster

Eine große Herausforderung im Umfeld verteilter und nebenläufiger Analyse und Verarbeitung großer Datenmengen stellt der Netzwerkverkehr da. Der klassische Aufbau einer verteilten Anwendung hält die Daten auf einer dafür vorgesehen Plattform im Netzwerk. Häufig ist dies ein dedizierter File- oder Datenbankserver, der mit möglichst großer Bandbreite mit dem Applikationsserver verbunden ist (Vergleich Oracle InfinyBand).

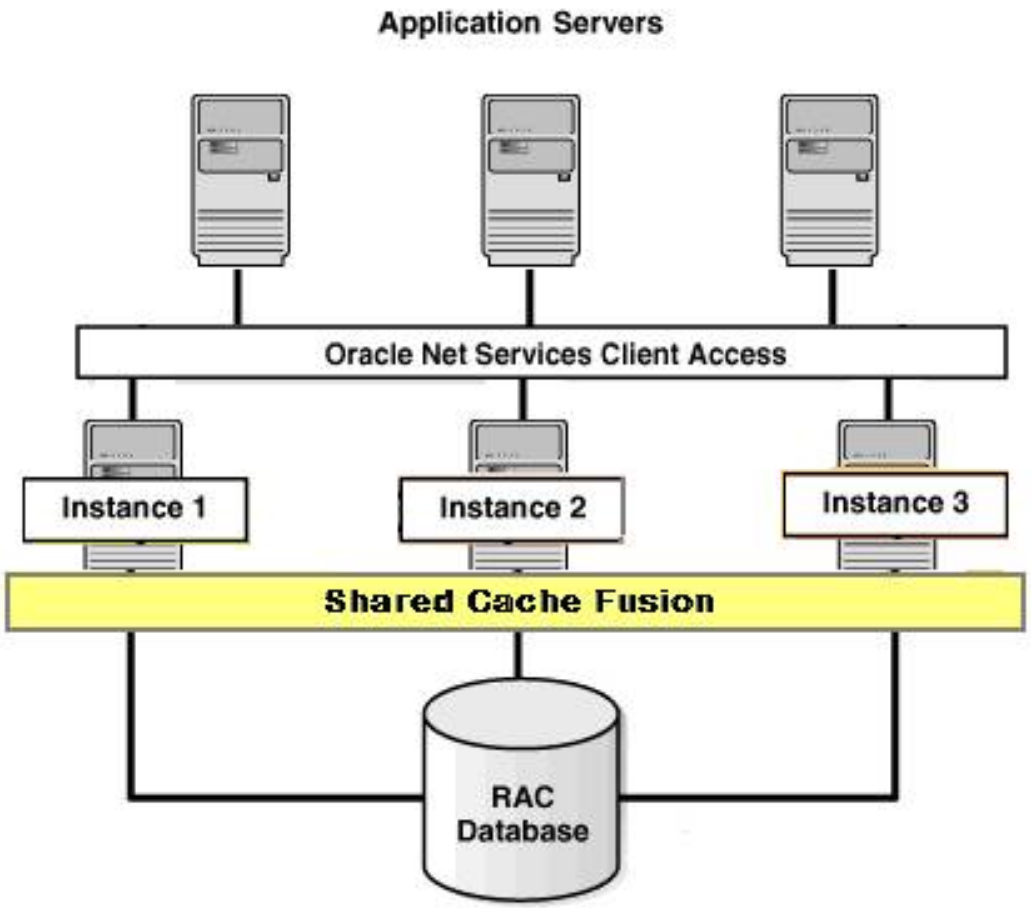

Abbildung 5.1: Aufbau eines Standardcluster im Rechenzentrumsbetrieb mit Application-Server und Oracle Datenbanken [Sto15].

In diesem Aufbau entsteht in der Regel eine sehr hohe Netzwerklast, da die für die Applikation benötigten Daten dieser zunächst zur Verfügung gestellt werden müssen.

Spark geht hier einen anderen Weg. Ein Spark-Cluster besteht typischerweise aus einem zentralen *Master* und n *Worker-Nodes*. Diese können aus einfachen Servern bestehen, aber auch aus Clustern von Großrechnern (beispielsweise IBM Z, Oracle Exa). Das Hadoop Distributed File System und Spark skalieren über Cluster beliebiger Größenordnung. Über ein verteiltes Dateisystem werden die Daten auf dem Cluster gehalten und sowohl dem *Master*, als auch den *Worker-Nodes* so zur Verfügung gestellt.

Spark-Anwendungen laufen als unabhängiges Set von Prozessen auf Cluster-Infrastrukturen. Das Hauptprogramm, der sogenannte *Spark Driver*, instanziert das *SparkContext-Objekt*, das die einzelnen Prozesse koordiniert. Auf Clustersystemen hält der *SparkContext* die Verbindung zum jeweiligen *Cluster-Ressource-Manager* (Mesos, Yarn), im Standalone-Betrieb instanziert der Context selbst einen Dummy-Manager und allokiert in beiden Fällen die für die Anwendung nötigen Hardware-Ressourcen. Die Cluster-Manager liefern ihren aktuellen Status an Spark

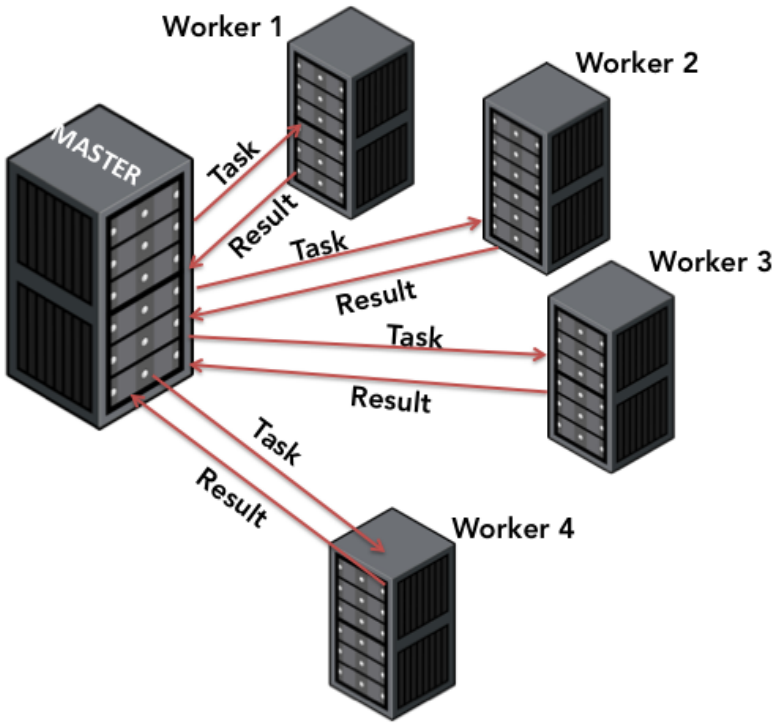

Abbildung 5.2: Clusteraufbau mit Spark mit einem Master und vier Worker-Nodes.

zurück und melden Auslastung und Gesundheitszustand der einzelnen Knoten. Über interne *Load-Balancing-Systeme*[1] wird ermittelt, welche Worker-Nodes die jeweiligen Tasks aus dem Spark-Kontext zugewiesen bekommen. Der SparkContext repräsentiert sowohl für die Spark-Konsole REPL, als auch in eigenen Spark-Programmen innerhalb der APIs das gesamte Cluster. Dem SparkContext wird bei der Initialisierung über ein Konfigurationsobjekt mitgeteilt, welche Ressourcen ihm für das aktuelle Programm zur Verfügung stehen. Die Entscheidung, welche, der initial zur Verfügung gestellten Nodes oder Ressourcen des Clusters von Spark wann in Anspruch genommen werden, obliegt der Kombination aus Spark und *Cluster-Ressource-Manager*.

