jonas **FREIKNECHT**
stefan **PAPP**

BIG DATA in der Praxis

Lösungen mit **Hadoop**, **Spark**, **HBase** und **Hive**
Daten speichern, aufbereiten, visualisieren

2. Auflage

HANSER

Im Internet: **Github-Repository zum Buch**

Freiknecht/Papp

Big Data in der Praxis

Jonas Freiknecht
Stefan Papp

Big Data in der Praxis

Lösungen mit Hadoop, Spark, HBase und Hive

Daten speichern, aufbereiten, visualisieren

2., erweiterte Auflage

HANSER

Die Autoren:

Jonas Freiknecht, Karlsruhe, www.jofre.de

Stefan Papp, Gaflenz, stefan.papp@data-wizard.net

Bibliografische Information der Deutschen Nationalbibliothek:

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Lektorat: Sylvia Hasselbach
Copy editing: Jürgen Dubau, Freiburg/Elbe
Umschlagdesign: Marc Müller-Bremer, München, www.rebranding.de
Umschlagrealisation: Stephan Rönigk
Gesamtherstellung: Kösel, Krugzell
Ausstattung patentrechtlich geschützt. Kösel FD 351, Patent-Nr. 0748702
Printed in Germany

Print-ISBN: 978-3-446-45396-8
E-Book-ISBN: 978-3-446-45601-3

„Wenn Du schnell gehen willst, geh allein.
Wenn Du weit gehen willst, geh mit anderen.“

Afrikanisches Sprichwort

Inhalt

1 Einleitung 1

2 Big Data 7

2.1 Historische Entstehung 9

2.2 Big Data – ein passender Begriff? 10

2.2.1 Die drei V 11

2.2.2 Weitere Vs 14

2.2.3 Der Verarbeitungsaufwand ist big 14

2.2.4 Sicht der Industrie auf Big Data 15

2.3 Eingliederung in BI und Data Mining 16

3 Hadoop 21

3.1 Hadoop kurz vorgestellt 21

3.2 HDFS – das Hadoop Distributed File System 23

3.3 Hadoop 2.x und YARN 28

3.4 Hadoop als Single-Node-Cluster aufsetzen 30

3.4.1 Falls etwas nicht funktioniert 44

3.5 Map Reduce 46

3.6 Aufsetzen einer Entwicklungsumgebung 49

3.7 Implementierung eines Map-Reduce-Jobs 56

3.8 Ausführen eines Jobs über Kommandozeile 68

3.9 Verarbeitung im Cluster 72

3.10 Aufsetzen eines Hadoop-Clusters 74

3.11 Starten eines Jobs via Hadoop-API 86

3.12 Verketten von Map-Reduce-Jobs 99

3.13 Verarbeitung anderer Dateitypen 115

3.14 YARN-Anwendungen 130

3.14.1 Logging und Log-Aggregation in YARN 131

3.14.2 Eine einfache YARN-Anwendung 134

3.15 Vor- und Nachteile der verteilten Verarbeitung 159

3.16 Die Hadoop Java-API ... 160
3.16.1 Ein einfacher HDFS-Explorer ... 161
3.16.2 Cluster-Monitor ... 173
3.16.3 Überwachen der Anwendungen im Cluster ... 175
3.17 Gegenüberstellung zur traditionellen Verarbeitung ... 177
3.18 Big Data aufbereiten ... 178
3.18.1 Optimieren der Algorithmen zur Datenauswertung ... 178
3.18.2 Ausdünnung und Gruppierung ... 180
3.19 Ausblick auf Apache Spark ... 182
3.20 Markt der Big-Data-Lösungen ... 184

4 Das Hadoop-Ecosystem ... 187
4.1 Ambari ... 188
4.2 Sqoop ... 189
4.3 Flume ... 189
4.4 HBase ... 190
4.5 Hive ... 191
4.6 Pig ... 191
4.7 ZooKeeper ... 191
4.8 Oozie ... 192
4.9 Mahout ... 193
4.10 Data Analytics und das Reporting ... 193

5 NoSQL und HBase ... 195
5.1 Historische Entstehung ... 195
5.2 Das CAP-Theorem ... 196
5.3 ACID und BASE ... 197
5.4 Typen von Datenbanken ... 198
5.5 Umstieg von SQL und Dateisystemen auf NoSQL oder HDFS ... 201
5.5.1 Methoden der Datenmigration ... 201
5.6 HBase ... 203
5.6.1 Das Datenmodell von HBase ... 203
5.6.2 Aufbau von HBase ... 206
5.6.3 Installation als Stand-alone ... 207
5.6.4 Arbeiten mit der HBase Shell ... 209
5.6.5 Verteilte Installation auf dem HDFS ... 211
5.6.6 Laden von Daten ... 214
5.6.7 HBase Java-API ... 226
5.6.8 Der Umstieg von einem RDBMS auf HBase ... 249

6 Data Warehousing mit Hive ... 253
6.1 Installation von Hive ... 254

6.2 Architektur von Hive ... 256
6.3 Das Command Line Interface (CLI) ... 257
6.4 HiveQL als Abfragesprache ... 259
6.4.1 Anlegen von Datenbanken ... 259
6.4.2 Primitive Datentypen ... 260
6.4.3 Komplexe Datentypen ... 260
6.4.4 Anlegen von Tabellen ... 261
6.4.5 Partitionierung von Tabellen ... 262
6.4.6 Externe und interne Tabellen ... 262
6.4.7 Löschen und Leeren von Tabellen ... 263
6.4.8 Importieren von Daten ... 264
6.4.9 Zählen von Zeilen via count ... 265
6.4.10 Das SELECT-Statement ... 265
6.4.11 Beschränken von SELECT über DISTINCT ... 269
6.4.12 SELECT auf partitionierte Tabellen ... 269
6.4.13 SELECT sortieren mit SORT BY und ORDER BY ... 270
6.4.14 Partitionieren von Daten durch Bucketing ... 271
6.4.15 Gruppieren von Daten mittels GROUP BY ... 272
6.4.16 Subqueries – verschachtelte Abfragen ... 273
6.4.17 Ergebnismengen vereinigen mit UNION ALL ... 273
6.4.18 Mathematische Funktionen ... 274
6.4.19 String-Funktionen ... 276
6.4.20 Aggregatfunktionen ... 276
6.4.21 User-Defined Functions ... 277
6.4.22 HAVING ... 285
6.4.23 Datenstruktur im HDFS ... 286
6.4.24 Verändern von Tabellen ... 286
6.4.25 Erstellen von Views ... 289
6.4.26 Löschen einer View ... 289
6.4.27 Verändern einer View ... 289
6.4.28 Tabellen zusammenführen mit JOINs ... 290
6.5 Hive Security ... 292
6.5.1 Implementieren eines Authentication-Providers ... 298
6.5.2 Authentication-Provider für HiveServer2 ... 303
6.5.3 Verwenden von PAM zur Benutzerauthentifizierung ... 303
6.6 Hive und JDBC ... 304
6.7 Datenimport mit Sqoop ... 322
6.8 Datenexport mit Sqoop ... 324
6.9 Hive und Impala ... 325
6.10 Unterschied zu Pig ... 326
6.11 Zusammenfassung ... 327

7 Big-Data-Visualisierung ... 329
7.1 Theorie der Datenvisualisierung ... 329
7.2 Diagrammauswahl gemäß Datenstruktur ... 335
7.3 Visualisieren von Big Data erfordert ein Umdenken ... 336
7.3.1 Aufmerksamkeit lenken ... 337
7.3.2 Kontextsensitive Diagramme ... 339
7.3.3 3D-Diagramme ... 341
7.3.4 Ansätze, um Big-Data zu visualisieren ... 342
7.4 Neue Diagrammarten ... 344
7.5 Werkzeuge zur Datenvisualisierung ... 348
7.6 Entwicklung einer einfachen Visualisierungskomponente ... 352

8 Auf dem Weg zu neuem Wissen – Aufbereiten, Anreichern und Empfehlen ... 365
8.1 Eine Big-Data-Table als zentrale Datenstruktur ... 368
8.2 Anreichern von Daten ... 370
8.2.1 Anlegen einer Wissensdatenbank ... 371
8.2.2 Passende Zuordnung von Daten ... 372
8.3 Diagrammempfehlungen über Datentypanalyse ... 376
8.3.1 Diagrammempfehlungen in der BDTable ... 378
8.4 Textanalyse – Verarbeitung unstrukturierter Daten ... 384
8.4.1 Erkennung von Sprachen ... 385
8.4.2 Natural Language Processing ... 386
8.4.3 Mustererkennung mit Apache UIMA ... 394

9 Infrastruktur ... 415
9.1 Hardware ... 416
9.2 Betriebssystem ... 417
9.2.1 Paketmanager ... 417
9.2.2 Git ... 418
9.2.3 VIM ... 419
9.2.4 Terminalumgebung ... 419
9.3 Virtualisierung ... 420
9.4 Container ... 420
9.4.1 Docker-Crashkurs ... 421
9.4.2 Infrastructure as Code ... 424
9.5 Distributionen ... 424
9.6 Reproduzierbarkeit ... 425
9.7 Zusammenfassung ... 425

10 Programmiersprachen ... 427
10.1 Merkmale ... 428
10.1.1 Funktionale Paradigmen ... 428

10.2 Big-Data-Programmiersprachen ... 429
10.2.1 Java ... 429
10.2.2 Scala ... 430
10.2.3 Python ... 433
10.2.4 R ... 436
10.2.5 Weitere Programmiersprachen ... 437
10.3 Zusammenfassung ... 438

11 Polyglot Persistence ... 439
11.1 Praxis ... 440
11.1.1 Redis ... 440
11.1.2 MongoDB ... 443
11.1.3 Neo4j ... 443
11.1.4 S3 ... 444
11.1.5 Apache Kudu ... 447
11.2 Zusammenfassung ... 447

12 Apache Kafka ... 449
12.1 Der Kern ... 450
12.2 Erste Schritte ... 450
12.3 Dockerfile ... 454
12.4 Clients ... 454
12.5 Python Chat Client ... 454
12.6 Zusammenfassung ... 456

13 Data Processing Engines ... 457
13.1 Von Map Reduce zu GPPEs ... 457
13.1.1 Herausforderungen ... 458
13.1.2 Verfahren zur Verbesserung ... 459
13.1.3 Von Batch und Streaming zu Lambda ... 461
13.1.4 Frameworks in a Nutshell ... 462
13.2 Apache Spark ... 462
13.2.1 Datasets ... 462
13.2.2 Von RDDs zu Data Frames ... 463
13.2.3 Hands On Apache Spark ... 463
13.2.4 Client-Programme schreiben ... 465
13.2.5 Das Spark-Ecosystem ... 470
13.3 Zusammenfassung ... 474

14 Streaming ... 475
14.1 Kernparadigmen ... 475
14.2 Spark Streaming ... 478
14.2.1 Beispiel ... 479

14.3 Apache Flink ... 480
14.4 Zusammenfassung ... 483

15 Data Governance ... 485
15.1 Begriffsdschungel ... 486
15.2 Governance-Pfeiler ... 487
15.2.1 Transparenz ... 487
15.2.2 Verantwortung ... 488
15.2.3 Standardisierung ... 489
15.3 Fokusthemen von Data Governance ... 489
15.3.1 Policies ... 489
15.3.2 Quality ... 490
15.3.3 Compliance ... 490
15.3.4 Business Intelligence ... 490
15.4 Datenschutz ... 491
15.4.1 Werkzeuge ... 492
15.5 Sicherheit im Hadoop-Ecosystem ... 497
15.6 Metadatenmanagement ... 498
15.6.1 Open-Source-Werkzeuge ... 499
15.6.2 Kommerzielle Datenkataloge ... 500
15.7 Organisatorische Themen ... 500
15.7.1 Privacy by Design ... 501
15.7.2 k-Anonymity ... 501
15.7.3 Standards ... 503
15.8 Zusammenfassung ... 503

16 Zusammenfassung und Ausblick ... 505
16.1 Zur zweiten Auflage 2018 ... 505
16.2 Zur ersten Auflage 2014 ... 507

17 Häufige Fehler ... 511

18 Anleitungen ... 517
18.1 Installation und Verwendung von Sqoop2 ... 517
18.2 Hadoop für Windows 7 kompilieren ... 523

19 Literaturverzeichnis ... 527

Index ... 531

1 Einleitung

Der Begriff Big Data ist seit der ersten Auflage im Jahr 2014 zu einem zentralen Thema in der IT-Landschaft von Unternehmen geworden. Kaum jemand spricht noch von einem kurzfristigen Hype. Big-Data-Technologien bestimmen auch den Inhalt vieler Fachzeitschriften und -bücher.

Mit Apache Hadoop und NoSQL-Datenbanken können große, meist unstrukturierte Datenmengen effizient verteilt verarbeitet werden. Darüber hinaus haben in der letzten Zeit Data Processing Engines wie Apache Spark und Messaging-Systemen wie Kafka an Popularität gewonnen, da sie neue Möglichkeiten bieten, wie noch schneller Wert aus Daten extrahiert werden kann. Machine-Learning- und Deep-Learning-Werkzeuge sind mittlerweile ebenfalls ein wesentlicher Teil einer Big-Data-Systemlandschaft.

In den Medien wird häufig der generelle Mehrwert, der durch den Einsatz von Big-Data-Technologien für Banken, Automobilhersteller, Forschungseinrichtungen, Versicherungen etc. entsteht, hervorgehoben. Es wird die Notwendigkeit betont, sich mit den firmeninternen und öffentlichen Datenmengen zu beschäftigen, um dem eigenen Unternehmen einen Wettbewerbsvorteil zu verschaffen. Durch die Auswertung großer Datenmengen sollen neue, geschäftskritische Informationen gewonnen werden, welche die Grundlage für Unternehmensentscheidungen sein können. Daten gelten als das Öl des 21. Jahrhunderts.

Die sogenannte „Value Proposition“ für Unternehmen ist somit klar: Mit Big Data können Firmen Kosten sparen, schnellere Entscheidungen treffen und neue Märkte generieren. Zahlreiche Berater nutzen Domänensprache, also eine spezialisierte Businesssprache, und erstellen für ihre Kunden neue Churn-Modelle[1], Customer Experience Solutions[2] und Data Monetarisation Use Cases[3].

Die Frage nach dem *Wie* bleibt in vielen Büchern unbeantwortet. Wie funktionieren die Lösungen im Detail, um Informationen aus Daten zu extrahieren? Wie integrieren sich Hadoop und Co. in bisherige Business-Intelligence-Architekturen und wie ist das Zusammenspiel mit dem Data Warehouse? Welche Bedeutung haben Frameworks wie Spark und Kafka in einer modernen Big-Data-Architektur? Und wie schaffen es Data Scientists, große

[1] Modelle, in denen untersucht wird, warum Kunden kündigen und wie man das verhindern kann.

[2] Lösungen, mit denen man untersucht, wie sich Kunden verhalten, um ihr Verhalten abzuschätzen und zu lernen, wie man Geschäftsprozesse optimieren kann, um Kundenanliegen besser zu erfüllen.

[3] Data Monetisation ist die Disziplin, wie Firmen Geld aus Daten machen können. Beispielsweise können Telekommunikationsunternehmen gesammelte Geolokationsdaten auch an Dritte verkaufen

Datenmengen aufzubereiten, zu visualisieren und den Fachabteilungen zugänglich zu machen? Mit welchen Tools wird eine Big-Data-Architektur geschaffen, um für neue, bisher unbekannte Herausforderungen optimal aufgestellt zu sein?

Dieser Fragenkatalog ließe sich beliebig fortführen und erweitern, denn die *technischen* Aspekte des Big-Data-Trends werden nur selten in ausreichendem Detailgrad diskutiert. Und wenn, dann nicht in Form von Gesamtlösungen, sondern in kleinen, gut verdaulichen Häppchen. Gründe dafür gibt es viele. Zum einen sind Big Data-Experten rar, und nur wenige haben die Zeit, ihr Wissen in Büchern oder Fachartikeln weiterzugeben. Darüber hinaus steht hinter Big Data nicht nur eine einzelne neue Technologie. Big-Data-Architekturen basieren oft auf vielen verschiedenen Komponenten, die erst im Zusammenspiel ihr ganzes Potenzial entfalten.

Das Thema ist unter technischen Gesichtspunkten umfangreich und bietet zahlreiche Bereiche, in denen es sich neues Wissen anzueignen gilt. Wichtig ist daher, dass Sie eine hohe Affinität zum Forschen und zum Experimentieren mitbringen. Viele Big-Data-Technologien werden laufend weiterentwickelt und andere verschwinden wieder. Dazu kommen die unterschiedlichsten Anforderungen von Unternehmen an Big Data, die wiederum unterschiedlich aufgebaute Big-Data-Architekturen erfordern.