Wenn ein SparkContext initialisiert wurde, installiert der Spark sogenannte *Executors* auf sämtlichen Worker-Nodes des Clusters. Der Applikationscode wird nun als JAR[2] direkt an die Executors verteilt und dieser anschließend durch entsprechende Tasks ausgeführt.

[1] Load-Balancing-Systeme sind Überwachungsmechanismen in verteilten Systemen. Jedes Teilsystem meldet seine eigene Auslastung und seine Verfügbarkeit an den Load-Balancer. Dieser verteilt anstehenden Aufgaben so auf die Ressourcen, dass eine möglichst gleichmäßige Verteilung über die gesamte Infrastruktur möglich ist.

[2] Ein JAR (Java ARchive) ist ein gepacktes und auf einer Java Virtual Machine ausführbares (Java, Scala, Clojure) Programmpaket, häufig inklusive der benötigten Bibliotheken.

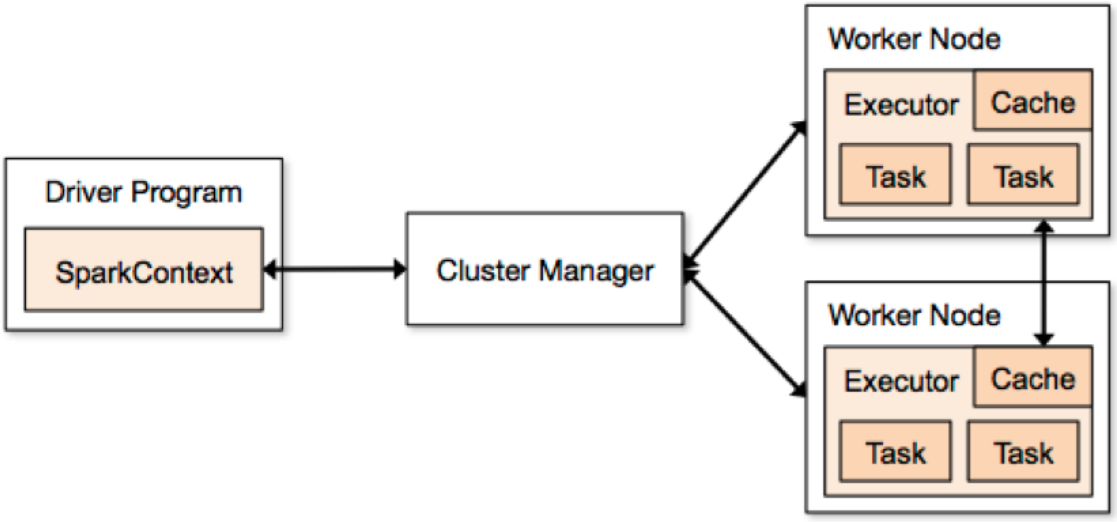

Abbildung 5.3: Clusteraufbau mit Spark [Apa14]

5.2 Das Konzept der Resilient Distributed Datasets

Die Resiliient Distributed Datasets (RDD) sind das eigentliche Kernelement von Apache Spark. Hierbei handelt es sich um fehlertolerante, parallele Datenstrukturen, die dem Anwender erlauben, Zwischenergebnisse explizit im Hauptspeicher zu halten, ihre Partitionierung zu steuern, um Daten bewusst an bestimmten Stellen halten zu können und diese mittels umfangreichen Operatoren zu manipulieren [MZ12]. Das Konzept der RDDs entspricht prinzipiell den Views[3] in relationalen Datenbanksystemen. Werden die RDDs für weitere Zugriffe persistiert, entspricht dies dem Prinzip der Materialized Views[4].

RDDs werden mittels deterministischer, paralleler Operationen erstellt, den sogenannten *Transformationen*. Im Fehlerfall, also wenn beispielsweise ein *Node* im Cluster ausfällt, können die RDDs automatisch an Hand der durchgeführten Regeln neu aufgebaut werden. Deshalb merken sich die RDDs die Transformationen, die zu ihrem Aufbau geführt haben und können so verlorene Datenstrukturen schnell rekonstruieren. Die *Resilient Distributed Datasets* können ausschließlich durch Transformationen, wie beispielsweise *map, filter, join*, etc., aus Daten aus dem Dateisystem oder aus anderen, bereits vorhandenen RDDs erzeugt werden, oder durch Verteilen einer Object-Collection in der Driver-Applikation. Diese Datenstrukturen müssen nicht persistiert werden, da sie für einen Partitionierung ausreichende Informationen über ihre Erstellungsregeln und die entsprechenden Datensätze enthalten, das sogenannte *Lineage* [MZ12].

Da es sich bei RDDs prinzipiell um Scala-Collections handelt, können diese auch direkt in Scala-Code eingebunden und verarbeitet werden, oder interaktiv über die Scala-Konsole REPL

[3] Eine View ist in einem relationalen Datenbanksystem die Ergebnismenge einer persistierten Datenbankabfrage auf bestimmte Daten. Diese lässt Abfragen analog zu einer gewöhnlichen Tabelle zu.

[4] Materialized Views entsprechen bei Abfragen den regulären Views. Allerdings wird hier im Gegensatz zu den Views bei Zugriff keine Abfrage auf die zugrundeliegenden Tabellen durchgeführt. Stattdessen sind die Daten als Kopie in eigenen Datenbankobjekten persistent vorhanden.

genutzt werden. Aber auch für Java und Python bietet Spark APIs an. Für weitere Sprachen wie R oder Clojure existieren Wrapper-Frameworks, welche die RDDs und ihre Operationen verfügbar machen (Vergleich 5.5. RDDs können nur durch grobgranulare, deterministische Transformationen, erstellt werden [Kun14].

WIe eingangs beschrieben, verfügen die Anwender von Spark über die Kontrolle der Aspekte *Persistence* und *Partitioning* [MZ12]. Mit der Persistence lässt sich festlegen, welche RDDs wiederverwendet werden sollen, also welche RDDs nach welcher Strategie persistiert werden. Dies ist besonders wichtig, da nicht explizit persistierte RDDs bei jeder darauf ausgeführten Operation neu berechnet werden. Das Partitioning legt fest, nach welchen Kriterien die RDDs geteilt und über das Cluster verteilt werden sollen, also beispielsweise sortiert nach bestimmten Keys. Die Optimierung der Partitionierung ist unter Anderem wichtig für Join-Operationen. Je nach Partitionierungsstrategie kann dies erhebliche Unterschiede in Laufzeit und Ressourcennutzung bedeuten [HK15].

RDDs können prinzipiell auf folgende Arten für die Wiederverwendung gespeichert werden [Apa14]:

- Als deserialisiertes Java-Objekt im Speicher der JVM – dieses Variante bietet die beste Performance, da die Objekte sich direkt im JVM-Heap befinden
- Als serialisiertes Java-Objekt direkt im Speicher – dieses Verfahren ist effizienter bezüglich der Hauptspeicherauslastung , aber schlechter in der Zugriffsgeschwindigkeit
- Im Dateisystem – diese Variante ist erwartungsgemäß die langsamste, jedoch nötig, wenn die RDDs zu groß für die Haltung im RAM sind.