Was den Leser in diesem Buch erwartet

Big Data wird in diesem Buch aus einer Engineering-Sicht betrachtet. Einzelne fachliche Anwendungsgebiete werden natürlich berücksichtigt, aber das Primärziel ist, Lesern, die an der Technologie interessiert sind, einen praktischen Einstieg in moderne Lösungsansätze zu bieten, die für die Verarbeitung von Daten z. B. aus sozialen Netzwerken, unstrukturierten Webseiten, umfangreichen Fließtextdokumenten und geografischen Daten nötig sind. Dabei wird nicht nur gezeigt, wie große Datenmengen in einem Cluster verarbeitet, sondern auch über ein Data Warehouse bereitgestellt oder mit neuen, innovativen Diagrammen visualisiert werden können. Themen wie *NoSQL* werden besprochen und im praktischen Teil HBase als Vertreter dieser Kategorie aktiv eingesetzt. Apache Hive wird als Data-Warehouse-Software vorgestellt, um zu zeigen, inwiefern auf Big Data mit Abfragesprachen ähnlich SQL zugegriffen werden kann. Darüber hinaus wird Hive dann auch neueren SQL-Trends wie dem Framework Apache Spark gegenübergestellt. Sie lernen, welche neuen Diagrammarten dabei unterstützen, große Datenmengen mit komplexen Beziehungen untereinander zu visualisieren und zu verstehen. Diese versprochenen Erläuterungen werden nicht nur in Textform gegeben, Sie werden auch aktiv in den Entwicklungsprozess miteinbezogen und wo möglich, werden die theoretischen Hintergründe nähergebracht. Neben den bekannten Apache-Projekten wie Hadoop, Hive und HBase werden auch einige weniger bekannte Frameworks wie Apache UIMA oder Apache OpenNLP besprochen, um gezielt die Verarbeitung unstrukturierter Daten zu behandeln. Dazu wird in dem Buch gezeigt, wie Sie kleinere Projekte entwickeln, um die Kniffe bezüglich der Nutzung der neuen Software kennenzulernen und zu verstehen. Das Ziel ist es, Sie auf den Effekt und den Mehrwert der neuen Möglichkeiten aufmerksam zu machen, sodass Sie diese konstruktiv in Ihr Unternehmen tragen und für sich und Ihre Kollegen ein Bewusstsein für den Wert Ihrer Daten schaffen.

Voraussetzungen

Wie im Vorwort erwähnt, ist die wichtigste Voraussetzung sicherlich die Experimentierfreude und die Bereitschaft, Neues zu erlernen und alte Gewohnheiten und Denkweisen zu hinterfragen. Die technischen Vorkenntnisse, die Sie mitbringen sollten, um dieses Buch flüssig lesen und nachvollziehen zu können, lassen sich in die drei Bereiche *Entwicklungsumgebungen*, *Entwicklung* und *Betrieb* unterteilen. Sie sollten sich mit Java und IntelliJ oder Eclipse als Entwicklungsumgebung auskennen. Da Hadoop, HBase, Hive, Sqoop etc. auf Ubuntu, in virtuellen Umgebungen und Dockercontainern installiert wird, ist es ebenfalls hilfreich, sich in einer Unix-Umgebung bewegen zu können und die grundlegenden Befehle zu kennen, um etwa Verzeichnisse zu wechseln, anzulegen oder zu löschen. Zudem ist eine etwas stärkere Systemumgebung vonnöten, um alle Szenarien aus dem Buch zu Hause nachvollziehen zu können. Wesentlich ist dabei RAM. 8 GB RAM sollten schon vorhanden sein. Die gute Nachricht ist, dass Sie für die verwendete Software kein Geld bezahlen müssen, denn alle in diesem Buch verwendeten Komponenten stehen in vollem Umfang kostenlos im Internet zur Verfügung.

Alle Beispiele in diesem Buch werden Schritt für Schritt aufgebaut und führen Sie somit hin zu einer fertigen und funktionstüchtigen Implementierung.

Die verwendeten Softwareversionen

In der ersten Auflage dieses Buches wurde Hadoop in der Version 2.2.0 verwendet, ergänzt um die weiteren Komponenten wie Hive und HBase in der jeweils zum damaligen Zeitpunkt aktuellen Version. Diese Versionierung wurde in der 2. Auflage beibehalten.

Die vorgestellten Komponenten wie Hadoop, Hive, HBase werden natürlich regelmäßig um neue Features erweitert, bei der Kernarchitektur wird jedoch auf eine hohe Abwärtskompatibilität geachtet. Aus diesem Grund sind alle Beispiele aus dem Buch weiterhin lauffähig und die Software kann entsprechend der Anleitungen im Buch installiert werden. Wenn Sie mit einer neueren Version arbeiten möchten, muss lediglich die Versionsnummer angepasst werden.

Steht also im Buch beispielsweise

```
sudo wget http://mirror.lwnetwork.org.uk/APACHE/hadoop/common/hadoop-2.2.0/hadoop-
2.2.0.tar.gz
```

dann können Sie z. B. `hadoop-2.2.0` durch `hadoop-3.1.0` ersetzen und für die Java-Installation die Version 8 verwenden, wobei Sie anstelle von `sudo apt-get install openjdk-7-jdk` den Befehl `sudo apt-get install openjdk-8-jdk` ausführen.

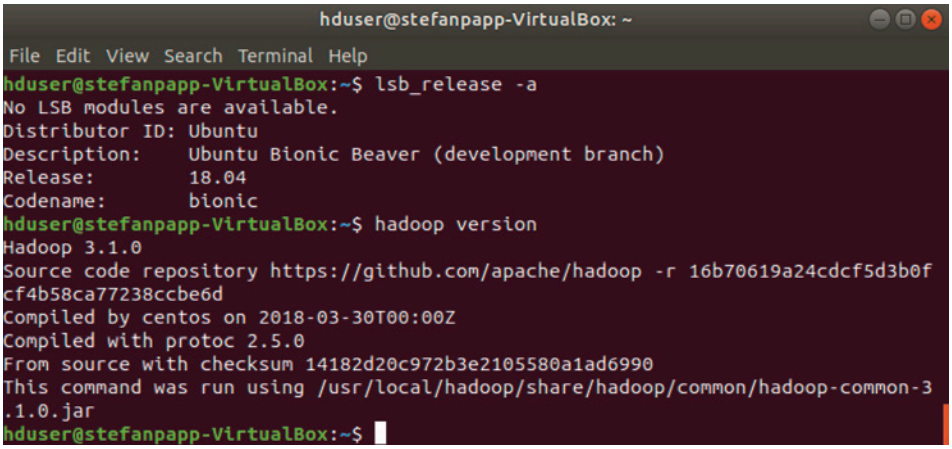

Bild 1.1 Beispiel mit Hadoop 3.1.0 und Ubuntu 18.04

Sie können also problemlos mit den aktuellen Versionen arbeiten, indem Sie beim Ausführen der Praxisbeispiele die Versionsnummern ersetzen. Darüber hinaus können Sie die Beispiele natürlich auch auf Basis der in diesem Buch verwendeten Versionen nachvollziehen.

Für wen ist dieses Buch geschrieben?

Dieses Buch ist für Menschen konzipiert, die sich in ihrem Beruf praktisch mit dem Thema Big Data auseinandersetzen. Dieses Buch soll aber nicht nur Informatiker ansprechen, sondern auch solche, die aus anderen Disziplinen kommen und vielleicht eine tolle Idee (und jede Menge Daten) haben und damit planen, z. B. ein Start-up zu gründen. Ebenso sollen Studenten und Auszubildende mit dieser Lektüre begleitet werden, die das Buch als Ergänzung zu einer Vorlesung oder zum Schreiben einer Seminararbeit verwenden können. Wenn Sie sich also zu einer Gruppe der in Bild 1.1 genannten Tätigkeitsfelder zählen oder mit einem der genannten Themen beschäftigen, dann lohnt sich ein Blick ins Buch besonders.

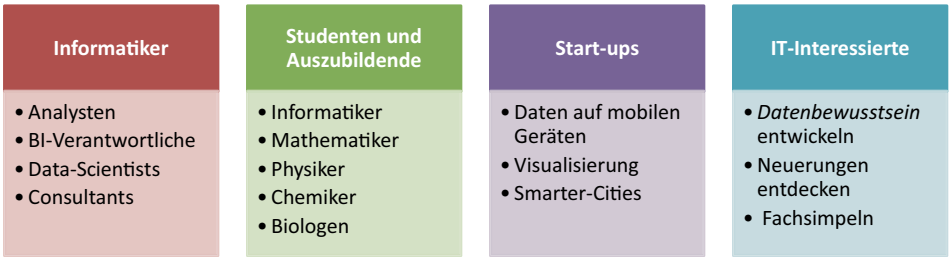

Bild 1.2 Gruppen und Themen, die in Zusammenhang mit Big Data genannt werden

Menschen, die Big Data von einer Business-Ebene betrachten, können durchaus das eine oder andere aus dem Buch lernen, wenngleich der Fokus des Buches mehr auf Technologie als auf den Möglichkeiten einer „Data Monetisation“ liegt. Wer jedoch die Technologie versteht, wird auch besser verstehen, wie man daraus Geschäftsfelder erschließen kann.

Ein großes Thema bei Big Data sind auch gesellschaftliche Auswirkungen der Digitalisierung (bei der Big Data eine wesentliche Rolle spielt) und das Thema Datenschutz. In einer Zeit, in der Unternehmen bei Datenmissbrauch horrende Strafen drohen, kann dieser Aspekt nicht ignoriert werden. Allerdings wird sich dieses Buch mehr auf die technischen als auf die juristischen oder ethischen Herausforderungen konzentrieren.

Warum „Big Data in der Praxis“?

Jonas Freiknecht, der Autor der ersten Auflage, sieht sich als Learning-by-Doing-Typ, dem es liegt, sich Wissen anhand von praktischen Erfahrungen anzueignen. Wenn er Befehle auf der Tastatur eingibt, kann er sich diese einfach besser merken. Häufig entsteht auch dann ein tiefergehendes Interesse an der Materie, mit der er sich gerade beschäftigt, sodass er dann bereit ist, die Theorie nachzuholen.

Stefan Papp, der Autor, der die erste Auflage des Buches aktualisiert und erweitert hat, sieht das genauso: Technische Details lassen sich oft durch die praktische Anwendung am besten verstehen. Dadurch erhoffen sich beide Autoren den Effekt, dass Sie die Verbindung zwischen Hintergrundwissen und der tatsächlichen Anwendung schnell herstellen und verinnerlichen und es Ihnen beim Lesen nicht langweilig wird.

Vorgehensweise und Struktur

Dieses Buch beginnt mit einer theoretischen Einführung in alle Themen rund um Big Data. Neben der historischen Entwicklung des Begriffs und der Diskussion einiger unterschiedlicher Definitionen wird mithilfe von Studien und Umfragen gezeigt, welche Industrien welche Themen mit dem Begriff Big Data verbinden. Somit wird die Erwartungshaltung verschiedener Menschen in Bezug auf den Trend dargelegt. Eine Gegenüberstellung der Begrifflichkeiten BI, Data Mining und Big Data soll dabei helfen, Zusammenhänge, Unterschiede und gegenseitige Ergänzung der drei Begrifflichkeiten zu erkennen und Sie befähigen, diese gegeneinander abzugrenzen und an anderer Stelle Schnittpunkte zwischen ihnen zu finden.

Im Anschluss folgt ein Kapitel über Hadoop, in dem die Installation, Konfiguration und Bedienung erklärt wird. Dabei wird jeweils auf die Besonderheiten bei der Verwendung mit einem *Single-* oder *Multi-Node-Cluster* eingegangen. Es schließt sich die Entwicklung von Map-Reduce-Jobs und YARN-Anwendungen an, gefolgt von einem ausführlichen Abschnitt zur Arbeit mit der Hadoop-API, um den Zugriff auf das HDFS, den Resource Manager etc. zu erklären.

Nachdem die Funktionsweise und die Idee hinter Hadoop bekannt sind, wird in Kapitel 4 kurz vorgestellt, welche Projekte um Hadoop herum in dessen Eco-System existieren und welche Aufgaben diese haben. In Kapitel 5 wird das Thema NoSQL aufgegriffen, theoretisch erläutert und praktisch unter Zuhilfenahme von HBase umgesetzt. Dabei wird nicht nur gezeigt, wie HBase installiert und eingerichtet wird, sondern auch, wie auf dessen Daten, entweder über das Terminal oder die Java-API, zugegriffen werden kann.

Analog dazu wird in Kapitel 6 das Thema Data Warehousing mit Apache Hive besprochen und gezeigt, wie sich Hive in das Big-Data-Umfeld integrieren lässt. Elementarer Bestandteil dieses Kapitels ist die Abfragesprache *HiveQL* mit all ihren Ausprägungen und die Verwendung von Hive über einen herkömmlichen JDBC-Adapter, um aus einer Java-Anwendung Abfragen absetzen und auswerten zu können.

Das Thema Datenvisualisierung wird in Kapitel 7 behandelt, in dem zuerst einige Visualisierungsframeworks vorgestellt und verglichen werden. Danach wird mit *D3.js* ein Set von Visualisierungskomponenten erarbeitet, mit denen in einer Beispielanwendung ein paar ansehnliche Diagramme gezeichnet werden. Im theoretischen Teil dieses Kapitels geht es darum, was man beachten muss, wenn jemand plant, große Datenmengen auf kleinem Raum unterzubringen, und welche Trends und Möglichkeiten es dabei gibt.

In Kapitel 8 soll das Thema Informationsgewinnung nähergebracht werden, das zum einen einen Zusammenschnitt aller bisher kennengelernten Techniken in einer schicken Gesamtlösung vereint und des Weiteren auf die Besonderheiten bei der Verarbeitung von unstrukturierten Daten mit aktuellen Text-Mining-Frameworks eingeht, darunter Apache UIMA und Apache OpenNLP. Diese werden ebenfalls als Bestandteil in das hier zu entwickelnde Programm einfließen.

Erweitert wurde die zweite Auflage des Buches um sechs neue Kapitel zu Hadoop-unabhängigen Frameworks. In Kapitel 9 wird das Konzept *Infrastructure as Code* vorgestellt und gezeigt, wie man mit Docker-Containern arbeitet. Kapitel 10 widmet sich den funktionalen Programmierparadigmen und stellt verschiedene Programmiersprachen vor, die häufig in Big-Data-Szenarien Anwendung finden. Kapitel 11 beschreibt das Konzept von *Polyglot Persistence*, dem Weg, Daten in mehrere unterschiedliche Datenbanken und Container zwischenzulagern. In Kapitel 12 wird Apache Kafka als Messaging-System vorgestellt, bevor es in Kapitel 13 um die Data Processing Machine Apache Spark geht und in Kapitel 14 Daten-Streaming mit Apache Flink vorgestellt wird. Kapitel 15 beschäftigt sich mit dem Thema *Data Governance* und der effektiven Verwaltung von Daten.

Das Buch schließt ab mit einem Ausblick und zwei weiteren Kapiteln, die Ihnen Lösungen zu häufigen Fehlern bei der Arbeit mit Hadoop, Hive und HBase und ergänzende Anleitungen bieten.

Die Beispieldaten zum Buch

Die Beispieldaten zum Buch können unter *https://github.com/StefanPapp/bigdata_buch* heruntergeladen werden und enthalten u. a. die fertigen Projekte, die im Buch erarbeitet werden. Nutzen Sie diese gerne als Nachschlagewerk, um Vorgehensweisen und Verwendung der entsprechenden APIs im Detail zu verstehen. Last, but not least sind online diverse Testdatensätze zu finden, die gerne während der Entwicklung und Erprobung der Anwendungen genutzt werden dürfen. Für die Daten in der Wissensdatenbank liegen im Ordner *Lizenzdateien* die Quellen der Daten vor. Die generierten Beispieldatensätze sind zufällig gewählt bzw. generiert, sodass Übereinstimmungen von Namen, Adressen, Berufen oder anderen Eigenschaften mit denen von realen Personen nur zufällig sind.

Wenn Sie Fragen oder Anregungen bezüglich dieses Buchs haben, würden sich beide Autoren freuen, wenn Sie sie kontaktieren. Jonas Freiknecht ist über *www.jofre.de* kontaktierbar. Stefan Papp erreichen Sie am besten über LinkedIn oder Xing Beide Autoren wünschen viel Spaß bei der vorliegenden Lektüre.

2 Big Data

Zu Beginn haben Sie bereits eine kurze Einführung in das Thema Big Data erhalten, die in diesem Kapitel weiter vertieft wird. Bild 2.1 zeigt eine Statistik, die eine Umfrage von Gartner (Kart, 2012) aus dem Jahre 2012 verbildlicht. Darin wurde evaluiert, welche Industrien sich mit dem Thema Big Data in welchem Ausmaß auseinandersetzen.

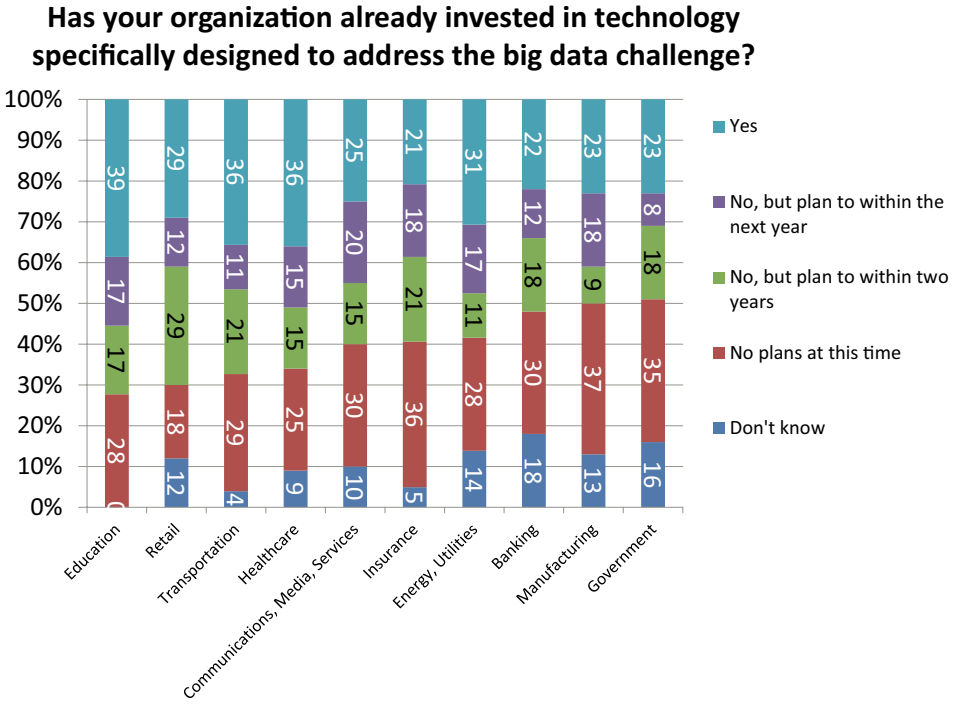

Bild 2.1 Big Data regte in 2012 das Interesse sämtlicher Industrien an.