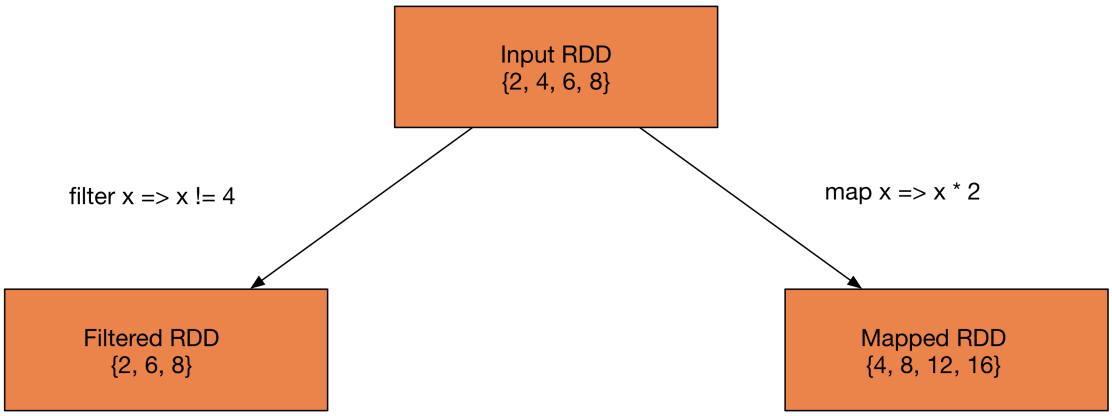

Abbildung 5.4: Transformation von RDDs in Spark.

In der folgenden Tabelle befindet sich exemplarisch eine Auswahl[5] der wichtigsten Transformationen auf RDDs (aus [Spa14]):

Transformation	Zweck
map(func)	Return a new distributed dataset formed by passing each element of the source through a function func.
filter(func)	Return a new dataset formed by selecting those elements of the source on which func returns true.
flatMap(func)	Similar to map, but each input item can be mapped to 0 or more output items (so func should return a Seq rather than a single item).
mapPartitions(func)	Similar to map, but runs separately on each partition (block) of the RDD, so func must be of type Iterator<T> => Iterator<U> when running on an RDD of type T.
intersection(otherDataset)	Return a new RDD that contains the intersection of elements in the source dataset and the argument.
distinct([numTasks]))	Return a new dataset that contains the distinct elements of the source dataset.

Tabelle 5.1: Übersicht der einiger wichtiger Transformationen auf RDDs

Transformationen werden auf RDDs grundsätzlich nach dem Prinzip der *Lazy Evaluation* durchgeführt (Vergleich [HK15]). Dies bedeutet, dass eine Ausführung der Transformation erst dann stattfindet, wenn die betreffenden Daten auch wirklich benötigt werden. Dies wurde in Spark so umgesetzt, um die Anzahl der Datentransfers, beispielsweise bei Gruppierungsaktionen, drastisch zu reduzieren. Diese Art der Ausführung birgt allerdings auch Problempotential bei einer etwaigen Fehlerlokalisierung, da häufig nicht transparent ersichtlich ist, ob die Evaluierung zum Zeitpunkt des Fehlerauftretens schon durchgeführt wurde.

Wie in Abbildung 5.2 dargestellt, werden die Daten bei einer Verarbeitung durch Spark zunächst aus dem HDFS geladen, in Resilient Distributed Datasets (RDD) transformiert, und dann im Hauptspeicher für Verarbeitung- oder Analysefunktionen zur Verfügung gestellt. Abfragen werden direkt entweder über eigene Programme, via Scala REPL oder SQL-artige Abfragen zur Laufzeit, über Batch-Jobs oder via Spark Streaming/Storm an die im RAM befindlichen RDDs geleitet.

[5] Die vollständige Übersicht über die Transformationen und Actions auf RDDs befindet sich im Anhang dieser Ausarbeitung.

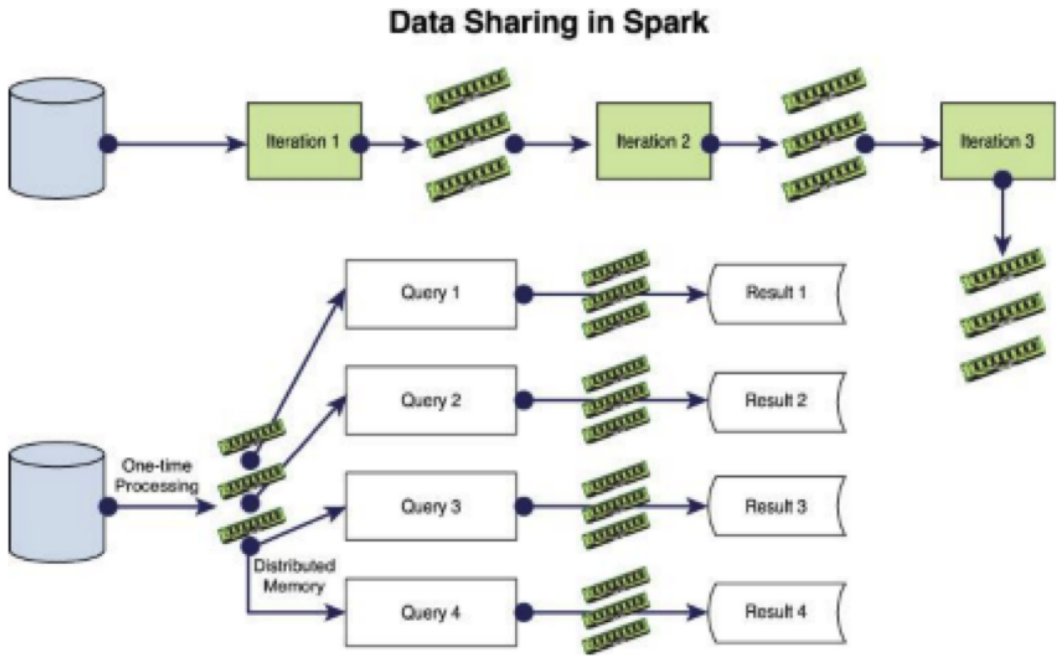

Abbildung 5.5: Schematische Darstellung der Funktionsweise von Spark [Agn14]

Actions	Zweck
reduce(func)	Aggregate the elements of the dataset using a function func (which takes two arguments and returns one). The function should be commutative and associative so that it can be computed correctly in parallel.
collect()	Return all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data.
count()	Return the number of elements in the dataset.
first()	Return the first element of the dataset (similar to take(1)).

Tabelle 5.2: Übersicht der einiger wichtiger Actions auf RDDs

5.3 Die In-Memory-Primitives von Spark

- MEMORY_ONLY Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, some partitions will not be cached and will be recomputed on the fly each time they're needed. This is the default level.
- MEMORY_AND_DISK Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, store the partitions that don't fit on disk, and read them from there when they're needed.

- MEMORY_ONLY_SER Store RDD as serialized Java objects (one byte array per partition). This is generally more space-efficient than deserialized objects, especially when using a fast serializer, but more CPU-intensive to read.
- MEMORY_AND_DISK_SER Similar to MEMORY_ONLY_SER, but spill partitions that don't fit in memory to disk instead of recomputing them on the fly each time they're needed.
- DISK_ONLY Store the RDD partitions only on disk.
- MEMORY_ONLY_2, MEMORY_AND_DISK_2, etc. Same as the levels above, but replicate each partition on two cluster nodes.
- OFF_HEAP (experimental) Store RDD in serialized format in Tachyon. Compared to MEMORY_ONLY_SER, OFF_HEAP reduces garbage collection overhead and allows executors to be smaller and to share a pool of memory, making it attractive in environments with large heaps or multiple concurrent applications. Furthermore, as the RDDs reside in Tachyon, the crash of an executor does not lead to losing the in-memory cache. In this mode, the memory in Tachyon is discardable. Thus, Tachyon does not attempt to reconstruct a block that it evicts from memory.

5.4 Die Spark-Console REPL

5.5 Die Spark APIs

5.5.1 Spark Scala API

5.5.2 Spark Java API

5.5.3 Spark Python API

5.5.4 Third Level API für Spark mit Clojure: Flambo

5.5.5 Third Level API für Spark mit R: SparkR

Kapitel 6

Architektur und Inbetriebhnahme von lokalen Apache Spark Infrastrukturen

Im folgenden Kapitel wird ein exemplarischer Architekturaufbau eines lokalen Entwicklungs- und Testsystems für Apache Spark vorgestellt. Wie in den vorangegangen Kapiteln gezeigt wurde, handelt es sich bei Spark in erster Linie um ein Framework für leistungsstarke Clustersysteme. Dennoch muss Spark auch auf entsprechend kleiner dimensionierten System betrieben werden können. Häufig ist in einem professionellen Umfeld beispielsweise nur ein Cluster für Produktionsaufgaben vorhanden, die Test- und vor allem auch die Entwicklersysteme stellen sich häufig in Form sogenannter *SIngle-Node-Cluster* dar.