Das wenig verblüffende Ergebnis war, dass sich alle bereits 2012 damit beschäftigt haben, allen voran die Sparte *Education*. Es gibt den berühmten Ausspruch: „Big data is like teenage sex: everyone talks about it, nobody really knows how to do it, everyone thinks everyone else is doing it, so everyone claims they are doing it.“

Es ist davon auszugehen, dass dieser Spruch 2012 stärker zutraf als heute. Viele Firmen wussten damals nicht, wie Apache Hadoop in ihren Unternehmen genutzt werden konnte. Viele Firmen erhofften sich auch eine kostenlose Datenbank, die ihr teures Data Warehouse ablösen sollte, was sich für manche später als kostspieliger Irrtum herausstellte. Es ist jedoch eine gewisse Grundeinstellung zu erkennen, die aussagt, dass Big Data schon 2012 als wichtige Neuerung wahr- und ernst genommen wurde.

2017 schaut die Lage schon anders aus. Niemand spricht mehr von einem Hype. Zahlreiche Unternehmen investieren Millionen Euro in Big Data. Einige davon gehen in die Cloud, andere investieren in eine sogenannte On-Premise-Lösung, was nichts anderes als ein eigener Cluster im eigenen Datencenter ist. In zahlreichen Big-Data-Programmen arbeiten Firmen daran, sich neue Märkte und Geschäftsfelder aufzubauen. Manche gehen sogar so weit zu behaupten, dass diverse Industrien nur über Investitionen in Big Data langfristig überleben können.

Banken, die seit Jahrzehnten mit Mainframe-Architekturen arbeiten und hohe Personalkosten durch Filialen haben, werden durch FinTechs unter Druck gesetzt, die schlanker und agiler aufgebaut sind und alle ihre Dienste online anbieten. Bessere Einsichten in Daten soll diesen Banken helfen, konkurrenzfähig zu bleiben.

Amazon weitet seine Lebensmittelsparte auch in Europa aus, und mittels Machine-Learning-Algorithmen und anderen Diensten ist es möglich, den Einkauf nahezu zu automatisieren. Das ist ein gutes Beispiel dafür, wie der Erfolg von Big Data auch von gesellschaftlichen Faktoren abhängt. Bestehende Generationen sind es noch gewohnt, im Supermarkt einzukaufen. Jedoch wächst eine neue Generation heran, die mit Online Stores aufgewachsen ist. Passt das Angebot und die Zustelllogistik, ist es durchaus denkbar, dass die sogenannten Digital Natives in ein paar Jahren auch Lebensmittel und Haushaltsprodukte fast ausschließlich im Internet einkaufen könnten und das Filialnetz der Supermarktketten ähnlich wie bei Banken zu schrumpfen beginnt.

Der Wert von Telekommunikationsdaten ist hoch, und diese können für zahlreiche Dienste verwendet werden. Irgendwann könnten die Einnahmen über die Bewegungsdaten der Kunden mehr Geld einbringen als die Einnahmen durch Mobilfunkverträge.

Wer kann sich noch eine Zukunft ohne autonomes Fahren vorstellen? Die zahlreichen Sensoren, die in Autos verbaut sind, können zudem dazu verwendet werden, Positionsdaten auszuwerten.

Es gibt de facto kaum eine Industrie, die nicht mit Big Data in Berührung ist. Manche Anwendung wirkt für viele bedrohlich, andere freuen sich darauf. Aber Big Data ist nicht mehr aufzuhalten.

Die Gartner Reports beschreiben nun oft andere Probleme als in 2012. Jeder nutzt Big Data, aber scheinbar haben viele Probleme damit, Big-Data-Systeme vom Labstatus in die Produktion zu bringen (Gardner, 2016). Auch fühlen sich viele Firmen von der Komplexität erschlagen. Früher wurde nur von Hadoop gesprochen, mittlerweile drängen neue Technologien und Verfahren auf den Markt, die teils von Hadoop unabhängig sind und deren Wert für viele Firmen erst erschlossen werden muss.

Auch wissen viele Firmen nicht, wie sie die volle Bandbreite von der Analyse der Daten bis zum Betrieb bewerkstelligen sollen. Zahlreiche Berater sprechen vom Data Swamp und der

Gefahr, dass Big-Data-Systeme ohne Governance undurchschaubar werden. Auch Datenschutz, speziell neue Gesetzesnovellen wie die GDPR, machen Firmen zu schaffen.

Bevor es in den folgenden Abschnitten an die Begriffsdefinitionen von Big Data geht, soll ein kurzer Einblick in dessen historische Entstehung gegeben und skizziert werden, wo die Big-Data-Bewegung ihren Ursprung hat.

2.1 Historische Entstehung

Einen konkreten Ursprung für den Begriff Big Data gibt es nicht. Zwar ist bekannt (Press, 2012), dass der Begriff selbst von Michael Cox und David Ellsworth verhältnismäßig früh (im Jahr 1997) öffentlich in einem Paper genannt wurde (Cox, et al., 1997), jedoch wird auf *Wikipedia* der Ursprung des Begriffs kontrovers diskutiert, und andere Quellen werden genannt (Mashey), die behaupten, dass John Mashey den Terminus *Big* bereits 1994 im Zusammenhang mit Datenmengen verwendete. Dies geschah jedoch vorwiegend als stilistisches Mittel, um die Komplexität einiger Teilbereiche der IT hervorzuheben. Darunter waren auch *Big Bandwidth*, *Big Physics*, *Big Latency* und eben auch *Big Data*, sodass der Begriff zwar gleich dem heutigen verwendet wurde, die Bedeutung jedoch nicht oder nur in Teilen übereinstimmte.

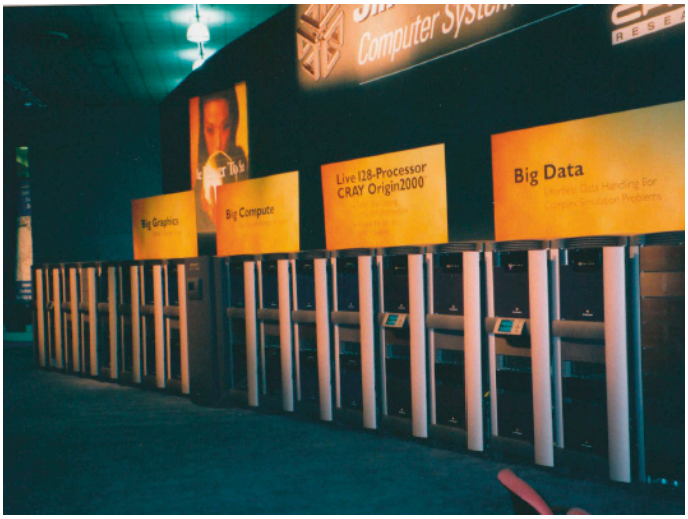

Bild 2.2 Frühe Nennung des Begriffs Big Data auf der IEEE Supercomputing Conference 1996 in Pittsburgh; Foto: Michael Woodacre

Im November 2009 entstand der erste Wikipedia-Artikel zu Big Data und wurde vom Benutzer John Blackburn prompt wieder gelöscht. Die Begründung war folgende:

> *„Delete as per nom – it is simply a combination of big and data, dictionary words which have no place here. I'm not even sure it's a neologism, and even if it was it doesn't need an article."* (John Blackburne)

Nach einigen Diskussionen, die besagten, dass *Big Band* oder *Big Bang* ebenso gelöscht werden müssten, wenn Big Data doch auch nur ein zusammengesetzter Begriff aus einem Adjektiv und einem Substantiv sei, wurde der Artikel letztendlich zugelassen und sogar später (2011) in den *Gartner Hype Cycle* aufgenommen (McBurney, 2012).

Besonders interessant an dieser Betrachtung ist, dass Doug Cutting bereits 2006 unter der Schirmherrschaft von Yahoo an Hadoop arbeitete Cutting. Die Definition des Begriffs Big Data hat also von der ersten Open-Source-Implementierung von Hadoop in 2006 bis hin zum Wikipedia-Artikel fünf Jahre benötigt, um zu reifen und um Big Data als eigenes Teilgebiet der Informatik anzuerkennen.

In den letzten Jahren hat es sich durchgesetzt, Big Data als neue Disziplin zu verstehen, die sich in erster Linie von traditionellen Business-Intelligence-Ansätzen dadurch abgrenzt, dass in Big Data auch Datenquellen für Analysen ausgewertet werden, die nicht in die traditionelle BI-Welt passen. Wir reden von Datenquellen, die kontinuierlich vorwiegend unstrukturierte (oder schemalose) Daten bereitstellen. Also von Social Networks wie Facebook bis hin zu Sensordaten, die von Maschinen in einer Produktion geliefert werden. Wie aber wird nun Big Data eigentlich definiert?

2.2 Big Data – ein passender Begriff?

In diesem Abschnitt werden zunächst verschiedene Quellen herangezogen, die Big Data aus verschiedenen Perspektiven betrachten. Ohne in die Tiefe zu gehen, lässt sich Big Data so definieren: Big Data sind Datenmengen, die zu groß für traditionelle Datenbanksysteme sind, eine hohe Halbwertszeit besitzen und in ihrer Form nicht den Richtlinien herkömmlicher Datenbanksysteme entsprechen (Dumbill,2012). Ziel ist es nun, diese Daten dennoch zu speichern und zu verarbeiten, sodass aus ihnen zeitnah wertvolle, neue Informationen gewonnen werden können. Diese neu gewonnenen Informationen können etwa passende Produktempfehlungen in E-Commerce-Lösungen sein, empfohlene Kontakte in sozialen Netzwerken oder Artikelvorschläge auf Nachrichtenseiten. Sind diese Daten nun wirklich *big* gemäß der oben gegebenen Definition?

Auf den ersten Blick wirkt es tatsächlich nicht so, denn Freundschaften können als klassische n:m-Relation gespeichert werden, Artikelvorschläge werden anhand der Tags des gelesenen Artikels erstellt, und Produktempfehlungen entstehen auf Basis der Produktkategorie der vorher betrachteten Artikel. Zieht man aber nun vom Benutzer generierte Inhalte (*User-Generated Content*) hinzu, trifft man auf Inhalte wie:

- Rezensionen und Bewertungsschreiben für Güter und Dienstleistungen
- Foren-Posts und Kommentare
- Pinnwandeinträge

- Blogeinträge
- (Wissenschaftliche) Artikel
- Tweets

Diese Daten enthalten oft subjektive Meinungen von Benutzern und Konsumenten und sind für Unternehmen, deren Geschäftserfolg vom Verstehen von Wünschen abhängt, dementsprechend wertvoll. Allerdings müssen sie vorher analysiert werden, um sie für Maschinen lesbar zu machen und somit dem Datenhalter einen Mehrwert zu liefern. Diese Datenbeschaffenheit geht mit einem Datenumfang einher, der die traditionelle Datenverarbeitung vor eine scheinbar unlösbare Aufgabe stellt. So sammelt Twitter beispielsweise 8 Terabyte Daten am Tag, Facebook 500 Terabyte, und Google verarbeitet pro Tag etwa 20 Petabyte an User-Generated Content.

Neben den von Menschen generierten Daten gibt es auch maschinengenerierten Daten. Maschinenbauer rüsten ihre Maschinen mit Sensoren aus. Eine Flugzeugturbine liefert 20 Terabytes an Daten pro Stunde (gigaom, 2010). In Produktionslinien, die kontinuierlich Rohstoffe in mehreren Schritten zu Produkten fertigen, enthält jeder Abschnitt eine hohe Zahl an Sensoren, die Druck, Temperatur und weitere Werte messen. Videodaten werden oft für die Auswertung der Qualität verwendet.

2.2.1 Die drei V

Die Frage nach der am meisten verbreiteten Definition von Big Data lässt sich wohl am ehesten durch die drei V beantworten, die der Anbieter von Marktanalysen Gartner 2001 einführte (Lancy, 2001). Zwar wurde hier noch kein Zusammenhang zu Big Data hergestellt, jedoch erkannte man bereits die zukünftigen Herausforderungen der Datenverarbeitung (hier im E-Commerce-Sektor), die den Big-Data-Begriff später prägen sollten.

Volume

Volume (deutsch: Volumen) bezieht sich auf Datenmengen. In den Neunzigern waren die ersten Homecomputer zu Beginn mit 20 Megabyte-Festplatten ausgestattet, heute gibt es mehrere Gigabyte auf USB-Sticks in Fingernagelgröße. Dieses einfache Beispiel spiegelt exponentiell wachsende Datenmengen wider. Dieses Wachstum fasst eine Studie von IDC (Gantz, et al., 2011) passend in Worte:

> *„Like our physical universe, the digital universe is something to behold – 1.8 trillion gigabytes in 500 quadrillion "files" – and more than doubling every two years."*

Datenmengen wachsen weit schneller als Datenzugriffsraten. Gleichzeitig müssen riesige Datenmengen in möglichst kurzer Zeit durchforstet werden. Festplatten haben oft durchschnittliche Datendurchsatzraten von 100 Mbit/s. Ein komplettes Durchforsten von einem Terabyte auf einem einzigen Speichermedium kann mehrere Stunden dauern. Selbst die schnellsten SSD-Laufwerke können daran nicht viel ändern.

Um die Datenmengen effizient zu verarbeiten, ist es demnach notwendig, die Ausführung zu parallelisieren. Es ist also ersichtlich, dass bei großen Datenmengen die Verarbeitung das *Big* ausmacht (Baron, 2013), nicht der Speicherbedarf (siehe Abschnitt 2.2.3).

Velocity

Das zweite V beschreibt die Geschwindigkeit, in der Daten verarbeitet werden müssen. Ein passendes Beispiel sind etwa Twitter-Meldungen oder Blog-Posts, die zu Zeiten von Wahlkämpfen verfasst werden. Dabei stellt das Internet ein starkes, aber schwer zu kontrollierendes Werkzeug dar. Bekommt ein Politiker beispielsweise eine schlechte Presse, so muss das bereitstehende Marketing-Team entsprechend zügig reagieren und diese durch korrigierende Meldungen relativieren. Die Betonung liegt auf *zügig*, denn ein einziges Gerücht zur falschen Zeit kann zu ungewollten Auswirkungen führen. So sorgte ein am 23. April 2013 geschriebener Tweet über einen Bombenanschlag auf das Weiße Haus für einen Börsencrash an der Wall Street. Bleibt man dem Beispiel der Börse treu, finden sich schnell noch weitere Szenarien, in denen die Geschwindigkeit der Erfassung von Daten eine große Rolle spielt. So sind zum Beispiel Ankündigungen über Firmenübernahmen Gold wert, wenn diese direkt nach der Bekanntgabe verifiziert werden können. Für beide Szenarien gilt: Je schneller die Auswertung stattfindet, desto höher ist der Wert der Information.

Variety

Hier kommt nun die bereits häufig angesprochene Abwesenheit von festen Strukturen und Normalisierungen zur Sprache. Das sicherlich beste Beispiel, um die Datenvielfalt zu beschreiben, mit der wir es zu tun haben, ist das Internet, das – außer vielleicht den Wiki-Seiten – keine feste Struktur vorweisen kann, aber doch einige Ähnlichkeiten aufweist, die eine maschinelle Verarbeitung ermöglichen. So können etwa bei der Analyse des HTML-Codes die Titel-Tags, z. B. *<h1>*, durchsucht werden, um eine thematische Einordnung des Inhalts vorzunehmen. Jedoch sind HTML-Seiten nicht die einzigen Daten, die verarbeitet werden. IBM befragte Mitte 2012 einige Unternehmen, die an einer Big-Data-Initiative teilnahmen, welche Quellen sie für ihre Analysen verwenden Schroeck, et al., 2012).

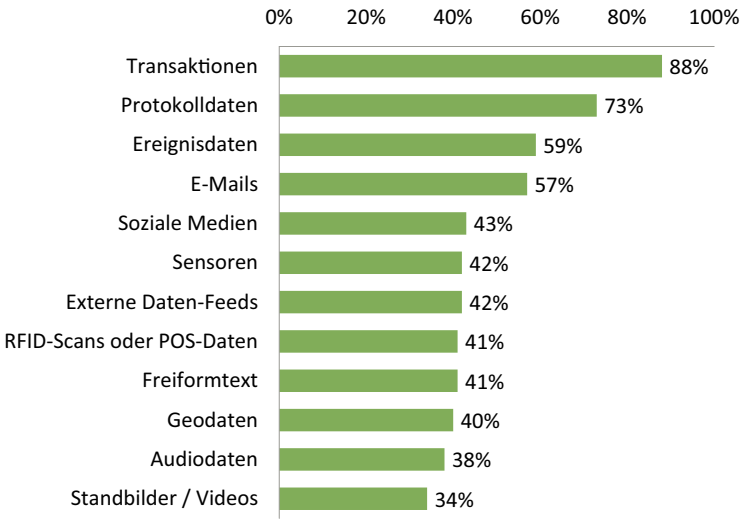

Bild 2.3 Quellen für Big-Data-Analysen auf Basis einer Umfrage von IBM im Jahr 2012

Transaktionsdaten beziehen sich dabei beispielsweise auf den klassischen Börsenhandel, in dem jeder Ver- und Einkauf gespeichert wird. Protokolldaten sind unter anderem Server-

logs, die entsprechend dem Log-Level der jeweiligen Architektur sehr groß ausfallen können und häufig dazu genutzt werden, um Klickpfade durch komplexere Anwendungen zu ermitteln und Benutzer möglichst lange im System zu halten bzw. zu einer bestimmten Aktion (Kauf, Registrierung, Empfehlung) zu bewegen. Ereignisdaten werden etwa in der Automobilindustrie protokolliert, in der Fahrzeugteile produziert, an die Logistik übergeben und verfrachtet werden. Das Feld dieser Daten ist jedoch weder auf die Automobilindustrie noch auf die Produktherstellung im Allgemeinen beschränkt, sondern beschäftigt sich in der IT mit allen Systemen, die bestimmte Ereignisse aufzeichnen und zur Auswertung bereitstellen. So können etwa die Ausschussquote einer Produktreihe überprüft und gegebenenfalls fehlerhafte Teile im gesamten Produktionszyklus, über mehrere Hersteller hinweg, ausfindig gemacht werden. Platz vier auf der Liste belegen E-Mails, die von Mail-Service-Anbietern gescannt, auf Muster von Malware oder Spam durchsucht und für gezielte Produktvorschläge für den jeweiligen Empfänger hin untersucht werden. Ob und wie dieses Vorgehen mit den Datenschutzbestimmungen des jeweiligen Landes und der Moral der Betreiber vereinbar ist, sei einmal dahingestellt. Dass soziale Medien, externe Datenfeeds und Freitextformen noch keine so starke Beachtung finden, mag daran liegen, dass Restriktionen für die Sichtbarkeit von Daten, etwa in Facebook, die Akquise erschweren oder aber, dass die Szenarien für eine Nutzung der Daten noch nicht gefunden sind, um einen produktiven Mehrwert daraus zu ziehen.

Da nun einige Quellen für Big Data vorgestellt wurden, lässt sich auch gleich auf die Vielgestalt der Formatierung eingehen. Neben klassischen, unformatierten Texten kommen häufig JSON (*JavaScript Object Notation*), XML (*Extensible Markup Language*), HTML- oder sogar Byte-Code vor. Gerade wenn man an den Aspekt der Visualisierung denkt, ist es wichtig, Relationen zwischen einzelnen Datensätzen herzustellen, um diese in Abhängigkeit voneinander zu präsentieren. Was in relationalen Datenbanken über simple Queries erreicht werden kann, bedarf bei Plain-Text-Analysen eines erheblichen Aufwands. Viel früher trifft man jedoch bei der Analyse auf die Herausforderung, die gewünschte Information aus jedem einzelnen der vielen Formate herauszufiltern. Des Weiteren gilt zu bedenken, dass sich Formate im Laufe der Zeit auch ändern können. Gerade bei der Auswertung fremder, externer Datenquellen erfolgt meist keine Benachrichtigung über eine Anpassung der Datenstruktur seitens des Datenhalters. Hier ist es wichtig, Auswertungen entsprechend zu überwachen, um Abweichungen frühzeitig festzustellen.

Ist es denn nun gerechtfertigt, von unstrukturierten Daten zu reden? Schließlich weisen ja viele Datensätze eine Struktur auf, nur eben keine feste, einheitliche. Als besserer Begriff wäre hier vielleicht polystrukturiert anzuführen, wie es der Analyst Mike Ferguson in seinem Blog beschreibt (Ferguson). Im späteren Verlauf des Buches, wenn wir zu den Eingabeformaten für die diversen Diagramme kommen, wird sich zeigen, dass diese Vielgestalt einen großen Teil der Arbeit eines Datenanalysten ausmacht, denn die Interpretation der Eingangsdaten einer Analyse variiert nur allzu häufig. Zu diesem Umstand gesellen sich ebenso Fehler in Daten, die schon vor dem Big-Data-Hype bekannt waren. Nathan Yau (Yau, 2010) benennt sechs davon wie folgt:

- Fehlende Werte
- Falsche Beschriftung
- Inkonsistenz
- Tippfehler

- Werte ohne Kontext
- Verteilte Datensätze (über mehrere Quellen hinweg)

Die Komplexität einer Verarbeitung von Daten, die in mehreren, gegebenenfalls unbekannten Strukturen vorliegen und diesen sechs Umständen unterliegen, erfordert also einen erheblichen Mehraufwand gegenüber der Aufbereitung von normalisierten Daten, z. B. aus einer relationalen Datenbank.

2.2.2 Weitere Vs

Es wurde zum Sport von Beratern, weitere Vs auf Präsentationen zu finden, um eine eigene Version der Big-Data-Vs zu generieren.

IBM z. B. führte ein viertes V (Veracity) ein, das die Richtigkeit und die Echtheit von Daten beschreibt (Zikopoulos, et al., 2013). Zwar steigt die Menge an zur Verfügung stehenden Daten nachweislich an, jedoch werden diese häufig durch generierte Inhalte verfälscht, die da sein können:

- Werbung und Spam, die eine einseitige Sicht auf Personen, Produkte oder Vorkommnisse wiedergeben.
- Per Automatismus übersetzte Texte, die häufig grammatikalische, sprachliche und inhaltliche Fehler aufweisen.
- Veraltete oder falsch kategorisierte Suchergebnisse oder Forenindizes.
- Gezielte Falschaussagen oder Fehlinformationen.

Das beste Beispiel sind die klassischen Falschmeldungen, die sich im Internet manchmal in wenigen Minuten verbreiten. So streute etwa ein junger Brite am 26. Februar 2012 das Gerücht, dass der Schauspieler Rowan Atkinson gestorben sei (Gardner, 2012). In nur drei Stunden wurde das Gerücht so schnell verteilt, dass sogar auf Wikipedia der Todestag des Schauspielers eingetragen wurde. Einmal mehr zeigt sich hier die Notwendigkeit, die gesammelten Daten vor der Nutzung zu verifizieren und auszusortieren.

Häufiger als von Veracity wird mittlerweile von Value gesprochen, um zu unterstreichen, dass Big Data erst dann Sinn macht, wenn Machine-Learning-Applikationen Zusammenhänge in den Daten erkennen können.

2.2.3 Der Verarbeitungsaufwand ist big

Ein weiterer interessanter Ansatz, den Aufwand der Verarbeitung großer Datenmengen als *big* zu sehen, liefert der Autor Pavlo Baron:

> *„Ich hatte z. B. einen Fall, bei dem es um lächerliche Datenmengen ging, die problemlos auf einen USB-Stick gepasst hätten. Man erwartete allerdings simultane Zugriffszahlen im zweifachen Millionenbereich pro Sekunde. […] das ist definitiv big […].“* (Baron, 2013)

Es ist also aus dieser Perspektive nicht die bloße Größe der Daten, sondern die Komplexität der Aufbereitung und der Informationsgewinnung. Ein gutes Beispiel sind dafür etwa Video-Streams in Kaufhäusern, die das Kaufverhalten von Kunden auswerten sollen. Auch

wenn eine einstündige Aufnahme lediglich ein paar Hundert Megabyte groß ist, ist die Schwierigkeit der Implementierung und des Trainierens von situationserkennenden Algorithmen sehr hoch und steht beispielsweise im Gegensatz zu einem einfachen Algorithmus, der lediglich alle *<h1>-Tags* aus einigen Millionen HTML-Seiten auslesen muss. Sind die notwendigen Daten extrahiert, müssen gegebenenfalls noch Beziehungen zu anderen Datensätzen hergestellt werden, etwa über einen übereinstimmenden Datumswert, Quellenübereinstimmungen oder, im Optimalfall, über vorliegende IDs. Relationale Daten hingegen verfügen über Schlüsselattribute, die eine Zuordnung von Datensätzen erheblich vereinfachen.

Roger Magoulas von O'Reilly Media gibt eine weitere sehr schöne Definition für Big Data:

„Big Data ist, wenn die Daten selbst Teil des Problems werden."

Dieses Zitat passt besonders gut, da Magoulas gar nicht erst versucht, eine Größendefinition zu geben, sondern einfach sagt, dass man, wenn die Datenmenge für *aktuelle* Verarbeitungsmethoden zu umfangreich wird, von Big Data spricht.

2.2.4 Sicht der Industrie auf Big Data

IBM befragte in der bereits in Abschnitt 2.2.1 erwähnten Studie (Schroeck, et al., 2012) mehrere Unternehmen nach deren Definition von Big Data. Eine Auswertung nach Schlagworten bestätigte weitestgehend die in den vorigen Abschnitten gegebene Sicht auf den neuen Trend. Auffällig ist, dass der Größenbegriff nicht immer im Sinne einer Datengröße verwendet wird. Stattdessen wird etwa in dieser Studie das Schlagwort *Größere Bandbreite an Informationen* verwendet, das sowohl auf große Datenmengen als auch auf mehr oder vielfältigere Informationsquellen hindeutet. 16 Prozent der befragten Unternehmen stellen neue Datenarten und Analysemethoden in den Vordergrund, was wieder für das *big* im Sinne des Verarbeitungsaufwands hindeutet (siehe Abschnitt 2.2.3).

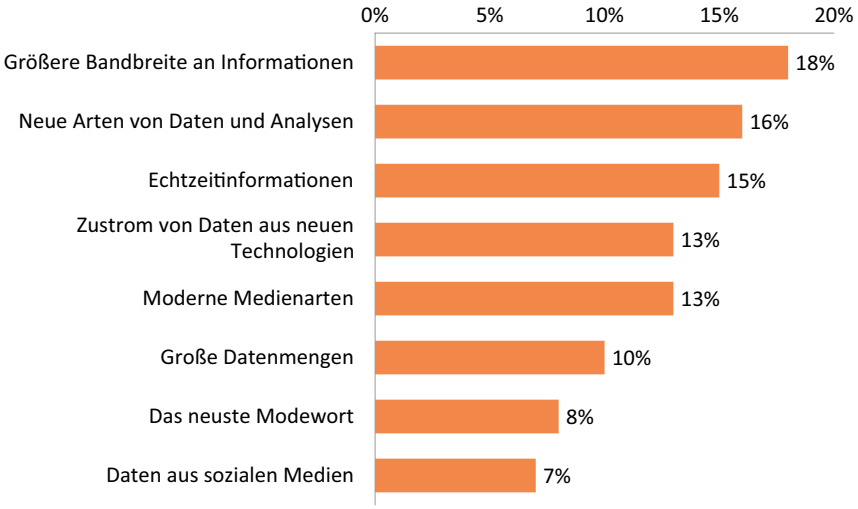

Bild 2.4 Definition von Big Data (Schroeck, et al., 2012)

2.3 Eingliederung in BI und Data Mining

Um die Begriffe BI und Data Mining in Relation zu Big Data setzen zu können, gilt es, diese im Vorfeld zu definieren. Kemper, Mehanna & Unger bezeichnen BI als Filter, der Daten in strukturierte Information umwandle (Kemper, er al., 2010). Gartner hingegen konstatiert etwas ausführlicher, dass BI ein Überbegriff für Anwendungen, Infrastruktur, Werkzeuge und Best Practices sei, die den Zugriff auf und die Analyse von Informationen ermöglichen, um Entscheidungsfindung und Performance zu erhöhen (Gartner). Hält man sich nun strikt an die Definitionen, besteht der Unterschied zwischen BI und Big Data darin, dass BI sich auf bereits vorliegende Informationen bezieht, die dazu noch strukturiert sind und sich auf einen eindeutigen Kontext beziehen.

Das Ziel von BI und der Big-Data-Explorationen ist jedoch dasselbe, nämlich aus vorhandenen Daten neue Erkenntnisse zu gewinnen, die der Entscheidungsfindung bei vorher definierten Fragestellungen dienen. Der Trend bei Big Data geht dabei auch oft in die Richtung zu lernen, was man noch nicht weiß.

BI ist jedoch mittlerweile mehr als diese einfache Begriffsdefinition. Es hat sich in den letzten Jahren zu einem festen Prozess samt einem Set aus technischen Werkzeugen entwickelt, um das Berichtswesen in Unternehmen zu automatisieren. Dazu gehören die Datenaufbereitung, die Datenspeicherung in DWHs sowie deren Darstellung aus verschiedenen Perspektiven.

Welche Techniken, Methoden und Arbeitsschritte werden aber nun angewandt, um Informationen aus vorliegenden Daten zu extrahieren? Die Antwort darauf gibt der sogenannte KDD-Prozess (*Knowledge Discovery in Databases*).

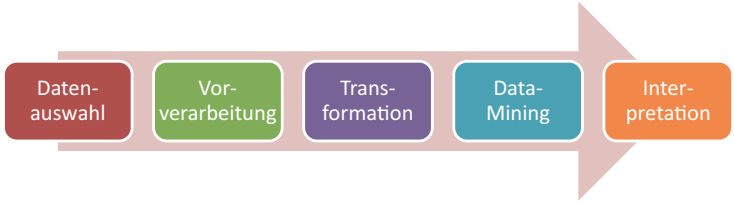

Bild 2.5 Der KDD-Prozess nach (Kononenko, et al., 2007)

Der (iterative und interaktive) KDD-Prozess hat das Ziel, gültige, neue, nützliche und verständliche Muster in vorhandenen Daten zu erkennen (Fayyad, et al., 1996). Wirft man nun einen Blick auf den vierten Schritt des in Bild 2.5 illustrierten Ablaufs, so ist zu erkennen, dass Data Mining einen Teil des KDD-Prozesses darstellt. Dieser nimmt gesäuberte, korrigierte und transformierte Daten entgegen, extrahiert daraus Muster und Zusammenhänge und gibt diese zur Interpretation frei. Quellen müssen, anders als der Begriff KDD vermuten lässt, nicht zwingend Datenbanken sein, sondern können auch als simple Datensätze gesehen werden, z. B. als Flat Files, CSV, XML oder Dateisystemstrukturen. Wichtig ist, dass diese bereits im Vorfeld aufbereitet wurden. Zu dieser Aufbereitung (*Preprocessing*) gehören:

- Formatanpassungen (z. B. Datums- und Zahlenformate)
- Korrigieren von Ausreißern (Messfehler, Verarbeitungsfehler, bewusste Falschangabe)
- Auffüllen dünn besetzter Daten

HINWEIS: In späteren Kapiteln wird der KDD-Prozess im Detail durchgegangen.

Stellt man nun die drei Begriffserklärungen BI, Data Mining und Big Data einander gegenüber, so erkennt man schnell einige Gemeinsamkeiten sowie Unterschiede.

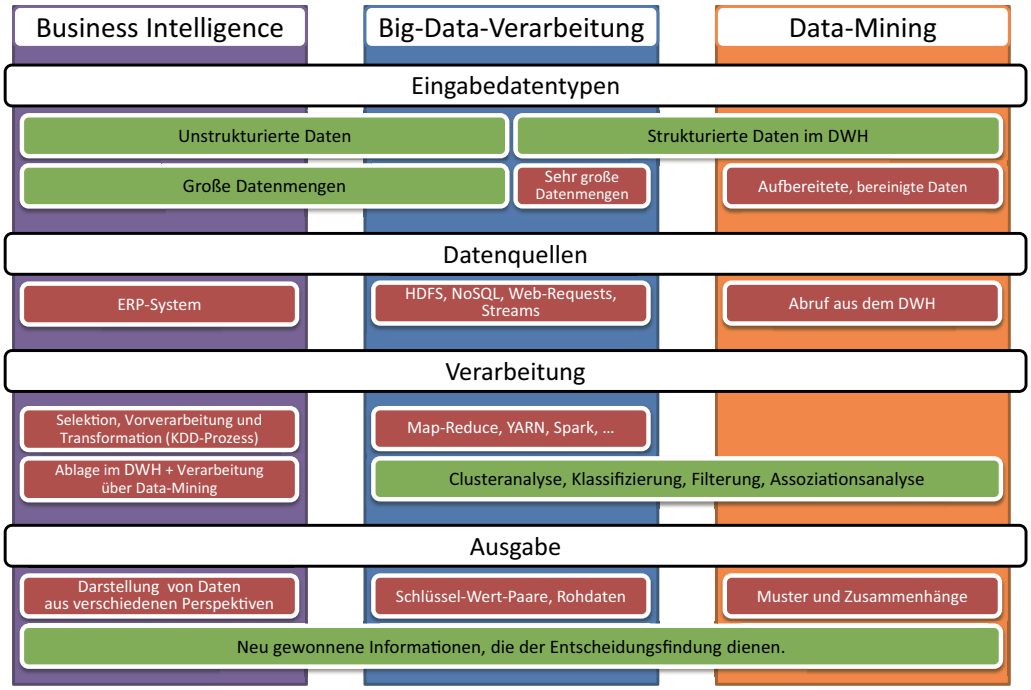

Bild 2.6 Definitionsvergleich von BI, Big Data und Data-Mining

Vor Big Data wurden größere Analysen fast ausschließlich in Datenbanken ausgeführt. Es gab dazu auch eine Königsklasse, das MPP (Massive Parallel Processing), das eine parallele Verarbeitung zuließ. Im Kern folgte aber jedes Data Warehouse den Prinzipien der relationalen Datenbanken.

Bild 2.7 Das Data Warehouse

Das obige Beispiel illustriert den Prozess in den DWHs. Daten werden aus ihren Quellen entnommen, in sogenannten ETL-Tools aufbereitet und als relationale, strukturierte Daten im DWH abgelegt. Vom DWH werden Extrakte in Data Marts abgelegt, auf die dann Analysten zugreifen, um Ergebnisse abzuleiten.

In der ersten Evolutionsstufe von Hadoop empfahlen manche DWH-Systemhersteller, Hadoop als billigen Speicherplatz zu nehmen. Hadoop würde dann die Rolle von ETL übernehmen, die Daten aufbereiten und ins DWH laden. Die Auswertung erfolgte dann im DWH. Für Spezialfälle boten die Systemhersteller auch Möglichkeiten an, sogenannte Kaltdaten aus Kostengründen auf Hadoop zu belassen und sie bei Bedarf ins DWH zu laden.

Der Vorteil vom DWH wurde so angepriesen, dass bekannte analytische Verarbeitungsmethoden (Stichwort *Online Analytical Processing Cube*) und eine reichhaltige Funktionsbibliothek zur Verfügung stehen. Auch steckte jahrelanges Know-how in Komponenten wie der SQL Engine des DWHs. SQL auf Hadoop war zumindest am Anfang extrem langsam und hatte bei Joins eine schlechte Performance.