Besonders der Aspekt der Entwicklungsarbeitsplätze rückt hier in den Vordergrund. Da Spark lediglich über den *Master-Node* und hier über den *Spark-Context* innerhalb des *Driver-Program* für die Entwickler zugreifbar ist und die interne Verteilung der Tasks auf die jeweiligen *Nodes* von Spark und den *Clustermanagement-Systemen* übernommen und maskiert wird, stellt sich die Fehleranalyse mittels klassischem *Debugging*[1] als sehr große Herausforderung dar. Problematisch ist hierbei in erster Linie, dass vor und während des Debugging-Prozesses zu keiner Zeit der aktuelle Ausführungs-Node im Cluster determiniert werden kann. Da auf jedem *Node* eine unabhängige *JVM*[2] instanziert ist, kann nicht zentral ermittelt und nachvollzogen werden, wo und zu welchem Ausführungszeitpunkt welches Verhalten auftritt. Für initiale Entwicklungstätigkeiten ist aus diesen Gründen immer eine lokale Instanz von Spark als Single-Node-Cluster nötig.

Zum Zweck von Versuchen bezüglich Skalierbarkeit im Cluster-Umfeld und Verhalten im verteilten Betrieb empfiehlt es sich, darüber hinaus eine lokale Cluster-Infrastruktur mit mindestens einem Master-und einem unabhängigen Worker-Node zu installieren. Dies kann auf

[1] Unter klassischem Debugging wird hier das Setzen von Breakpoints und das explizite Überwachen von Variablenwerten zur Laufzeit durch Entwickler verstanden.

[2] JVM = Java Virtual Machine. Hierbei handelt es sich um die Laufzeitumgebung von Java. Auch Scala-Code wird intern in Bytecode übersetzt und innerhalb der JVM ausgeführt.

verschiedene Arten stattfinden. Im Rahmen dieser Ausarbeitung wurden verschiedene Konfigurationen aufgebaut und miteinander verglichen. Diese Aufbauten mit ihren jeweiligen Stärken und Schwächen werden im folgenden Kapitel beschrieben.

Des weiteren wird die Installation und die grundsätzliche Anwendung der Bibliotheken rund um Spark, bzw. der Alternativimplementierungen gezeigt. Abschließend werden Empfehlungen für verschiedene Einsatzbereiche gegeben.

6.1 Prinzipieller Aufbau einer lokalen Spark Infrastruktur

Eine Spark-Infrastruktur besteht immer aus verschiedenen Schichten, welche im BDAS definiert sind und von denen einige zwingend notwendig sind, einige optional und andere durch Alternativimplementierungen ersetzt werden können. In Abbildung 6.1 ist der Aufbau des BDAS schematisch dargestellt. Die grün hinterlegten Elemente markieren die Bestandteile des aktuellen BDAS, die violett hinterlegten zeigen alternative Implementierungen auf der jeweiligen Schicht. Grün schraffiert ist die Applikationsschicht, in der Applikationen oberhalb von Spark und dessen direkten Bibliotheken angesiedelt sind.

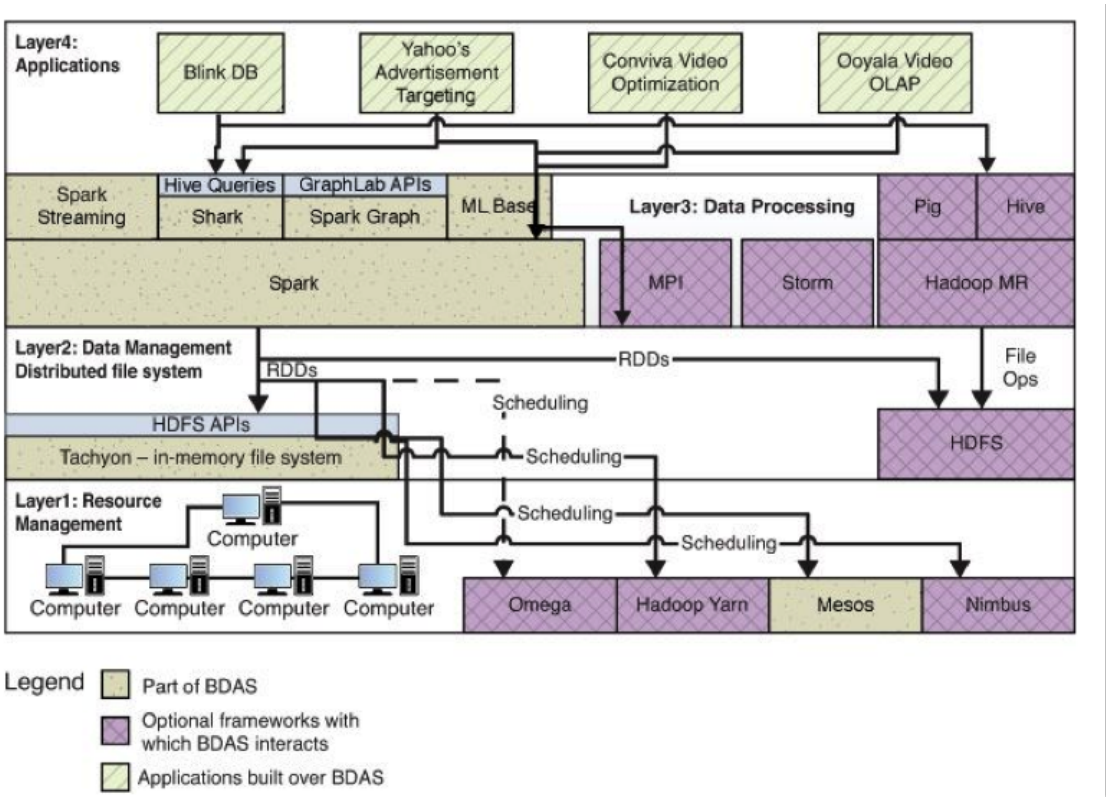

Abbildung 6.1: Der BDAS. Abbildung aus „Big Data Analytics Beyond Hadoop“, S 15 [Agn14]

Der BDAS beginnt in der untersten Schicht mit Mesos oder einer seiner Alternativen. Für eine lokale Installation von Spark wird allerdings in der Regel keine eigenes Cluster-Resource-

Management-System benötigt, da Spark die Grundfunktionalitäten für die Ressourcenverwaltung selbst im Kernel implementiert hat.

Prinzipiell ist zunächst zu entscheiden, ob Spark lokal als Single- oder Mulitnode-Cluster betrieben werden soll. Diese Entscheidung sollte auch maßgeblich von der zur Verfügung stehenden Hardware abhängig gemacht werden und von den Einsatzgebieten. Sind die Tasks gut parallelisierbar, bietet sich ein Cluster aus Master und $1...n$ Worker-Nodes an, ansonsten sollte den einzelnen Knoten oder einem Single-Node-Cluster möglichst viel Hauptspeicher zur Verfügung gestellt werden um die Festspeicherzugriffe zu minimieren.

Da Spark durch seine beiden Hauptmerkmale *In-Memory-Computation* und *massive Parallelverarbeitung* eine möglichst starke Hardwareinfrastruktur benötigt und hier besonders die Aspekte Hauptspeicher, Prozessorkerne, Knotenanzahl und Netzwerkperformance essentiell sind, sollte generell ist eine lokale Spark-Installation so einfach wie möglich aufgebaut werden. Der Fokus sollte hier auf Entwicklungstätigkeit, Debugging, und Vergleichsmessungen gelegt werden.