Auch die Rolle von sogenannten Self-Service BI Tools (wie Tableau) und vor allem Excel darf nicht unterschätzt werden. Soll also in einer Grafik der Gewinn eines Unternehmens inklusive aller Tochterfirmen angezeigt werden, müssen im Reporting-Werkzeug lediglich vom Benutzer die bereits ermittelten Gewinne der Tochterfirmen dem Gesamtgewinn hinzuaddiert werden. Um das zu bewerkstelligen, gibt es Konnektoren zu verschiedenen Datenlieferanten.

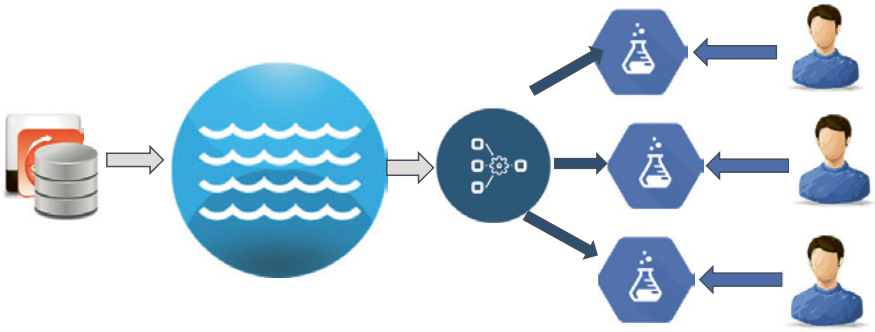

Bild 2.8 Der Data Lake

Mit Hadoop hat sich vieles geändert. Daten werden sofort auf den Hadoop-Cluster geladen. Wir sprechen hier auch vom Data Lake. Man sprach von ETL (Extract Transform Load), wenn die Daten vor der Beladung vom DWH noch transformiert wurden, bevor sie abgelegt werden konnten,

Im Falle des Data Lakes spricht man aber von ELT (Extract Load Transform). Daten werden aus den Quellsystemen gezogen, auf den Data Lake geladen und schließlich in ein Zielsystem transformiert.

Bei der Aufbereitung von Daten werden die unstrukturierten Daten oft in ein strukturiertes Schema überführt. Daten, die in Apache Hive-Tabellen abgelegt werden, können auch als Datenlieferanten von Self-Service BI Tools wie Tableau dienen.

Mit der Zeit wurde Hadoop zu einer Datenplattform, die auf einem verteilten Dateisystem aufbaut und zahlreiche Möglichkeiten bietet, die Daten aufzubereiten. Diese Daten können in Strukturen gebracht und gleichzeitig analysiert werden.

Die Reise war damit aber noch nicht zu Ende. Das komplette Open-Source-Ecosystem von Big Data bietet mittlerweile zahlreiche Werkzeuge an, die die Verarbeitung von Daten noch weiter vereinfachen und neue Methoden wie die Echtzeitverarbeitung mit sich bringen.

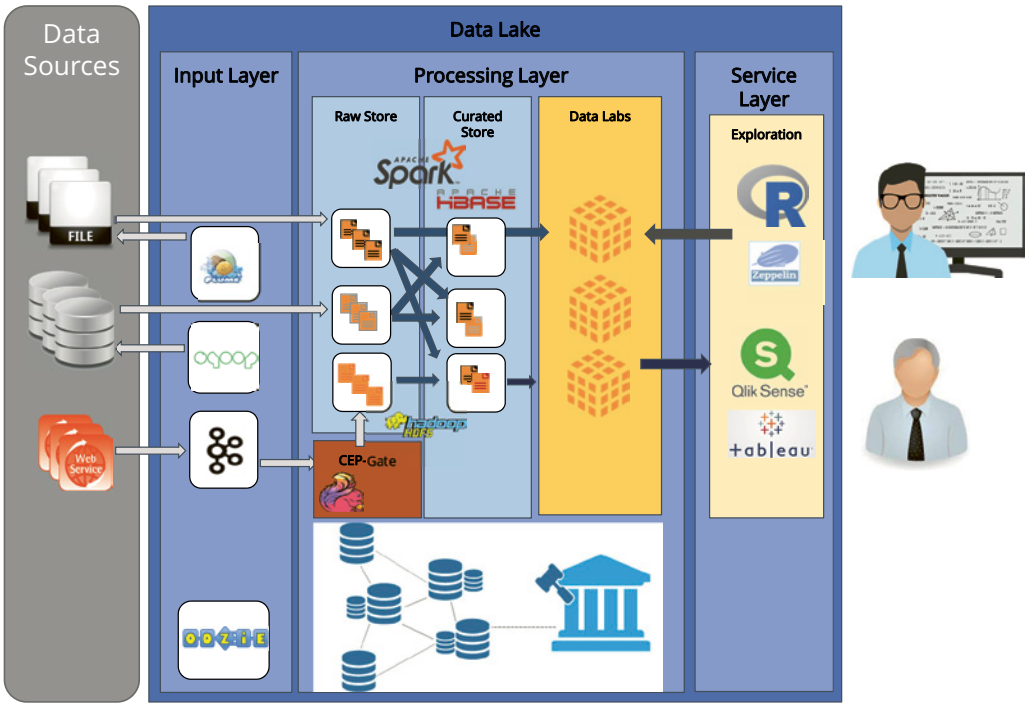

Bild 2.9 Big-Data-Referenzarchitektur

Bild 2.9 zeigt eine komplette Referenzarchitektur mit Big-Data-Komponenten. Es landen unterschiedliche Daten auf einer Datenplattform (in diesem Beispiel Hadoop). Sind die Daten auf Hadoop geladen, werden sie für verschiedene Szenarien unterschiedlich weiterverarbeitet und dann in Tabellen abgelegt, auf die dann analytische Werkzeuge zugreifen können.

Architekturen wie die Lambda- und Kappa-Architektur zeigen, dass man auch Architekturmuster bauen kann, die auf Echtzeitverarbeitung aufbauen.

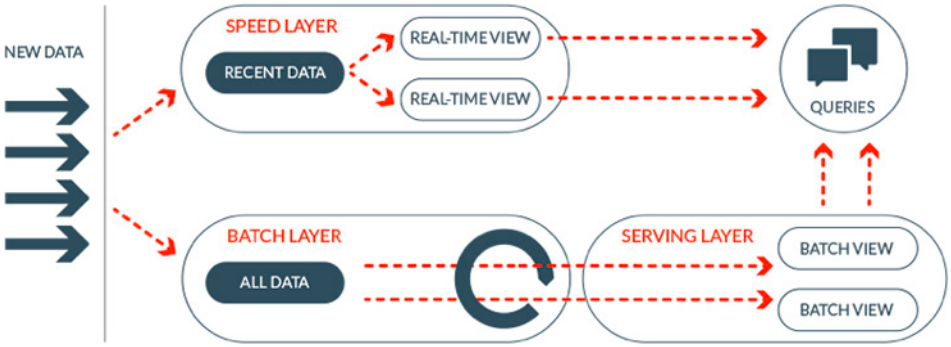

Bild 2.10 Big-Data-Architekturen

Zusammengefasst kann man sagen, dass in in Lambda-Architekturen Daten beim Laden in zwei Bereiche geteilt werden. Im Speed Layer werden die Daten im Regelfall in Memory gehalten. Im Batch Layer werden alle Daten archiviert. Verschiedene Prozesse verarbeiten die Daten kontinuierlich und bereiten die Ergebnisse im Serving Layer für einen schnellen Zugriff auf. Details können im Buch von Nathan Marz nachgelesen werden (Marz, 2015).

Ein weiteres zentrales Thema, das im Umfeld von Big Data anzutreffen ist, ist Data Governance. Dieser Begriff umfasst die Verwaltung von Daten, konkret geht es um Katalogisierung, Klassifikation und weitere Prozesse wie z. B. Retention, um Daten nach einem bestimmten Ablauf auch wieder zu löschen. In Data Governance spielen auch Themen wie Sicherheit und Datenqualität hinein.

Ein weiteres Thema sind natürlich alle Methoden, um Daten zu analysieren. Nicht umsonst hat der Beruf des Data Scientist einen hohen Stellenwert bekommen. Auch gibt es mittlerweile zahlreiche Kurse zu Deep Learning und Machine Learning.

Als dieses Buch in der ersten Version geschrieben wurde, wurden fast ausschließlich Hadoop-Technologien beschrieben. In einem Interview erklärte Ted Dunning, Chief Architect von MapR, dass Open-Source-Technologien deswegen so erfolgreich sind, weil sie über saubere Schnittstellen verfügen. Einzelne Komponenten wären somit leichter austauschbar.

Die Konsequenz daraus ist, dass viele Open-Source-Big-Data-Systemlandschaften wie ein Baukasten aufgebaut sind, wo einzelne Komponenten austauschbar sind. Das trifft auch auf Hadoop zu. MapR hat beispielsweise ein eigenes Dateisystem geschrieben, das auf der HDFS-API aufbaut und mehr Funktionalität aufweist als HDFS. Neue Komponenten wie Apache Spark nutzen nur mehr Schnittstellen und können Daten auf HDFS schreiben, aber auch auf viele andere Plattformen wie Cassandra oder Amazon S3. Die zentrale Botschaft ist, dass Hadoop in diesem Buch zwar nach wie vor einen hohen Stellenwert hat, aber dieses Buch kein reines Hadoop-Buch ist, sondern viele Technologien vorstellt, die man mit Hadoop verwenden kann, die aber im eigentlichen Sinne von Hadoop unabhängig sind.

3 Hadoop

In diesem Kapitel wird der Bogen von der anwendungsbezogenen hin zur technischen Betrachtung geschlagen. Man kann behaupten, Hadoop diene dazu, Big Data zu verarbeiten, und danach kann man damit beginnen, diese Behauptung anhand von Beispielen umzusetzen. Wer einen Blick auf die Apache Hadoop-Webseite wirft, wird jedoch feststellen, dass der Begriff *Big Data* nicht verwendet wird, sondern von verteilter Datenverarbeitung gesprochen wird.

Hadoop ist darauf angelegt, große Datenmengen verteilt zu verarbeiten. Skalierbarkeit und Zuverlässigkeit stehen dabei im Vordergrund. Sprechen wir von Hadoop, sprechen wir im Grunde von einem verteilten Dateisystem (HDFS) und einem Standardverfahren, die Daten zu verarbeiten (Map Reduce). Analytische Verfahren wie Machine Learning sind nicht Teil des Hadoop-Kerns, wohl aber gibt es eigene analytische Open-Source-Komponenten, die Hadoop als Datenplattform nutzen, auf Hadoop aufbauen und auch Teil des Hadoop Ecosystems sind. Wir werden in späteren Kapiteln zeigen, dass es sowohl Frameworks gibt, die ausschließlich für den Einsatz mit Hadoop entwickelt wurden, als auch solche, die zwar Hadoop unterstützen, aber auch auf anderen Datenplattformen laufen.

Seit der ersten Auflage des Buches hat sich der Big-Data-Markt rasant weiter entwickelt. Galt Hadoop vor drei Jahren noch als Synonym für eine Big-Data-Plattform, so ist die Welt vielfältiger geworden, und mittlerweile haben auch andere Plattformen an Bedeutung gewonnen. Es macht dennoch Sinn, Hadoop von Grund auf vorzustellen und hier tief in technische Details zu gehen, denn auch wenn Hadoop Konkurrenz bekommen hat, ist es nach wie vor die vielleicht wichtigste und bekannteste Komponente im Big Data Ecosystem.

3.1 Hadoop kurz vorgestellt

Hadoop ist ein größtenteils in Java geschriebenes Framework zum verteilten Speichern von Daten und zu deren paralleler Verarbeitung auf *Commodity-Hardware*. Dabei wird Hadoop im Regelfall in einem horizontal skalierbaren Cluster betrieben, dem einfach weitere Knoten hinzugefügt werden können, sodass schnell eine Vergrößerung oder Verkleinerung

erfolgen kann. Große Unternehmen wie Yahoo betreiben Cluster mit über 4000 Knoten[1]. Statt auf teure Hardware zu setzen, ist Hadoop darauf ausgelegt, günstige Systeme einzusetzen. Diese als Commodity-Hardware bezeichneten Geräte sind verhältnismäßig günstig, leicht zu beziehen und leicht auszutauschen. Statt der Anschaffung neuer, schnellerer Hardware (*Scale Up*) wird also beim Betrieb von Hadoop vielmehr die Erweiterung des Clusters (*Scale Out*) um weitere Knoten empfohlen.

Apache Hadoop besteht maßgeblich aus drei Komponenten:

- **Hadoop Distributed File System (HDFS)**: Ein über den gesamten Cluster verteiltes Dateisystem für die am Cluster abzulegenden Daten.
- **Hadoop Map Reduce**: Ein Programmierframework zur verteilten Verarbeitung von Daten gemäß der zweiphasigen Verarbeitung durch Mapper und Reducer.
- **YARN**: Eine Plattform, die Ressourcen auf dem Hadoop-Cluster verwaltet. YARN legt unter anderem fest, welche Prozesse welchen Ressourcen zugeteilt und wie Jobs abgearbeitet werden.

Ein wesentliches Merkmal von Open-Source-Werkzeugen sind stabile und klar definierte Schnittstellen. Einzelne Komponenten müssen modular gestaltet und gegebenenfalls auch austauschbar sein. Somit besteht die Möglichkeit, selbst einzelne dieser drei Kernkomponenten auszutauschen.

HINWEIS: Ein Beispiel für ein System, das HDFS als Kernkomponente ausgetauscht hat, ist MapR. MapR-XD (vormals als MapR-FS bezeichnet) ist ein verteiltes Dateisystem und kann über die HDFS-API angesprochen werden.
Die Implementierung von MapR-XD unterscheidet sich von HDFS und ist beispielsweise darauf optimiert, auch viele kleine Dateien effizient zu verarbeiten.

YARN kann mit Mesos ausgetauscht werden. Mesos verwaltet wie YARN die Ressourcen eines Clusters und unterscheidet sich in den Details, wie Jobs verarbeitet werden.

Alternativ zu Map Reduce können alternative Frameworks wie Apache Spark oder Apache Tez verwendet werden. Apache Spark ist mittlerweile die Nummer 1 bei den Data Processing Engines, worauf in späteren Kapiteln eingegangen wird.

Eine große Rolle spielt bei Hadoop das *Konzept der Datenlokalität*. Anders als in traditionellen datenverarbeitenden Applikationen, in denen die Daten dem Programm zur Verfügung gestellt werden, wird bei Hadoop der Programmcode auf dem Cluster zu den Datenknoten geschickt, um die Notwendigkeit des Datentransports zu minimieren. In einem System, das für große Datenmengen geschaffen ist, macht das Prinzip durchaus Sinn, denn die Programmlogik stellt einen Bruchteil der Datenmenge von Big-Data-Daten dar.

[1] Referenzzahlen für Unternehmen, die Hadoop einsetzen, finden Sie unter: http://wiki.apache.org/hadoop/PoweredBy

HINWEIS: Der **Data Lake** gilt als primäres Architekturmuster für Big-Data-Plattformen. Die Stellung eines Data Lakes kann durchaus mit der Stellung eines Data Warehouses (DWH) für die relationale Datenbankwelt verglichen werden.

Der Data Lake ist ähnlich wie Big Data ein Marketingbegriff. Er kann in Details unterschiedlich aufgefasst werden. Der Kern eines Data Lakes sind Best Practices, wie Daten in einen Big-Data-Cluster geladen werden und wie sie darauf weiterverarbeitet werden.

Der wesentlichste Unterschied zum traditionellen Data Warehouse liegt darin, dass Daten, nachdem sie aus Datenquellen extrahiert wurden, erst geladen und dann transformiert werden (ELT - Extract Load Transform). Im DWH gilt das ETL-Prinzip (Extract Transform Load): Daten müssen vor der Beladung in das passende Schema transformiert werden.

Neben diesen Beladungspraktiken ist auch Data Governance ein zentrales Thema, welches in einem späteren Kapitel noch genauer erklärt wird. Data Governance bedeutet, dass firmentaugliche Prozesse und Policies definiert werden, wie mit Daten umzugehen ist. Auf firmenspezifische Regeln und Policies wird in diesem Buch nicht eingegangen, aber die Wichtigkeit dieses Feldes darf – vor allem in Bezug auf die neue Datenschutzgrundverordnung (DSGVO) – nicht unterschätzt werden.

Wichtig ist, Hadoop nicht mit einem Data Lake gleichzusetzen. Ein Data Lake ist ein Konzept, das auch für andere Plattformen angewandt werden kann, die mit **Schema on Read** Use Cases verwendet werden.

Bevor nun auf die Neuerungen in Hadoops Version 2 eingegangen wird, sollen das HDFS und dessen Konzept erklärt werden.

3.2 HDFS – das Hadoop Distributed File System

In der Vergangenheit wurden aufbereitete Daten größtenteils in relationalen Datenbanken gespeichert. Eine wesentliche Voraussetzung für Inserts in eine Tabelle ist, dass die hinzuzufügenden Daten dem vorgegebenen Schema der Tabelle entsprechen. Ein häufig verwendeter Fachbegriff für dieses Verfahren ist **Schema on Write.**. Es muss deswegen betont werden, dass bei strukturierten Daten immer von Daten gesprochen wird, die erfolgreich gegen ein Schema validiert werden können.

Big-Data-Plattformen zielen darauf ab, auch Daten verarbeiten zu können, die unstrukturiert (also schemalos) sind. Wie bereits im Vergleich von Data Warehouse und **Data Lake** erwähnt, werden Daten in eine Big-Data-Plattform unverarbeitet oder roh abgelegt und dann später verarbeitet. Dieses Verfahren **(Schema on Read)** führt dadurch zu schnellen Schreib-

zugriffen. Bei der späteren Verarbeitung von unstrukturierten Daten steht auch die Parallelisierung zur Verfügung.

Ein Dateisystem kann auch als universeller Ablageort für alle Formen von Daten verstanden werden. Da es kein einheitliches Format für Dateien gibt, hängt die Verarbeitung der Daten von den Programmen ab, die diese Dateien laden. Dem Dateisystem selbst ist es egal, ob XML, DOC oder sonstige Daten in den Verzeichnissen gespeichert werden.

Mit Sicherheit werden Sie als Leser mit Dateisystemen vertraut sein und sind bestens geübt, die traditionellen Funktionen wie das Kopieren und Löschen von Dateien anzuwenden. Aus diesen Funktionen haben sich Standards entwickelt wie POSIX, zu dem fast jedes Dateisystem kompatibel ist. Bekannte Vertreter lokaler Dateisysteme sind NTFS oder ext4.

Ziel von verteilten Dateisystemen ist es im Regelfall, dieselben Standards der lokalen Dateisysteme zu nutzen. Dazu gehört beispielsweise die API, um Dateien zu verschieben oder zu löschen. Im Hintergrund werden aber die Operationen parallelisiert und auf verteilten Knoten ausgeführt.

HDFS ist demnach ein verteiltes (distributed) Dateisystem, das weitgehend POSIX-kompatibel ist. Das Kernkonzept basiert auf dem von Google im Jahr 2003 vorgestellten *GFS* (*Google File System*) (Ghemawat, Gobioff & Leung, 2003).

Folgende Anforderungen wurden während des Designs an das Dateisystem gestellt:

- Betrieb auf Commodity-Hardware
- Ausfallsicherheit einzelner Knoten
- Speicherung und Verarbeitung großer Datenmengen
- Einfache Skalierbarkeit

Ein Masterknoten, genannt *Name Node*, verwaltet alle Metadaten des Dateisystems, darunter Verzeichnisstrukturen, Dateien und Dateizugriffe der Clients (Apache Software Foundation, 2013). Parallel dazu existieren mehrere *Data Nodes*, die den Speicher verwalten, der den entsprechenden Knoten im Cluster zugeordnet ist. Das HDFS bietet einen Set an Funktionen an, der es erlaubt, Daten in das Dateisystem zu schreiben und auszulesen. Es ist nicht nötig, eine eigene Partition für ein HDFS anzulegen, denn es setzt auf ein existierendes Dateisystem auf, z. B. das gängige *ext4* (*Fourth Extended Filesystem*). Ein signifikantes Merkmal ist, dass die Blockgröße in einem HDFS im Durchschnitt 64 bis 128 Megabyte beträgt. Traditionelle Dateisysteme arbeiten mit 1 bis 64 Kilobyte großen Blöcken.

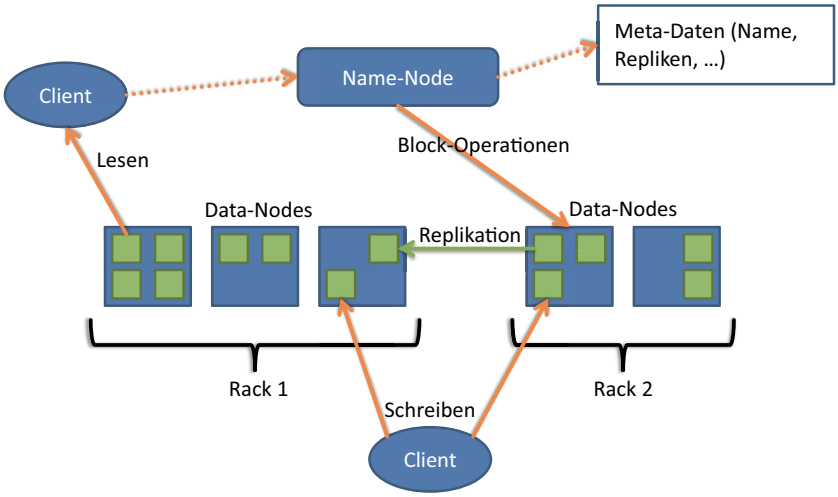

Bild 3.1 Architektur und Funktionsweise des HDFS

Erhält der *Name Node* nun vom Client eine Datei, die im Dateisystem abgelegt werden soll, benötigt dieser zwei weitere Informationen: erstens die eben genannte Blockgröße, in die er die Datei aufteilen soll, und zweitens die Anzahl der Repliken, die über den Cluster verteilt werden. Darauf werden vom *Name Node* so viele *Data Nodes* herausgesucht, wie der Client durch die Replikenanzahl fordert. Deren Adressen werden an den Client zurückgeliefert, sodass dieser mit dem Beschreiben der *Data Nodes* beginnen kann. Dabei wird nur ein *Data Node* beschrieben. Dieser gibt dann die Daten an die anderen Knoten weiter. Für eine effiziente Übermittlung sortiert der *Name Node* die *Data Nodes* vor der Übergabe nach bestimmten Parametern (höchster Datendurchsatz, beste Netzanbindung, geringster momentaner Work-Load), sodass der Client weiß, welcher *Data Node* am besten erreichbar und geeignet ist. Am häufigsten werden drei Repliken für einfache Setups angelegt. Diese Anzahl kann jedoch zu jeder Zeit per Konfiguration erhöht oder verringert werden. Eine sehr anschauliche Erklärung für das HDFS liefert *Maneesh Varshney* in einem eigens dafür gezeichneten Comic[2].

Wie werden nun aber Dateioperationen auf dem Dateisystem ausgeführt? Einer der meistgenutzten Wege ist sicherlich die Kommandozeile. Hadoop bietet dafür parametrisierbare Anwendungen an, die etwa dabei helfen, Dateien aus dem lokalen Dateisystem in das HDFS zu kopieren (oder vice versa), die Lese- und Schreibrechte der Dateien zu ändern oder neue Verzeichnisse im HDFS anzulegen bzw. ungewünschte zu löschen.

Befehlen, über die man das Dateisystem anspricht, wird ein *hdfs* vorangestellt. Als zweiter Parameter wird die Art des Zugriffs festgelegt. Der Bezeichner *dfs* entspricht hierbei einem Standardbefehl. Alternativ kann z. B. *dfsadmin* für administrative Befehle verwendet werden. Die Befehle werden nun vorgestellt.

Für den späteren, praktischen Teil werden einige dieser Operationen angewendet. Dazu werden hier einige Beispiele zu geläufigen Operationen vorgestellt.

[2] http://de.slideshare.net/jaganadhg/hdfs-10509123

Dateien vom lokalen Dateisystem auf das HDFS kopieren

```
hdfs dfs -copyFromLocal QUELLE ZIEL
hdfs dfs -copyFromLocal /usr/input/test.txt /hdfs/input/test.txt
```

Dieses Kommando legt die Datei *test.txt* im Ordner */hdfs/input/* im HDFS ab. Alternativ zu `copyFromLocal` kann auch der Parameter `put` verwendet werden.

Dateien vom HDFS in das lokale Dateisystem kopieren

```
hdfs dfs -copyToLocal QUELLE ZIEL
hdfs dfs -copyToLocal /hdfs/input/test.txt /usr/input/test.txt
```

Kopiert die Datei *test.txt* aus dem HDFS in das lokale Dateisystem unter */usr/input/*. Alternativ zu `copyToLocal` kann auch der Parameter `get` verwendet werden.

Alle Dateien und Ordner in einem HDFS-Ordner auflisten

```
hdfs dfs -ls ORDNER
hdfs dfs -ls /
```

Listet alle Dateien und Ordner im Hauptverzeichnis des HDFS auf.

Erstellt einen neuen Ordner im HDFS

```
hdfs dfs -mkdir [-p] ORDNER
hdfs dfs -mkdir /input/
```

Erstellt den Ordner *input* im Hauptverzeichnis des HDFS. Ist das Argument *-p* gesetzt, so werden auch alle nicht existenten Überordner mit erzeugt.

Kopiert eine Datei im HDFS

```
hdfs dfs -cp QUELLE ZIEL
hdfs dfs -cp /hdfs/input/test.txt /hdfs/input/test2.txt
```

Legt eine Kopie der Datei */hdfs/input/test.txt* als */hdfs/input/test2.txt* an.

Verschiebt eine Datei im HDFS

```
hdfs dfs -mv QUELLE ZIEL
hdfs dfs -mv /hdfs/input/test.txt /hdfs/input2/test.txt
```

Verschiebt die Datei *test.txt* aus */hdfs/input/* nach */hdfs/input2/*.

Löschen von Dateien und leeren Verzeichnissen

```
hdfs dfs -rm LEERER_ORDNER_ODER_DATEI
hdfs dfs -rm /hdfs/input/test.txt
```

Löscht die Datei *test.txt*.

Rekursives Löschen von Verzeichnissen

```
hdfs dfs -rm -r ORDNER
hdfs dfs -rm -r /hdfs/
```

Löscht den Ordner */hdfs/* und alle Unterverzeichnisse und -dateien. Achten Sie darauf, dass das Argument *-r* gesetzt ist. Früher wurde der Befehl separat als *rmr* geführt, ist jedoch seit Kurzem als *deprecated* gekennzeichnet.

Erstellen einer leeren Datei

```
hdfs dfs -touchz DATEI
hdfs dfs -touchz /hdfs/input/test3.txt
```

Erstellt eine leere Datei mit Namen *test3.txt*.

Ausgeben des letzten Kilobytes einer Datei

```
hdfs dfs -tail [-f] DATEI
hdfs dfs -tail -f /hdfs/input/test.txt
```

Gibt die letzten Kilobyte der Datei *test.txt* aus. Ist das Argument *-f* angegeben, so wird jede Änderung am Ende der Datei ausgegeben, bis der Prozess mit **STRG+C** beendet wird.

TIPP: In früheren Versionen waren alle HDFS-Operationen über den Befehl *hadoop fs* abrufbar (also z. B. `hadoop fs -mkdir /xyz/`). Seit Hadoop 2 sind sie jedoch unter der Anwendung *hdfs* zu finden, funktionieren jedoch unter *hadoop fs* immer noch. Allerdings werden Anwender über einen *Deprecated-Vermerk* auf diesen Umstand hingewiesen.

Es muss jedoch betont werden, dass MapR, einer der drei großen Hadoop-Distributoren, weiterhin die alte Notation verwendet und die Schnittstelle mit `hadoop mfs` für MapRFS-spezifische Abfragen erweitert hat.

Neben der Kommandozeile existieren zahlreiche APIs in unterschiedlichen Programmiersprachen, mit denen HDFS-Operationen ausgeführt werden können. Auch weist jede der grafischen, webbasierten Administrationsoberflächen für Hadoop, die von den Distributoren bereitgestellt werden, wie Apache Ambari oder Cloudera Navigator, ein HDFS-Werkzeug auf. Eine weitere Möglichkeit ist, HttpFS[3] zu nutzen, um über eine REST-API auf das HDFS zuzugreifen.

Wem das Schreiben von *hdfs dfs* zu umständlich ist, der kann auch hdfs direkt in ein bestehendes Dateisystem mounten, sodass Dateimanager direkt darauf zugreifen können.

[3] http://hadoop.apache.org/docs/r2.2.0/hadoop-hdfs-httpfs/ServerSetup.html

3.3 Hadoop 2.x und YARN

Seit 2013 existiert die Version 2 von Hadoop. Nachdem 2.1 noch als Beta veröffentlicht wurde, folgte dann am 15. Oktober 2013 die Version 2.2 als *Stable Release*. Die Gründe für eine Erneuerung des Kerns des Frameworks waren z. B. eine Limitierung der Clustergröße auf etwa 4500 Knoten und die Identifizierung des *Name Nodes* von HDFS als *Single Point of Failure*.

Die wesentlichste Änderung war jedoch die Entkopplung von Map Reduce als exklusive Datenverarbeitungskomponente. In Version 1 war Hadoop eine reine Map-Reduce-Umgebung. Applikationen mussten Map Reduce verwenden, um Daten zu verarbeiten. Map Reduce war zuständig dafür, wie die Daten abgearbeitet werden, und parallel dazu auch dafür, welche Ressourcen-Jobs zur Verfügung gestellt wurden.

Map-Reduce-Applikationen zu schreiben, war nicht trivial. Hierfür war es notwendig, von zwei Java-Klassen abzuleiten und eine sogenannte Treiber-Klasse zu schreiben. Es gibt zwar mehrere Frameworks, die den Zugriff abstrahierten und eine einfachere API bereitstellten. Aber im Endeffekt wurden bei Hadoop 1 diese Programme im Hintergrund auch zu Map Reduce Jobs umgewandelt.

YARN (*Yet Another Resource Negotiator*) wurde als Zwischenschicht mit Hadoop 2 eingeführt und spaltet die Ressourcenverwaltung von Map Reduce ab. Die Aufgabe von YARN ist es, eingehenden Jobs auf einem Hadoop Cluster Ressourcen zuzuteilen und sicherzustellen, dass die Jobs abgewickelt werden. Dazu gehört auch, dass einzelne Tasks neu starten, sollten diese abgebrochen werden.

So bleibt Map Reduce zwar bestehen, ist aber nur eines von mehreren möglichen Programmiermodellen. YARN hat seinen Namen daher, dass es als Zwischenschicht alle im Cluster verfügbaren Ressourcen wie CPU, RAM und Speicher verwaltet und den Anwendungen zur Verfügung stellt. Für die Entwickler unter uns ist als weiterer Vorteil zu nennen, dass viele Funktionen von YARN über REST Services angesprochen werden können (siehe Abschnitt 3.10). Die Basisarchitektur von Hadoop ist jedoch weitestgehend dieselbe geblieben, sodass denjenigen, die bereits mit Hadoop 1.x gearbeitet haben, der Umstieg leicht gemacht wird.

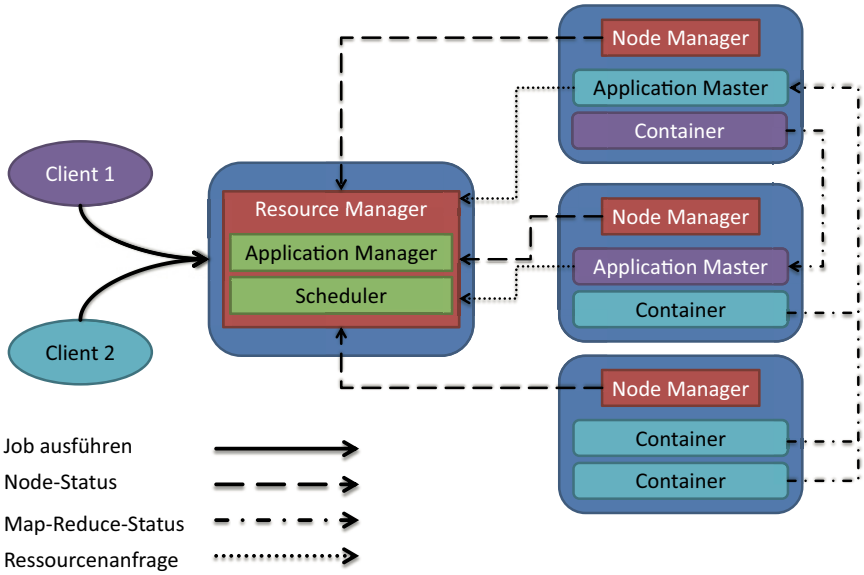

Bild 3.2 Architekturübersicht zu YARN

Wer sich bereits mit der Architektur von Hadoop 1.x beschäftigt hat, wird den *JobTracker* vermissen, der in YARN in *Resource Manager* und *ApplicationMaster* unterteilt wird. Wie in der Abbildung zu sehen, ist der *Resource Manager* eine globale Komponente, die nur auf dem Master zu finden ist. Vom *ApplicationMaster* hingegen existiert pro Anwendung eine Instanz auf einem beliebigen Knoten im Cluster. Der *Resource Manager* verwaltet alle Ressourcen im Cluster und vergibt diese auf Anfrage an die *ApplicationMaster*, die diese benötigen, um die Jobs auszuführen und zu überwachen.

Der *Resource Manager* ist abermals in *Application Manager* und Scheduler unterteilt. Der Scheduler übernimmt das Allokieren von Ressourcen für die Ausführung von Jobs unter Einbeziehung von Kriterien wie Warteschlagen (*Queues*) und Kapazitäten einzelner Knoten. Der *Application Manager* hingegen nimmt neue Job-Anfragen von Clients entgegen und initialisiert diese. Dazu gehört unter anderem auch das Ausführen des jobspezifischen *ApplicationMasters* für die jeweilige Anwendung. Der *ApplicationMaster* fordert bei Ausführung eines Jobs eine bestimmte Anzahl an Containern an. Ein Container steht für eine Ressource (u. a. in Form von Speicher) auf einem Knoten. Diese Ressourcen werden einer Anwendung zum Ausführen zur Verfügung gestellt. Ein *Node Manager* kann auf Anfrage eines *ApplicationMasters* neue Container bereitstellen[4]. Daneben hat der *Node Manager* weiterhin die Aufgabe, als Agent den Knoten hinsichtlich CPU-Gebrauch, Speicher- und Netzwerkkapazitäten zu überwachen und regelmäßig Statusmeldungen an den *Resource Manager* zurückzuliefern.

Neben einer geringfügig angepassten Architektur bringt YARN ebenso ein neues Programmiermodell mit, das in Abschnitt 3.14 vorgestellt wird. Bis hierhin soll es erst einmal genügen, den Aufbau des Frameworks zu kennen, um eine einfache Hadoop-Instanz aufzusetzen.

[4] In Bild 3.44 sehen Sie diesen Prozess anhand eines Beispiels.

HINWEIS: In Kapitel 13, Data Processing Engines, in dem es u. a. um Apache Spark geht, wird im Detail darauf eingegangen, warum der Wechsel von Hadoop 1 auf Hadoop 2 auch dazu führte, dass sich Hadoop-unabhängige, verteilte Programmiermodelle verbreiteten.