Für die lokale Installation besteht die Möglichkeit, Spark nativ auf dem System zu installieren, oder in virtualisierten Umgebungen. Eine native Installation hat den Vorteil, dass Spark eine systemnahe JVM als Ausführungsumgebung zum Einsatz kommt. So kann ein Großteil der vorhandenen Systemressourcen für die Spark-Ausführung zur Verfügung gestellt werden. Eine native Installation lässt sich entweder als Single-Node (Master-Only) konfigurieren, oder als virtuelles Cluster mit einem Master und einem Worker-Node. Hier empfiehlt sich ein Hybridbetrieb. Für Debugging-Aufgaben ist auf Grund der Taskverteilung von Spark nur ein Single-Node-Einsatz praktikabel. Für Skalierbarkeitstests bietet sich das Setup mit einem Master und einem Worker-Node an.

Als Alternativen zu dieser nativen Installation lässt sich ein theoretisch beliebig großes virtuelles Cluster auf einer lokalen Umgebung mit einer entsprechenden Anzahl von virtuellen Maschinen realisieren. Dies hat den Vorteil, dass die Skalierbarkeit der Anwendung durch entsprechende Partitionierung der RDDs besser testbar ist. Allerdings haben virtuelle Maschinen in der Regel einen sehr hohen Ressourcenverbrauch und lassen so nur bedingt Rückschlüsse auf das tatsächliche Laufzeitverhalten zu.

Eine weitere Möglichkeit des Setups ist das Deployment der kompletten Spark-Infrastruktur in einen Docker-Container. Diese Möglichkeit wird im folgenden Unterkapitel 6.2 beschrieben.

6.2 Ausführungscontainer: Docker

Bei Docker handelt es sich um eine Open-Source-Plattform zur Automatisierung des Software-Deployments innerhalb von Ausführungscontainern [Doc15]. Prinzipiell legt Docker eine weitere Abstraktionsschicht über ein vorhandenes Linux-System und nutzt dessen *Resource-Isolation-Features*, wie beispielsweise *cgroups*[3] [VN15].

Durch Docker werden isolierte Prozesse mittels *High-Level-API* in leichtgewichtigen Ausführungscontainern erzeugt. Der Vorteil gegenüber herkömmlichen Virtualisierungstechnologien, wie beispielsweise Oracle Virtual Box, Citrix XenServer, VMWare, besteht unter anderem darin, dass bei Docker im Gegensatz zu diesen Applikationen kein eigenes Betriebsystem in einer virtuellen Maschine installiert werden muss. Dadurch sind Docker-Container äußerst ressourcenschonend. Voraussetzung für einen Betrieb von Docker ist allerdings ein installiertes Linux-Betriebssystem auf dem Hostrechner. Docker bietet die Möglichkeit, relativ kleine Services in eigenen Containern unterzubringen. Die Prinzipien einer *Microservice-Infrastruktuktur*[4], allen voran die *Immutability*[5], können so mittels Docker-Containern umgesetzt werden (Vergleich [Doc15]).

Docker ist sehr gut geeignet, eine lokale Apache Spark Infrastruktur aufzusetzen und diese inklusive aller Abhängigkeiten und deployten Anwendungen sowohl an andere Standalone-, als auch an Clustersysteme auf sehr einfache Weise zu verteilen. Bei der lokalen Installation mit Docker-Containern besteht die Wahl zwischen einer portablen Single-Node-Umgebung und einem virtualisierten Cluster. Prinzipiell gelten für den Fall der Docker-Installation die gleichen Bedingungen, wie für eine native Installation. Darüber hinaus lässt ein Setup mit Docker-Containern auch mehr als einen Worker-Node zu. Doch auch hier muss beachtet werden, dass Spark seine Stärken beim In-Memory-Processing nur mit ausreichenden Ressourcen nutzen kann. Auch wenn Docker eine sehr leichtgewichtige Virtualisierungslösung darstellt, muss dennoch beachtet werden, dass hier nicht die vollen Ressourcen wie bei einer nativen Installation zur Verfügung stehen.

Bereits konfigurierte Docker-Container können in einer eigenen Plattform, dem sogenannten Docker Hub verwaltet, mit anderen Nutzern geteilt und heruntergeladen werden.

Im Folgenden wird exemplarisch gezeigt, wie ein Docker Container für Apache Spark aufgebaut, konfiguriert und deployt werden kann [Vya15]. Dieser Container wurde auf einem Windows Hostrechner innerhalb einer VirtualBox-Instanz erstellt. Diese virtuelle Maschine wurde mit

[3] Bei cgroups handelt es sich um eines der Linux-Kernel-Features, um Ressourcen (CPU, Speicher, I/O, etc.) für verschiedene Prozesse isolieren zu können. Weitere RIFs sind *capabilities, namespaces, SELinux, Netlink, Netfilte, AppAmor* (Vergleich [SS14])

[4] Microservices sind ein Designpattern in der Softwarearchitektur, in dem komplexe Anwendungen in kleine, unabhängige Services ausgelagert werden. Diese können mittels verschiedener Sprachen entwickelt werden und stellen ihre jeweilige API den Konsumenten zur Verfügung.

[5] Unter Immutability versteht man in der objektorientierten und funktionalen Programmierung, dass ein Objekt nach dessen Erzeugung nicht mehr modifiziert werden kann.

einer leichtgewichtigen CentOS-Installation aufgebaut.

Da CentOS auf einer RedHat-Linuxdistribution aufsetzt, wird als Paketmanager *YUM* eingesetzt. Für die einfache Provisionierung der Docker Container bietet sich du Nutzung von Vagrant an [Has15]. Dies ist eine Konfigurations- und Provisionierungslösung für virtuelle Maschinen und bietet seit Version 1.6 eine native Unterstützung von Docker. So lassen sich die Docker Container mittels entsprechender Scripts einfach konfigurieren und auf beliebige Umgebungen verteilen.

Listing 6.1: Setup Spark mit eigener JVM

```
1 FROM jvm
2 RUN yum clean all
3 RUN yum install -y tar yum-utils wget
4 RUN yum-config-manager --save
5 RUN yum update -y
6 RUN yum install -y java-1.8.0-openjdk-devel.x86_64
7 COPY spark-1.2.0-bin-hadoop2.4.tgz /opt/
8 RUN tar -xzf /opt/spark-1.2.0-bin-hadoop2.4.tgz -C /opt/
9 RUN echo "SPARK_HOME=/opt/spark-1.2.0-bin-hadoop2.4" >> /etc/environment
10 RUN echo "JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk/" >> /opt/spark
     -1.2.0-bin-hadoop2.4/conf/spark-env.sh
```

Listing 6.2: Vagrant Provisionierungs-Skript für Spark

```
1 # First instance is spark master.
2 $spark_num_instances = 2
3 Vagrant.configure("2") do |config|
4     # nodes definition
5     (1..$spark_num_instances).each do |i|
6         config.vm.define "scale#{i}" do |scale|
7             scale.vm.provider "docker" do |d|
8                 d.build_dir = "spark/"
9                 d.name = "node#{i}"
10                d.create_args = ["--privileged=true"]
11                d.remains_running = true
12                if "#{i}" == "1"
13                    d.ports = [ "4040:4040", "7704:7704" ]
14                else
15                    d.create_args = d.create_args << "--link" << "node1:
                          node1.dockerspark"
16                end
17            end
18            scale.vm.synced_folder "./", "/spark_infrastructure/"
19            scale.vm.hostname = "node#{i}.dockerspark"
20        end
21    end
```

6.3 Cluster Management: Mesos und Yarn

Blablba

6.4 Caching-Framework: Tachyon

Blablba

6.5 Der eigentliche Kern: Apache Spark

Blablba

6.6 Streaming-Framework: Spark Streaming

Blablba

6.7 Abfrageschicht: Spark SQL

Blablba

6.8 Machine Learning Algorithmen: MLLib

Blablba

6.9 Graphenanwendungen: GraphX

Blablba

6.10 Einrichten und Konfigurieren der IDE IntelliJ Idea für Spark

Blablba

6.11 Alternativimplementierung zu MLLibs: H2O

Blablba

6.12 Alternativimplementierung zu Spark: Apache Flink

Blablba

Kapitel 7

Implementierung der Prototypen

Blablba

7.1 Prototyp: Spark

Blablabla

7.1.1 Prototyp: Vergleich Prototyp Apache Flink

Blablba

7.2 Prototyp: MLLib

Blablba

7.2.1 Prototyp: Vergleich Prototyp H2O

Blablba

7.3 Prototyp: Spark Streaming

Blabla

7.4 Prototyp: GraphX

Blablba

Kapitel 8

Evaluierung der Komponenten und Alternativen

Im vorhergehenden Kapitel wurden die Implementierungen von Prototypen für einzelne Bibliotheken des BDAS vorgestellt. Nachfolgend werden diese Bibliotheken hinsichtlich ihres spezifischen Laufzeitverhaltens untersucht. Zu diesem Zweck wurden für die einzelnen Anwendungsbereiche zunächst Metriken definiert, die Aussagen über die relative Leistungsfähigkeit zulassen.

Für diese Beurteilung wurden unterschiedliche Umgebungen eingesetzt, wobei hier die relativen Performancewerte gegenüber den absoluten prioritär betrachtet werden.

Zum Einsatz für die Evaluierung kamen für diese Untersuchung unterschiedliche Ansätze für Clusterinfrastrukturen. Es kamen diverse lokale Installationen, sowie ein leistungsfähiges Rechnercluster an der Beuth Hochschule Berlin zum Einsatz.

Die Tabelle 8.1 gibt eine detaillierte Übersicht über die lokal verwendeten Hardwarekonfigurationen. Alle lokal eingesetzten Maschinen verfügen über SSDs (Solid State Drives) als Festspeicher.

Die folgende Tabelle zeigt die Konfiguration des verwendeten Clustersystems an der Beuth Hochschule Berlin (Konfigurationsstand Dezember 2014)

8.1 Definition von Metriken für die Bibliotheken des BDAS

Blabla

System	CPU	Anzahl Cores	RAM	OS
MacBook Pro Mid 2014	Intel i7	4 physisch, 8 mit Hyperthreading	16 GB	Mac OS-X 10.10.2 Yosemite
Apple iMac Mid 2011	Intel i5	4 physisch, 8 mit Hyperthreading	16 GB	Mac OS-X 10.10.2 Yosemite
Xeon Workstation	Intel Xeon 1230	4 physisch, 8 mit Hyperthreading	32 GB	Windows 8.1, VirtualBox Instanzen mit CentOS
Lenovo ThinkPad 410T	Intel i5	4 physisch	8 GB	Windows 8.1, VirtualBox Instanzen mit CentOS

Tabelle 8.1: Übersicht der lokal verwendeten Hardware

System	CPU	Anzahl Cores	RAM	OS
Master Node	AMD 6320	2 * 8	128 GB	Debian Linux
Worker Node 1	AMD 6378	4 * 16	512 GB	Debian Linux
Worker Node 2	AMD 6378	4 * 16	512 GB	Debian Linux
Worker Node 3	AMD 6378	4 * 16	512 GB	Debian Linux

Tabelle 8.2: Übersicht über das Cluster der Beuth Hochschule Berlin

8.2 Beschreibung der Messverfahren

Blabla

8.3 Beschreibung der Messumgebungen

8.3.1 Lokales Single Node Cluster

Blablba

8.3.2 Lokales Multi Node Cluster

Blablba

8.3.3 Remote Cluster an der Beuth Hochschule

Blablba

8.4 Ergebnisse

Blablba

8.4.1 Messergebnisse Apache Spark

Blablba

8.4.2 Messergebnisse Apache Flink

Blablba

8.4.3 Messergebnisse MLLib

Blablba

8.4.4 Messergebnisse H2O

Blablba

Kapitel 9

Schlussbetrachtung

Blablabla

9.1 Zusammenfassung

Blablba

9.2 Ausblick

Die bisher verbreiteten Cluster-Computing-Frameworks für Big Data Analytics, allen voran Hadoop, besitzen zwar umfangreiche Funktionen für Parallelverarbeitung und Funktionen für Task-Verteilung und Fehlerrobustheit, doch trotz umfangreicher Zugriffsmechanismen auf die Ressourcen eines Clusters werden die Vorteile der Nutzung von verteiltem Hauptspeicher hier nicht genutzt [MZ12]. Speziell für Aufgaben, die auf berechnete Zwischenergebnisse zugreifen müsssen,

RDDs are conceptually similar to views in a database, and persistent RDDs resemble materialized views [28]. However, like DSM systems, databases typically allow fine-grained read-write access to all records, requiring logging of operations and data for fault tolerance and additional overhead to maintain consistency. These overheads are not required with the coarse-grained transformation model of RDDs.

Kapitel 10

Verzeichnisse

Literaturverzeichnis

[Agn14] Agneeswaran, Vijay: *Big Data Analytics Beyond Hadoop*. Pearson, 2014.

[AST07] Andrew S. Tanenbaum, Maarten van Steen: *Distributed Systems - Principles and Paradigms*. Pearson, Prentice Hall, Second Edition Auflage, 2007.

[Bis06] Bishop, C.M.: *Pattern Recognition and Machine Learning*. Springer, 2006.

[BJA06] Benjamin J. Anderson, Deborah S. Gross, et al.: *Adapting K-Medians to Generate Normalized Cluster Centers*. Carleton College, Northfield Minnesota, 2006.

[Bro98] Brosius, Felix: *SPSS 8: Professionelle Statistik*. International Thompson Publishing, 1998.

[CBD08] Chuong B Do, Serafim Batzoglou: *What is the expectation maximization algorithm?* Nature Publishing Group, 2008.

[Cou13] Council, National Research: *Frontiers in Massive Data Analysis*. National Academic Press, 2013.

[CV14] Claus Vielhauer, Tobias Scheidat: *Fusion von biometrischen Verfahren zur Benutzerauthentifikation*. Otto-von-Guericke Universität Magdeburg Advanced Multimedia and Security Lab (AMSL), 2014.

[GJ13] Gareth James, Daniela Witten, et al.: *An Introduction to Statistical Learning with Applications in R*. Springer, 4 Auflage, 2013.

[HK15] Holden Karau, Andy Kowinski, Matei Zaharia: *Learning Spark - Lightning Fast Data Analysis*. O'Reilly, Early Release Auflage, 2015.

[HS83] H.A. Simon, P. Langley, G.L. Bradshaw: *Rediscovering chemistry with the BACON system - Machine learning, an artificial intelligence approach*. Tiogra Publishing Co., 1983.

[JA03] Johannes Assfalg, Christian Böhm, et. al: *Knowledge Discovery in Databases*. Ludwig Maximilians Universität München, Institut für Informatik, 2003.

[JC14] Jürgen Cleve, Uwe Lämmel: *Data Mining*. De Gruyter/Oldenbourg Wissenschaftsverlag, 1 Auflage, 2014.

[JD04] Jeff Dean, Sanjay Ghemawat: *MapReduce: Simplified Data Processing on Large Clusters*. Google, Inc., 2004.

[JWS90] Jude W. Shavlik, Thomas G. Dietterich: *Reading in Machine Learning*. Morgan-Kaufman Publishers, Inc., 1 Auflage, 1990.

[Kun14] Kuntamukkala, Ashwini: *DZone Refcardz 204: Apache Spark*. DZone, 1 Auflage, 2014.