3.4 Hadoop als Single-Node-Cluster aufsetzen

Im ersten Schritt wird Apache Hadoop als Single-Node-Cluster installiert. Auch wenn es keinen Sinn macht, ein skalierbares Big-Data-System auf nur einem Knoten produktiv zu einzusetzen, ist es eine sinnvolle Übung, um sich für eine verteilte Installation vorzubereiten.

Dieses Buch stellt die Installation von Apache Hadoop in seiner Open-Source-Ausprägung vor.

HINWEIS: Auch wenn es nicht Ziel des Buches ist, einzelne kommerzielle Hadoop-Distributionen im Detail vorzustellen, ist es dennoch sinnvoll sie zu erwähnen, da sie im Hadoop-Nutzerkreis einen hohen Bekanntheitsgrad haben.

Open-Source-Systeme zeichnen sich durch ihre Modularität aus und dadurch dass sie jederzeit in andere Systeme eingebunden werden können. Das führt dazu, dass einzelne Komponenten einfach ausgetauscht warden können. Anders ist das bei Closed-Source-Produkten, die aus mehreren Komponenten eines Herstellers bestehen.

Die naheliegende Folge ist, dass Organisationen – die sogenannten Distributoren – darum bemüht sind, für Anwender eine Konfiguration aus den unterschiedlichen Open-Source-Komponenten als Paket zu schnüren. Im Umfeld von Linux sind Distributionen wie Red Hat, SUSE oder Ubuntu entstanden. Der Hauptvorteil bei der Verwendung einer kommerziellen Distribution für Firmen ist, dass sie beim Hersteller Support einkaufen können.

Was für Linux Red Hat und Co. sind, sind für Hadoop die Distributionsanbieter Cloudera, Hortonworks und MapR. Ihre Distributionen und auch ihre Geschäftsstrategien unterscheiden sich teilweise sehr voneinander. Interessierte Leser infomieren sich am besten auf den Webseiten der Firmen über deren Angebot.

Es ist auch möglich, Hadoop-Distributionsimages in der Cloud (z. B. Microsoft Azure oder Amazon AWS) zu erstellen. Das ist vielleicht der schnellste Weg zum ersten Hadoop-Testsystem. Allerdings muss erwähnt werden, dass dies ebenfalls zu Kosten führen kann. Auch muss bedacht werden, dass sich die Distributionen teils massiv von der Open-Source-Variante unterscheiden, da in diesen Hadoop oft über eine eigene Administrationsoberfläche verwaltet wird.

Bei der Installation von Hadoop unterscheidet man hauptsächlich drei Konfigurationen.

Tabelle 3.1 Die drei möglichen Konfigurationen von Hadoop

Bezeichnung	Beschreibung	Verwendung
Standalone	Hadoop läuft als einzelner Java-Prozess auf einer einzigen Maschine.	Apache empfiehlt, diese Konfiguration lediglich für Entwicklungs- und Debugging-Zwecke zu verwenden. Hadoop ist nach dem Entpacken bereits fertig für den Stand-alone-Modus konfiguriert.
Pseudo distributed	Hadoop wird auf einer einzelnen Maschine eingerichtet, und jeder Hadoop-Daemon läuft in einem eigenen Java-Prozess.	Dieser Modus erfordert eine Konfiguration der einzelnen Komponenten im Unterverzeichnis Hadoops */etc/hadoop*. Durch das Aufsetzen eines Pseudo-Distributed-Clusters können Konfigurationen erprobt und das gesamte Setup der Hadoop-Instanz getestet werden.
Fully distributed	Hadoop läuft auf mehreren Maschinen im Cluster.	Ein Fully-Distributed-Setup wird in Produktionsumgebungen eingesetzt. Nur durch die Verwendung mehrerer Maschinen kann Hadoop seine Stärken in der verteilten Verarbeitung ausspielen.

Wir überspringen den *Stand-alone-Modus* und beginnen mit dem *Pseudo-Distributed-Mode*, da auch dieser recht leicht zu konfigurieren ist und wir daran lernen können, was für Konfigurationsmöglichkeiten in Hadoop vorliegen.

Installation eines Ubuntu-Servers

So, genug der Vorreden, bitte laden Sie nun den kostenlosen *VMware Player* herunter und installieren Sie diesen. Zu finden ist die Software unter folgendem Link *http://www.vmware.com/de/products/player/*. Wenn Sie es gewohnt sind, mit einer anderen virtuellen Umgebung zu arbeiten, können Sie das natürlich auch gerne tun.

HINWEIS: VMWare ist ein 1998 gegründetes, amerikanisches Unternehmen, das sogenannte Virtualisierungslösungen bereitstellt. Virtualisierung erlaubt es, einen Computer durch Software zu simulieren, sodass ein Betriebssystem nicht zwingend physikalische Hardware benötigt. Durch sogenannte **VMs (Virtuelle Maschinen)** kann ein beliebiges Betriebssystem als Gast in einem Hostsystem betrieben werden. Das erleichtert etwa Softwaretests auf vielen verschiedenen Betriebssystemversionen sowie das Testen von unbekannter, möglicherweise schädlicher Software (*Sandbox*). Ebenso können VMs leicht gesichert, vervielfältigt und weitergegeben werden.

Laden Sie nun bitte zusätzlich das Installations-Image von *Ubuntu Server* herunter (ich verwende Version 12.04.3 des 64-Bit-Servers). Dieses finden Sie hier: *http://www.ubuntu.com/download/server*. Wenn Sie sich im Umgang mit dem Terminal unsicher fühlen, dann können Sie auch zur Desktop-Variante von Ubuntu greifen.

Haben Sie das Image heruntergeladen, starten Sie bitte den *VMware Player*, klicken Sie auf **Create a New Virtual Machine**, selektieren Sie **Install disc image file** und wählen Sie dann das eben heruntergeladene ISO-Image.

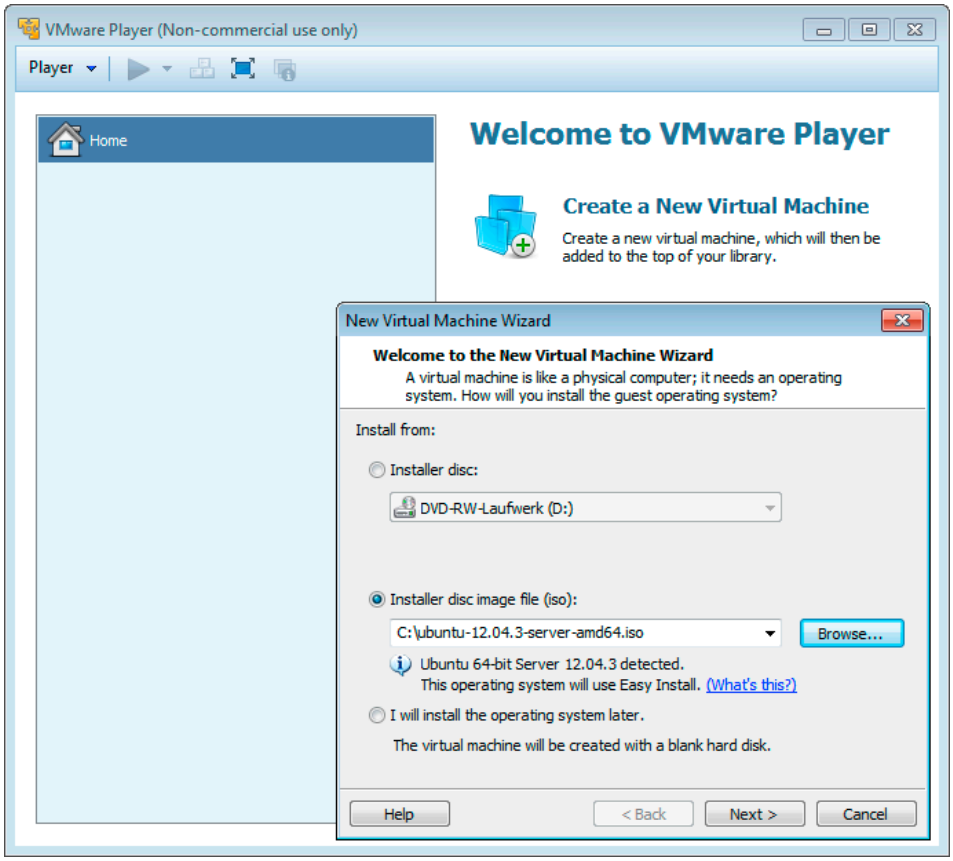

Bild 3.3 Installieren eines virtuellen Ubuntu-Servers

Klicken Sie auf **Next** und geben Sie einen Namen und ein Passwort ein. Ich werde im Folgenden für Name und Passwort *user1* verwenden. Ein weiterer Klick auf **Next** bringt Sie zu einem Dialog, in dem Sie bestimmen, wie die VM heißen soll. Nennen Sie diese *hadoop1*.

Klicken Sie ein weiteres Mal auf **Next** und dann auf **Customize Hardware**. Dort erhöhen Sie bitte den Arbeitsspeicher auf mindestens 2 Gigabyte (besser sind 8 GB) und die Anzahl der verwendeten Kerne auf 3 (besser sind 4). Nun schließen Sie den Dialog und klicken dann auf **Finish**. Der *VMware Player* beginnt jetzt, Ubuntu zu installieren, und Sie können sich kurz zurücklehnen.

TIPP: Sie werden eventuell gefragt werden, ob Sie **VMware Tools** für Ihre VM installieren möchten. Diese Anwendung erleichtert die Kommunikation zwischen Host- und Gastsystem, sodass Sie z. B. einfacher Dateien austauschen oder Einstellungen synchronisieren können. Wir benötigen das Werkzeug nicht explizit, allerdings ist es prinzipiell ratsam, es mit zu installieren.

(Optional) Anpassen des Tastaturschemas in Ubuntu

Das Tastaturlayout von Ubuntu ist zu Beginn auf Englisch gestellt. Mit folgendem Befehl können Sie es umstellen. Die Information, dass sich der *Bindestrich* der englischen Tastatur auf der Taste unseres *ß* befindet, spart Ihnen sicher einiges an Nerven.

Listing 3.1 Ändern des Tastaturlayouts unter Ubuntu

```
sudo dpkg-reconfigure keyboard-configuration
```

Geben Sie Ihr Benutzerpasswort ein, wenn Sie danach gefragt werden. Bestätigen Sie nun alle Einstellungen, bis Sie zu einer Auswahl mit dem Titel *Country of origin for the keyboard* kommen. Dort wählen Sie bitte *German* aus und klicken sich durch den restlichen Wizard.

Installation einer Java-Laufzeitumgebung

Da Hadoop auf Java basiert, benötigt es eine JRE (*Java Runtime Environment*), um ausgeführt werden zu können. Sie haben nun zwei Optionen. Entweder Sie benutzen das freie OpenJDK (*Java Development Kit*) oder die proprietäre Oracle JRE. Wir werden das OpenJDK verwenden. Öffnen Sie nun bitte die VM, loggen Sie sich mit Ihrem Benutzernamen und Passwort ein und führen Sie folgende Befehle aus, um die JRE zu installieren.

Listing 3.2 Installation der OpenJDK 7

```
sudo apt-get install openjdk-7-jdk
cd /usr/lib/jvm
sudo ln -s java-7-openjdk-amd64 jdk
```

In Listing 3.2 geschieht nichts anderes, als dass das OpenJDK installiert und auf das Verzeichnis *java-7-openjdk-amd64* ein symbolischer Link mit Namen *jdk* angelegt wird. Der Paketmanager fragt an einigen Stellen nach, ob er eine Operation wirklich durchführen soll. Bestätigen Sie das bitte je nach Vorschlag mit *Y*.

HINWEIS: Das **sudo** vor den jeweiligen Befehlen sorgt dafür, dass selbiger als Superuser ausgeführt wird. Das ist zum Beispiel bei der Installation von neuen Anwendungen oder bei Änderungen von Systemeinstellungen nötig. Sie werden folgend aufgefordert, Ihr Passwort einzugeben, um sich zu authentifizieren.

Der Paketmanager lädt nun die nötigen Dateien herunter und installiert diese anschließend. Mit dem Befehl in Listing 3.3 können Sie dann prüfen, ob Installation und Aktivierung der JRE funktioniert haben.

Listing 3.3 Überprüfung der primären Java-Laufzeitumgebung

```
java -version
```

Sieht das Ergebnis aus wie in Bild 3.4, dann war die Installation des JDK erfolgreich.

```
user1@localhost:~$ java -version
java version "1.7.0_25"
OpenJDK Runtime Environment (IcedTea 2.3.10) (7u25-2.3.10-1ubuntu0.12.04.2)
OpenJDK 64-Bit Server VM (build 23.7-b01, mixed mode)
user1@localhost:~$ _
```

Bild 3.4 Ausgabe des gesetzten JDK

Wird jedoch eine andere Ausgabe gezeigt, dann können Sie mit dem Befehl in Listing 3.4 eine alternative Java-Version auswählen.

Listing 3.4 Setzen einer alternativen JRE

```
sudo update-alternatives --config java
```

In der nun erscheinenden Liste können Sie mit den Zahlentasten die gewünschte Version setzen.

Einrichten eines dedizierten Hadoop-Users

Wir werden für ein passendes Rechtemanagement einen speziellen Benutzer samt Gruppe für die Hadoop-Installation anlegen. Führen Sie bitte dafür den in Listing 3.5 gezeigten Befehl aus.

Listing 3.5 Anlegen eines Hadoop-Benutzers mit Gruppe

```
sudo addgroup hadoop
sudo adduser --ingroup hadoop hduser
```

Die erzeugte Gruppe heißt *hadoop*, der Benutzer *hduser*. Als Passwort verwende ich ebenfalls *hduser*. Arbeiten Sie sich durch die nun erscheinenden Fragen zum Benutzer (Name, Raumnummer etc.). Diese können Sie leer lassen, der Inhalt spielt für uns keine Rolle. Bestätigen Sie die Korrektheit der Informationen mit *Y*, und der Benutzer wird vom System eingerichtet.

SSH-Zugang einrichten

Das Einrichten eines SSH-Zugangs (*Secure Shell*) auf der Hadoop-Instanz hat zwei Gründe. Später, wenn wir mehrere VMs zu einem Hadoop-Cluster zusammenschalten, damit diese ihre Arbeit untereinander aufteilen können, müssen sie miteinander kommunizieren können. Weiterhin wollen wir ebenso von unserem Hostsystem Dateien auf den VMs ablegen dürfen. Für beide Fälle wird ein SSH-Zugang vorausgesetzt.

Am Anfang gilt es, den *OpenSSH-Server* zu installieren.

Listing 3.6 Installation des OpenSSH-Servers

```
su - user1
sudo apt-get install openssh-server
```

HINWEIS: SSH besteht aus einem Netzwerkprotokoll und einigen Programmen, mit denen man eine sichere, verschlüsselte Netzwerkverbindung mit einem entfernten Computer herstellen kann.

Bitte führen Sie folgende Befehle aus, um das Erzeugen eines SSH-Keys für den Benutzer *hduser* zu initiieren.

Listing 3.7 Erzeugen eines SSH-Schlüssels für den Benutzer hduser

```
su – hduser
ssh-keygen –t rsa –P ""
```

Nachfragen nach dem Zielpfad des Schlüssels bestätigen Sie bitte einfach. Zuerst wechseln wir den aktiven Benutzer von *user1* hin zu *hduser*. Dann wird ein Schlüssel ohne ein Passwort erzeugt, was normalerweise nicht empfehlenswert, für unsere Zwecke aber in Ordnung ist. Den eben erzeugten Schlüssel müssen wir nun zu den autorisierten SSH-Schlüsseln unseres Benutzers hinzufügen.

Listing 3.8 Registrierung des eben generierten Schlüssels

```
cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
```

HINWEIS: Das **$HOME** verweist hier auf das Benutzerverzeichnis unseres *hdusers*, ähnlich den *Eigenen Dateien* unter Windows.

Um nun vice versa den Fingerprint des Hosts zu den bekannten Systemen (*known_hosts*) des *hdusers* hinzuzufügen, muss sich wenigstens ein einziges Mal im Vorfeld mit dem *localhost* verbunden werden.

Listing 3.9 Verbindung zum localhost herstellen

```
ssh localhost
```

Bestätigen Sie bitte die folgende Sicherheitsabfrage mit *Yes* und trennen Sie die Verbindung wieder mit dem Befehl *exit*.

(Optional) Deaktivieren des IPv6-Protokolls

Falls Sie eine ältere Hadoop-Version betreiben, sollten Sie das IPv6-Protokoll deaktivieren, da dieses in einigen Fällen zu Problemen führen kann. Dazu editieren wir die Datei */etc/sysctl.conf*. Da unser *hduser* keine Administratorenrechte besitzt, sollten Sie zunächst den Benutzer *user1* reaktivieren.

Listing 3.10 Editieren der Systemeinstellungen zu IPv6

```
su – user1
sudo nano /etc/sysctl.conf
```

Nun öffnet sich ein Editor, in dem Sie bitte folgende Zeilen am Ende der Textdatei einfügen.

Listing 3.11 Deaktivieren von IPv6

```
# disable ipv6
net.ipv6.conf.all.disable_ipv6 = 1
net.ipv6.conf.default.disable_ipv6 = 1
net.ipv6.conf.lo.disable_ipv6 = 1
```

Speichern Sie die Datei mit **Strg+o** ab und beenden Sie den Editor mit **Strg+x**. Um die Änderungen wirksam zu machen, müssen Sie Ubuntu neu starten.

Listing 3.12 Neustarten des Systems

```
sudo reboot
```

Melden Sie sich bitte wieder als *hduser* an und überprüfen Sie, ob IPv6 erfolgreich deaktiviert wurde.