[Mit15] Mitchell, Tom M.: *Machine Learning*. McGraw Hill, Draft of January 31, 2015 Auflage, 2015.

[MZ12] Matei Zaharia, Mosharaf Chowdhury, et al.: *Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing*. University of California, Berkeley, 2012.

[Ng02] Ng, Raymond T.: *CLARANS: a method for clustering objects for spatial data mining*. Nummer DOI: 10.1109/TKDE.2002.1033770. Dept. of Comput. Sci., British Columbia Univ., Vancouver, BC; IEEE Transactions on Knowledge and Data Engineering (Impact Factor: 1.82), 10 2002.

[NG12] Norbert Gronau, Corinna Fohrhol: *Wettbewerbsfaktor Analytics – Reifegrade ermitteln, Wirtschaftlichkeitspotentiale entdecken*. Addison-Wesley, 2012.

[NRP05] Nikhil R. Pal, Kuhu Pal, et. al: *A Possibilistic Fuzzy c-Means Clustering Algorithm*, Band 13. IEEE TRANSACTIONS ON FUZZY SYSTEMS, 2005.

[Rus11] Russom, Philip: *TDWI Best Practices Report: Big Data Analytics*. TDWI Research, 2011.

[Sat12] Sathi, Dr. Arvind: *Big Data Analytics: Disruptive Technologies for Changing the Game*. MC Press, First Edition Auflage, 2012.

[SS14] Stephen Soltesz, Andy Bavier, et al.: *Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors*. University of Toronto, Department of Computer Science, 2014.

[SW97] S.M. Weiss, N. Indurkhya: *Predictive data mining: A practical guide.* Morgan-Kaufman, 1997.

[TH09] T. Hastie, R. Tibshirani, et al.: *Data Mining, Inference, and Prediction*. Springer Publishing Company, 2009.

[TH14] Trevor Hastie, Rahul Mazumder, et al.: *Matrix Completion and Low-Rank SVD via Fast Alternating Least Squares*. Statistics Department and ICME Stanford University, 2014.

[TR10] Thomas Rauber, Gudula Rünger: *Parallel Programming: For Multicore and Cluster Systems*. Springer, 2010.

[UF96] U.M. Fayyad, G. Piatetsky-Shapiro, et al.: *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press, 1996.

[Whi13] White, Tom: *Hadoop: The Definitive Guide*, Band 2nd Edition. O'Reilly, 2013.

[WKH12] Wolfgang Karl Härdle, Léopold Simar: *Applied Multivariate Statistical Analysis*. Springer, 3 Auflage, 2012.

[YZ09] Yunhong Zhou, Dennis Wilkinson, et al.: *Large-scale Parallel Collaborative Filtering for the Netflix Prize*. HP Labs, Palo Alto, 2009.

Internetquellen

[AL15] Anna Lee, Gavin M. Joynt, et al.: *Making Sense of Decision Analysis Using a Decision Tree, US National Library of Medicine, National Institutes of Health URL: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2669856/*, 05 2009. Aufgerufen am 01.02.2015.

[AP09] Andrew Pavlo, Erik Paulson, et al.: *A Comparison of Approaches to Large-Scale Data Analysis. Brown University, Providence URL: http://database.cs.brown.edu/projects/mapreduce-vs-dbms/*, 08 2009.

[Apa14] Apache: *Cluster Mode Overview URL: http://spark.apache.org/docs/latest/cluster-overview.html*, 10 2014. Aufgerufen am 21.12.2014.

[BG15] Baltes-Götz, B.: *Logistische Regressionsanalyse mit SPSS, Universität Trier, ZIMK URL: http://www.uni-trier.de/fileadmin/urt/doku/logist/logist.pdf*, 06 2012. Aufgerufen am 27.01.2015.

[BIT14] BITKOM: *Big-Data-Technologien – Wissen für Entscheider URL: http://www.bitkom.org/files/documents/BITKOM_Leitfaden_Big-Data-Technologien-Wissen_fuer_Entscheider_Febr_2014.pdf*, 2 2014. Aufgerufen am 04.10.2014.

[Dat15] Databricks: *Movie Recommendation with MLLib, Databricks URL: https://databricks-training.s3.amazonaws.com/movie-recommendation-with-mllib.html*, 11 2014. Aufgerufen am 16.01.2015.

[DGC14] Dr. Goutam Chakraborty, Murali Krishna Pagolu: *Analysis of Unstructured Data: Applications of Text Analytics and Sentiment Mining URL: http://support.sas.com/resources/papers/proceedings14/1288-2014.pdf*, 08 2014. Aufgerufen am 08.10.2014.

[DJF15] Dr. J. Fürnkranz, Dr. G. Grieser: *Maschinelles Lernen: Symbolische Ansätze URL: http://www.ke.tu-darmstadt.de/lehre/archiv/ws0607/mldm/*, 2006. Aufgerufen am 22.01.2015.

[DML15] David M. Lane, David Scott, et al.: *Introduction to Statistics, Rice University, University of Houston URL: http://onlinestatbook.com/2/index.html*, 11 2013. Aufgerufen am 06.01.2015.

[Doc15] Docker: *Docker Documentation URL: https://docs.docker.com*, 12 2014. Aufgerufen am 25.01.2015.

[Dum14] Dumbhill, Edd: *What is big data? URL: http://radar.oreilly.com/2012/01/what-is-big-data.htm*, 11 2011. Aufgerufen am 04.10.2014.

[Enz15] Enzmann, Dr. Dirk: *Lineare Regression, Universität Hamburg, Juristische Fakultät URL: http://www2.jura.uni-hamburg.de/instkrim/kriminologie/Mitarbeiter/Enzmann/Lehre/StatIKrim/Regression.pdf* , 11 2014. Aufgerufen am 27.01.2015.

[Gha15] Ghahramani, Zoubin: *Unsupervised Learning URL: http://mlg.eng.cam.ac.uk/zoubin/papers/ul.pdf*, 09 2004. Aufgerufen am 18.01.2015.

[Has15] Hashimoto, Mitchell: *Feature Preview: Docker-Based Development Environments URL: http://www.vagrantup.com/blog/feature-preview-vagrant-1-6-docker-dev-environments.html*, 04 2014. Aufgerufen am 08.02.2015.

[Hor14] Hortonworks: *Hadoop Distributed File System URL: http://hortonworks.com/hadoop/hdfs/*, 04 2014. Aufgerufen am 26.11.2014.

[LB15] Leo Breiman, Adele Cutler: *Random Forests, University of Berkeley California URL: https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm*, 01 2006. Aufgerufen am 15.01.2015.

[Lou14] Loukides, Mike: *What is data science? URL: http://radar.oreilly.com/2010/06/what-is-data-science.html*, 10 2010. Aufgerufen am 16.10.2014.

[Mar15] Markowetz, Florian: *Klassifikation mit Support Vector Machines, Max-Planck-Institut für Molekulare Genetik, Berlin Center for Genome Based Bioinformatics URL: http://lectures.molgen.mpg.de/statistik03/docs/Kapitel_16.pdf* , 01 2003. Aufgerufen am 31.01.2015.

[McN14] McNutty, Eileen: *Understanding Big Data: The Seven V's, Dataconomy URL: http://dataconomy.com/seven-vs-big-data/*, 05 2014. Aufgerufen am 23.10.2014.

[Met14] Metz, Cade: *Spark: Open Source Superstar Rewrites Future of Big Data URL: http://www.wired.com/2013/06/yahoo-amazon-amplab-spark/all/*. Wired.com, 06 2013. Aufgerufen am 28.10.2014.

[Mir15] Mirkes, E. M.: *University of Leicester: K-means and K-medoids URL: http://www.math.le.ac.uk/people/ag153/homepage/KmeansKmedoids/Kmeans_Kmedoids.html*. University of Leicester, 01 2011. Aufgerufen am 15.01.2015.