Listing 3.13 Überprüfung der Deaktivierung von IPv6

```
cat /proc/sys/net/ipv6/conf/all/disable_ipv6
```

Gibt der Befehl die Zahl *1* zurück, so war das Vorgehen erfolgreich.

Installation von Hadoop

Nun sind wir endlich so weit, dass wir Hadoop installieren können.

Listing 3.14 Herunterladen und Entpacken von Hadoop

```
su – user1
cd /usr/local
sudo wget http://mirror.lwnetwork.org.uk/APACHE/hadoop/common/hadoop-2.2.0/hadoop-
2.2.0.tar.gz
sudo tar xzf hadoop-2.2.0.tar.gz
sudo mv hadoop-2.2.0 hadoop
sudo chown –R hduser:hadoop hadoop
```

Hier geschieht Folgendes: Wir wechseln zuerst in das Verzeichnis */usr/local* und laden dorthin die Datei *hadoop-2.2.0.tar.gz* herunter. Dieses Archiv entpacken wir und benennen den beim Entpacken erzeugten Ordner *hadoop-2.2.0* um in *hadoop*. Das erspart uns später jede Menge Tipparbeit. Zuletzt bestimmen wir, dass unser Benutzer *hduser* Besitzer des Ordners *hadoop* (und all seiner Unterordner und Dateien) ist.

Nun müssen noch einige Benutzervariablen erstellt werden, die beim Anmelden unseres *hdusers* initialisiert werden und danach im Terminal zur weiteren Verwendung bereitstehen. Dazu wechseln wir wieder zum *hduser* und bearbeiten die *.bashrc* in unserem Home-Verzeichnis.

Listing 3.15 Wechseln ins Home-Verzeichnis

```
su - hduser
nano .bashrc
```

Kopieren Sie nun die folgenden Zeilen ans Ende der *.bashrc*. Mit den Tasten *Bild-Auf* und *Bild-Ab* können Sie im Nano-Editor schneller scrollen.

Listing 3.16 Setzen der Umgebungsvariablen für Hadoop

```
# Java
export JAVA_HOME=/usr/lib/jvm/jdk/

# Hadoop
export HADOOP_INSTALL=/usr/local/hadoop
export PATH=$PATH:$HADOOP_INSTALL/bin
export PATH=$PATH:$HADOOP_INSTALL/sbin
export HADOOP_MAPRED_HOME=$HADOOP_INSTALL
export HADOOP_COMMON_HOME=$HADOOP_INSTALL
export HADOOP_HDFS_HOME=$HADOOP_INSTALL
export HADOOP_YARN_HOME=$HADOOP_INSTALL
```

Navigieren Sie anschließend in das Verzeichnis */usr/local/hadoop/etc/hadoop* und bearbeiten Sie die Datei *hadoop-env.sh*. Suchen Sie dort die Zeile, die mit *export JAVA_HOME* beginnt, und ändern Sie diese wie folgt.

Listing 3.17 Setzen der Umgebungsvariablen JAVA_HOME in hadoop-env.sh

```
export JAVA_HOME=/usr/lib/jvm/jdk/
```

TIPP: Der letzte Schritt ist nur deswegen notwendig, weil Hadoop es in der Version 2.2.0 versäumt, die Definition von *JAVA_HOME* aus der *.bashrc* auszulesen. Wenn das in einem der nächsten Fixes obsolet wird, können Sie diesen Schritt getrost auslassen.

Starten Sie nun Ihr System neu, und Hadoop ist fertig eingerichtet. Den Erfolg Ihrer Bemühungen prüfen Sie wie in Listing 3.18 gezeigt.

Listing 3.18 Überprüfen, ob Hadoop erfolgreich gestartet wurde

```
hadoop version
```

Der entsprechende Output sollte Bild 3.5 ähneln.

```
hduser@localhost:~$ hadoop version
Hadoop 2.2.0
Subversion https://svn.apache.org/repos/asf/hadoop/common -r 1529768
Compiled by hortonmu on 2013-10-07T06:28Z
Compiled with protoc 2.5.0
From source with checksum 79e53ce7994d1628b240f09af91e1af4
This command was run using /usr/local/hadoop/share/hadoop/common/hadoop-common-2
.2.0.jar
hduser@localhost:~$ _
```

Bild 3.5 Ergebnis der Überprüfung zum Start von Hadoop

Anschließend müssen lediglich noch ein paar kleinere Dinge erledigt werden, bis wir Hadoop testen können: unter anderem die Konfiguration der Ports, um die wir uns nun kümmern wollen. Davor müssen wir jedoch noch drei weitere Ordner anlegen.

Listing 3.19 Anlegen eines Temp-, Name- und Dataverzeichnisses

```
su - user1
sudo mkdir -p /usr/local/hadoop/tmp
sudo mkdir -p /usr/local/hadoop/name
sudo mkdir -p /usr/local/hadoop/data
sudo chown hduser:hadoop /usr/local/hadoop/tmp
sudo chown hduser:hadoop /usr/local/hadoop/name
sudo chown hduser:hadoop /usr/local/hadoop/data
```

Das Verzeichnis *tmp* dient Hadoop, wie der Name vermuten lässt, um temporäre Dateien abzulegen. Im *data* genannten Ordner speichert ein Datenknoten (*Data Node*) später die Datenblöcke. Das Name-Verzeichnis wird vom *Name Node* dafür verwendet, um die Namenstabellen (*Name Table*) zu speichern. Keine Angst vor den vielen neuen Begriffen, wir gehen später noch detailliert darauf ein.

HINWEIS: Die Dateien, die wir im Folgenden anlegen und editieren, finden Sie fertig in den Beispieldateien im Ordner *single_node_config* (Kapitel 3).

Navigieren Sie nun in das Verzeichnis */usr/local/hadoop/etc/hadoop/* und öffnen Sie als *hduser* die Datei *core-site.xml* mithilfe von *nano*. Fügen Sie dort zwischen den Tags <*configuration*> und </*configuration*> folgende Zeilen ein.

Listing 3.20 Einstellungen in core-site.xml

```
<property>
   <name>fs.defaultFS</name>
   <value>hdfs://localhost:9000</value>
</property>
<property>
   <name>hadoop.tmp.dir</name>
   <value>/usr/local/hadoop/tmp</value>
</property>
```

Die Eigenschaft *fs.defaultFS* bestimmt, wo der *Name Node* zu finden ist. Dieser übernimmt die Koordination der *Data Nodes*, auf denen später im Cluster die Daten liegen (siehe Abschnitt 3.9); er agiert also als Master. *fs.defaultFS* war früher als *fs.default.name* bekannt. Diese Bezeichnung ist allerdings nun veraltet. Sie finden eine Liste aller Eigenschaften, die in den letzten Releases ihren Namen geändert haben, unter der folgenden Adresse:

http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/DeprecatedProperties.html

Editieren Sie nun im gleichen Verzeichnis die Datei *hdfs-site.xml* und fügen Sie auch hier die genannten Eigenschaften hinzu.

Listing 3.21 Einstellungen in hdfs-site.xml

```
<property>
   <name>dfs.replication</name>
   <value>1</value>
</property>
<property>
```

```
    <name>dfs.permissions</name>
    <value>false</value>
</property>
<property>
    <name>dfs.namenode.name.dir</name>
    <value>file:/usr/local/hadoop/name</value>
</property>
<property>
    <name>dfs.datanode.data.dir</name>
    <value>file:/usr/local/hadoop/data</value>
</property>
```

Durch die Eigenschaft *dfs.replication* legen wir fest, in wie vielen Kopien unsere zu verarbeitenden Daten später auf dem Cluster verteilt werden sollen. Da wir in diesem Falle nur einen Knoten verwenden, können wir lediglich eine Kopie speichern.

Die Eigenschaft *dfs.permissions* bestimmt, ob das Rechtesystem von Hadoop genutzt werden soll. Gemäß dem Posix-Modell verfügt jede Datei über einen Besitzer und eine Gruppe. Alle Zugriffe von Fremdbenutzern, Besitzern und Nutzern in der entsprechenden Gruppe werden anders behandelt. Für unsere Zwecke schalten wir diese Funktion ab, das erleichtert uns die Entwicklung und das Testen unserer Jobs, die wir später schreiben werden.

Der Zweck von *dfs.datanode.data.dir* und *dfs.datanode.name.dir* wurde bereits oben erklärt. Die Verzeichnisse, die Sie hier angeben, müssen zwingend existieren, Hadoop erstellt diese nicht. Zudem können Sie hier auch mehrere Ordner durch Kommata getrennt angeben. Die Daten werden dann in allen gelisteten Verzeichnissen abgelegt.

Weiter geht es mit der Datei *mapred-site.xml*. Diese existiert noch nicht im entsprechenden Verzeichnis, jedoch stellt uns Hadoop ein Template für eine solche Datei zur Verfügung. Dieses benennen wir der Einfachheit halber in *mapred-site.xml* um. Natürlich können Sie es auch nach eigenem Gusto kopieren, um eine Sicherungskopie davon zu behalten. Da die Datei jedoch beinahe leer ist, müssen wir uns darum eigentlich keine Sorgen machen.

Listing 3.22 Umbenennen der Datei mapred-site.xml

```
mv mapred-site.xml.template mapred-site.xml
```

In *mapred-site.xml* bestimmen wir lediglich, dass wir das neue Map-Reduce-Framework (siehe Abschnitt 3.5) *YARN* benutzen möchte.

TIPP: Müssen Sie dann Ihre bisherigen Map-Reduce-Jobs umgestalten, sodass sie mit dem neuen Framework *YARN* funktionieren? Nein, die Entwickler von Hadoop versichern eine vollständige Kompatibilität zwischen den Map-Reduce-Jobs der ersten Generation und denen der zweiten. Diese wird übrigens auch MR2 genannt. Möchten Sie dennoch den alten Map-Reduce-Algorithmus verwenden, ersetzen Sie in Listing 3.23 das *yarn* durch *local*.

Listing 3.23 Einstellungen in mapred-site.xml

```
<property>
   <name>mapreduce.framework.name</name>
   <value>yarn</value>
</property>
```

TIPP: In manchen Quellen finden Sie in einem *Property-Tag* das Tag **<final >true</final>**. Wenn eine Eigenschaft auf diese Weise markiert wurde, kann sie später nicht mehr programmatisch, von uns oder von Hadoop selber, verändert werden.

Die letzten Änderungen nehmen wir in der Datei *yarn-site.xml* vor.

Listing 3.24 Einstellungen in yarn-site.xml

```
<property>
   <name>yarn.Node Manager.aux-services</name>
   <value>mapreduce_shuffle</value>
</property>
<property>
   <name>yarn.Node Manager.aux-services.mapreduce.shuffle.class</name>
   <value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
<property>
   <name>yarn.Node Manager.vmem-pmem-ratio</name>
   <value>3</value>
</property>
<property>
   <name>yarn.Node Manager.delete.debug-delay-sec</name>
   <value>600</value>
</property>
```

Die ersten beiden Eigenschaften (*yarn.Node Manager.aux-services* und *yarn.Node Manager.aux-services.mapreduce.shuffle.class*) legen fest, wie Hadoop später die Verteilung (Mischung) der Knoten im Cluster handhaben wird. In früheren Versionen waren diese Methoden fest implementiert. Ab Version 2.x wollten die Entwickler jedoch die Möglichkeit bieten, diese Verteilung selber festzulegen. Wir nutzen hier jedoch den altbekannten Standard. *yarn.Node Manager.vmem-pmem-ratio* stellt das Verhältnis von physikalischem zu virtuellem Speicher dar. Verwenden wir beispielsweise 4 GB RAM, steht uns 4 * 3 = 12 GB virtueller Speicher für unsere Map-Reduce-Jobs oder YARN-Anwendungen zur Verfügung. Der Default-Wert ist hier 2,1. *yarn.nodemanager.delete.debug-delay-sec* gibt an, wie viele Sekunden die Anwendungsdaten (lokale Ressourcen, Log-Dateien ...) für auf YARN laufende Anwendungen bestehen bleiben, bevor sie automatisch gelöscht werden. Für eine Produktivumgebung ist es ratsam, diesen Wert niedrig zu halten. Ich stelle ihn auf 600 Sekunden (10 Minuten), sodass wir später, wenn wir unsere erste YARN-Anwendung implementieren, einen Blick auf die Ressourcen werfen können, die der Anwendung zur Verfügung gestellt werden.

HINWEIS: In manchen Tutorials lesen Sie von **weiteren Eigenschaften**, die in der *yarn-site.xml* gesetzt werden, z. B. *yarn.resourcemanager.resource-tracker.address* oder *yarn.resourcemanager.address*. Diese können Sie bei Bedarf ebenfalls ändern, ansonsten werden die Dienste auf dem jeweiligen Default-Port gestartet. Da wir in diesem Abschnitt nur die Minimalkonfiguration verwenden wollen, lasse ich sie hier erst einmal weg. Wir werden später beim Aufbau eines Clusters darauf zurückkommen.

Formatieren des HDFS (Hadoop Distributed File System)

Das Dateisystem, das Hadoop benutzt (siehe 3.2), muss wie jedes andere Dateisystem auch vor dem ersten Benutzen formatiert werden. Das erreichen wir mit dem folgenden Befehl.

Listing 3.25 Formatieren des HDFS

```
hdfs namenode -format
```

Bestätigen Sie die Nachfrage, ob Sie das HDFS wirklich löschen wollen, mit einem großen *Y*, nicht bloß mit einem kleinen.

Nun wird sich zeigen, ob Sie die verschiedenen XML-Dateien fehlerfrei geschrieben haben. Sollte Hadoop diese nicht lesen können, da eventuell ein Schrägstrich vergessen wurde, so werden Sie in der Ausgabe des Formatierungsprozesses Fehlermeldungen finden. Meistens werden diese mit Hinweisen auf die entsprechende Fehlerquelle versehen.

TIPP: Achten Sie darauf, niemals ein laufendes HDFS zu formatieren, sonst werden Ihre Daten darauf, wie zu erwarten, gelöscht.

(Optional) Konfigurieren der Datenkompression im HDFS

Das HDFS bietet die Möglichkeit, die sich darin befindenden Daten zu komprimieren. Das bringt einerseits den Vorteil, dass I/O-intensive Operationen (Datentransfer, Speichern von Daten) schneller ausgeführt werden, da kleinere Daten schneller über den Cluster verteilt und schneller in einen Map-Reduce-Job eingelesen werden können. Andererseits wird natürlich beim Komprimieren sowie beim Dekomprimieren CPU-Zeit in Anspruch genommen, und die Verarbeitungszeit steigt an. Es gilt folglich abzuwägen, ob man mit eingeschalteter Kompression Speicherbedarf, Festplatten-I/O und Netzwerkbelastung reduzieren möchte oder ob man Wert darauf legt, die CPU zu entlasten, und dafür auf Kompression verzichtet. Tabelle 3.2 zeigt die verfügbaren Kompressionsverfahren in Hadoop.

Tabelle 3.2 Kompressionsverfahren in Hadoop

Format	Algorithmus	Dateiendung	Teilbar (Splittable)	Bemerkung
zlib	DEFLATE	.deflate	Nein	Standardverfahren
gzip	zlib Wrapper	.gz	Nein	Kann in HBase verwendet werden
bzip2	Burrows-Wheeler transformation	.bz2	Ja	Üblicherweise in PIG benutzt
lzo	Ähnlich LZ77	.lzo	Nein	Kann in HBase verwendet werden
lz4	Ähnlich LZ77	.lz4	Nein	In neueren Hadoop-Distributionen verfügbar
snappy	LZ77	.snappy	Nein	Kann in HBase verwendet werden, wurde von Google herausgebracht

Kompression ist in Hadoop per Default ausgeschaltet. Um sie zu aktivieren, müssen Sie die Option in der *mapred-site.xml* aktivieren. Dazu setzen Sie die folgenden Werte:

- **mapreduce.map.output.compress** auf *true*
- **mapreduce.map.output.compress.codec** auf die gewünschte Klasse (s.u.)
- **mapreduce.output.fileoutputformat.compress** auf *true*
- **mapreduce.output.fileoutputformat.compress.codec** auf die gewünschte Klasse
- **mapreduce.output.fileoutputformat.compress.type** auf den Kompressionstyp der Sequence-Files (NONE, RECORD (Default), BLOCK)

Die oben erwähnten Klassen, die den oben gezeigten Kompressionsmethoden entsprechen, sehen Sie hier:

- *org.apache.hadoop.io.compress.DefaultCodec* (zlib)
- *org.apache.hadoop.io.compress.GzipCodec* (gzip)
- *org.apache.hadoop.io.compress.BZip2Codec* (bzip2)
- *com.hadoop.compression.lzo.LzoCodec* (lzo)
- *org.apache.hadoop.io.compress.Lz4Codec* (lz4)
- *org.apache.hadoop.io.compress.SnappyCodec* (Snappy)

Ob und wann Sie ein Kompressionsverfahren einschalten, ist oft vom individuellen Fall abhängig. In einem Big-Data-Projekt bin ich mit einigen Kollegen mal in eine recht ungewöhnliche Lage geraten: Uns ist der Speicherplatz auf den Maschinen unseres Clusters ausgegangen, da wir die Daten auf dem HDFS für eine Echtzeitsuche indizieren mussten und die Indizes den Speicherbedarf über Nacht etwa um den Faktor 10 ansteigen ließen. In dem Fall war es klar, dass wir die Dateien komprimieren mussten, um weiterarbeiten zu können. Angemerkt sei, dass die Maschinen, die uns zur Verfügung standen, über ausreichend CPU-Leistung verfügten, um den Mehraufwand bei der Verarbeitung bewältigen zu können. Halten Sie also die Augen während der Arbeit mit Hadoop offen und bedenken Sie, dass Sie durch Datenkompression die Belastung einzelner Hardwarebauteile Ihres Clusters umwälzen können.