[Spa14] Spark, Apache: *Spark Programming Guide. Apache Spark URL: http://spark.apache.org/docs/1.2.0/programming-guide.html*, 01 2015. Aufgerufen am 02.11.2014.

[ST15] Schmidt-Thieme, Lars: *Entscheidungsbaumverfahren, Universität Karlsruhe URL: http://marketing.wiwi.uni-karlsruhe.de/institut/viror/kaiman/kaiman/index.xml.html*, 11 2003. Aufgerufen am 24.01.2015.

[Sto15] Storage, Lascon: *URL: http://www.lascon.co.uk/Oracle-RAC.php*, 11 2014. Aufgerufen am 07.02.2015.

[Tee14] Tee, Jason: *Handling the four 'V's of big data: volume, velocity, variety, and veracity, The ServerSide URL: http://www.theserverside.com/feature/Handling-the-four-Vs-of-big-data-volume-velocity-variety-and-veracity*, 08 2013. Aufgerufen am 06.10.2014.

[Tur14] Turner, James: *Hadoop: What it is, how it works and what can it do URL: http://radar.oreilly.com/2011/01/what-is-hadoop.html*, 3 2010. Aufgerufen am 26.10.2014.

[VN15] Vaughan-Nichols, Steven J.: *Docker libcontainer unifies Linux container powers URL: http://www.zdnet.com/article/docker-libcontainer-unifies-linux-container-powers/*, 06 2014. Aufgerufen am 26.01.2015.

[Vya15] Vyas, Jay: *Microservice principles and Immutability – demonstrated with Apache Spark and Cassandra URL: http://developerblog.redhat.com/2015/01/20/microservice-principles-and-immutability-demonstrated-with-apache-spark-and-cassandra/*, 01 2015. Aufgerufen am 04.02.2015.

[Xia15] Xia, C.: *Work Structure of MapReduce URL: http://xiaochongzhang.me/blog/wp-content/uploads/2013/05/MapReduce_Work_Structure.png*, 05 2013. Aufgerufen am 04.02.2015.

[Xin14] Xin, Gonzales et al: *A Resilient Distributed Graph System on Spark URL: {https://amplab.cs.berkeley.edu/publication/graphx-grades*. Amplab UC Berkeley, 08 2013. Aufgerufen am 04.10.2014.

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Anhang A

Zusätze

A.1 Übersicht der RDD Transformationen

Die folgende Tabelle zeigt alle Transformationen, die auf Spark RDDs möglich sind (Entnommen aus [Spa14] - Stand Spark 1.2.0):

Transformation	Zweck
map(func)	Return a new distributed dataset formed by passing each element of the source through a function func.
filter(func)	Return a new dataset formed by selecting those elements of the source on which func returns true.
flatMap(func)	Similar to map, but each input item can be mapped to 0 or more output items (so func should return a Seq rather than a single item).
mapPartitions(func)	Similar to map, but runs separately on each partition (block) of the RDD, so func must be of type Iterator<T> => Iterator<U> when running on an RDD of type T.
mapPartitionsWithIndex(func)	Similar to mapPartitions, but also provides func with an integer value representing the index of the partition, so func must be of type (Int, Iterator<T>) => Iterator<U> when running on an RDD of type T.
sample(withReplacement, fraction, seed)	Sample a fraction fraction of the data, with or without replacement, using a given random number generator seed.
union(otherDataset)	Return a new dataset that contains the union of the elements in the source dataset and the argument.
intersection(otherDataset)	Return a new RDD that contains the intersection of elements in the source dataset and the argument.
distinct([numTasks]))	Return a new dataset that contains the distinct elements of the source dataset.

Transformation	Zweck
groupByKey([numTasks])	When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<V>) pairs. Note: If you are grouping in order to perform an aggregation (such as a sum or average) over each key, using reduceByKey or combineByKey will yield much better performance. Note: By default, the level of parallelism in the output depends on the number of partitions of the parent RDD. You can pass an optional numTasks argument to set a different number of tasks.
reduceByKey(function, [numTasks])	When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function func, which must be of type (V,V) => V. Like in groupByKey, the number of reduce tasks is configurable through an optional second argument.
aggregateByKey(zeroValue) (seqOp, combOp, [numTasks])	When called on a dataset of (K, V) pairs, returns a dataset of (K, U) pairs where the values for each key are aggregated using the given combine functions and a neutral ßero"value. Allows an aggregated value type that is different than the input value type, while avoiding unnecessary allocations. Like in groupByKey, the number of reduce tasks is configurable through an optional second argument.
sortByKey([ascending], [numTasks])	When called on a dataset of (K, V) pairs where K implements Ordered, returns a dataset of (K, V) pairs sorted by keys in ascending or descending order, as specified in the boolean ascending argument.

Transformation	Zweck
join(otherDataset, [numTasks])	When called on datasets of type (K, V) and (K, W), returns a dataset of (K, (V, W)) pairs with all pairs of elements for each key. Outer joins are supported through leftOuterJoin, rightOuterJoin, and fullOuterJoin.
cogroup(otherDataset, [numTasks])	When called on datasets of type (K, V) and (K, W), returns a dataset of (K, Iterable<V>, Iterable<W>) tuples. This operation is also called groupWith.
cartesian(otherDataset)	When called on datasets of types T and U, returns a dataset of (T, U) pairs (all pairs of elements).
pipe(command, [envVars])	Pipe each partition of the RDD through a shell command, e.g. a Perl or bash script. RDD elements are written to the process's stdin and lines output to its stdout are returned as an RDD of strings.
coalesce(numPartitions)	Decrease the number of partitions in the RDD to numPartitions. Useful for running operations more efficiently after filtering down a large dataset.
repartition(numPartitions)	Reshuffle the data in the RDD randomly to create either more or fewer partitions and balance it across them. This always shuffles all data over the network.
repartitionAndSortWithin Partitions(partitioner)	Repartition the RDD according to the given partitioner and, within each resulting partition, sort records by their keys. This is more efficient than calling repartition and then sorting within each partition because it can push the sorting down into the shuffle machinery.

Die folgende Tabelle zeigt alle Actions, die auf Spark RDDs möglich sind (Entnommen aus [Spa14] - Stand Spark 1.2.0):

Actions	Zweck
reduce(func)	Aggregate the elements of the dataset using a function func (which takes two arguments and returns one). The function should be commutative and associative so that it can be computed correctly in parallel.
collect()	Return all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data.
count()	Return the number of elements in the dataset.
first()	Return the first element of the dataset (similar to take(1)).
take(n)	Return an array with the first n elements of the dataset. Note that this is currently not executed in parallel. Instead, the driver program computes all the elements.
takeSample(withReplacement, num, [seed])	Return an array with a random sample of num elements of the dataset, with or without replacement, optionally pre-specifying a random number generator seed.
takeOrdered(n, [ordering])	Return the first n elements of the RDD using either their natural order or a custom comparator.
saveAsTextFile(path)	Write the elements of the dataset as a text file (or set of text files) in a given directory in the local filesystem, HDFS or any other Hadoop-supported file system. Spark will call toString on each element to convert it to a line of text in the file.

Actions	Zweck
saveAsSequenceFile(path)	Write the elements of the dataset as a Hadoop SequenceFile in a given path in the local filesystem, HDFS or any other Hadoop-supported file system. This is available on RDDs of key-value pairs that either implement Hadoop's Writable interface. In Scala, it is also available on types that are implicitly convertible to Writable (Spark includes conversions for basic types like Int, Double, String, etc).
saveAsObjectFile(path)	Write the elements of the dataset in a simple format using Java serialization, which can then be loaded using SparkContext.objectFile().
countByKey()	Only available on RDDs of type (K, V). Returns a hashmap of (K, Int) pairs with the count of each key.
foreach(func)	Run a function func on each element of the dataset. This is usually done for side effects such as updating an accumulator variable (see below) or interacting with external storage systems.