THM TECHNISCHE HOCHSCHULE MITTELHESSEN | CAMPUS GIESSEN | MNI Mathematik, Naturwissenschaften und Informatik

Apache Hadoop

Hauptseminar "Cloud-Plattformen und Big Data"
Dozent Steffen Rupp

von

René Gentzen

rene.gentzen@mni.thm.de

im WS22/23

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Hadoop Grundlagen

"Die Apache Hadoop Softwarebibliothek ist ein Framework, das die über Computercluster verteilte Verarbeitung großer Datensätze mit einfachen Programmiermodellen ermöglicht. Es ist so konzipiert, dass es von einzelnen Servern bis hin zu Tausenden von Rechnern skaliert werden kann, von denen jeder lokale Rechenleistung und Speicherplatz bietet. Anstatt sich auf Hardware zu verlassen, um eine hohe Verfügbarkeit zu gewährleisten, ist die Bibliothek selbst so konzipiert, dass sie Ausfälle auf der Anwendungsebene erkennt und bewältigt, so dass ein hochverfügbarer Dienst auf einem Cluster von Computern bereitgestellt wird, von denen jeder für sich für Ausfälle anfällig sein kann."[1]

So beschreibt (übersetzt aus dem Englischen) die Apache Software Foundation ihr Top Level Projekt **Apache Hadoop**. Diese Arbeit wird einen pragmatischen Überblick über Hadoop und die Komponenten im Hadoop Ecosystem geben. Dabei soll der Fokus nicht auf technischen Details, sondern auf dem praktischen Einsatz von Hadoop liegen. Es soll anhand von Anwendungsfällen demonstriert werden, wie die einzelnen Hadoop Komponenten zur Lösung bestimmter Problemstellungen auf- und eingesetzt werden.

1.1 Technischer und geschichtlicher Hintergrund

1.1.1 Anforderungen von Big Data

"Der Begriff „Big Data" bezieht sich auf Datenbestände, die so groß, schnelllebig oder komplex sind, dass sie sich mit herkömmlichen Methoden nicht oder nur schwer verarbeiten lassen."[2]

Schon Anfang der Neunziger war es nicht mehr praktikabel, Webseiten händisch, zum Beispiel in "Web Directories", zu katalogisieren. Man wollte Nutzern trotzdem die Möglichkeit geben, Informationen durch das Durchsuchen zentraler Anlaufstellen ausfindig zu machen. Automatisierte Tools, die sogenannten "Web Crawler" wurden erfunden, um diese Arbeit zu übernehmen.[3]

Das Internet erlebte in den letzten Jahren des 20. Jahrhunderts ein explosionsartiges Wachstum an Nutzern und Webseiten, und damit auch an Informationen, die katalogisiert werden mussten.[4] Um eine immer größer werdende Menge an Informationen verarbeiten zu können, gibt es zwei Ansätze der Skalierung: Vertikale und horizontale Skalierung. Diese sollen in den folgenden Abschnitten erläutert werden, um die Designphilosophie hinter Hadoop zu verstehen.

1.1.2 Vertikale Skalierung

Bei der vertikalen Skalierung ("scaling up") werden *einem* System mehr Ressourcen wie zum Beispiel größerer Speicher, oder eine schnellere CPU hinzugefügt. Dadurch bekommt man einen Performance-Gewinn: Man kann mehr Daten speichern, oder Berechnungen werden schneller fertig gestellt. Ein großer Vorteil der vertikalen Skalierung ist, dass Anwendungsprogramme in der Regel nicht angepasst werden müssen, um vom diesem Performance-Wachstum zu profitieren. Wenn man eine 5TB große Festplatte gegen eine 10TB Festplatte austauscht, dann hat man den Speicherplatz eines Servers vertikal skaliert. Die darauf laufenden Programme müssen nicht angepasst werden, sondern man kann einfach doppelt so viele Daten speichern.[5]

Vertikale Skalierung hat drei große Nachteile: Erstens kann man nicht unbegrenzt vertikal skalieren. Ein Server kann physisch nur eine begrenzte Anzahl an Hardware aufnehmen. Zweitens wächst die Performance eines Systems bei vertikaler Skalierung höchstens linear[6], die Kosten allerdings nicht[7]. Heutzutage kann man gerade bei Cloud-Anbietern sehr leistungsfähige Systeme bei linearem Preisanstieg mieten.[8] Sucht man aber noch mehr Performance in *einem* System, dann steigen die Kosten exponentiell[9]. Drittens skalieren nicht alle Faktoren in einem System gleich gut vertikal. Die Speicherkapazität von SSDs ist zum Beispiel seit 1978 von 45MB auf 100TB gestiegen (Faktor $2222,22 \cdot 10^3$), während sich die Datenrate nur von 1.5MB/s auf 500/460MB/s (Sequential Read/Write) erhöht hat (Faktor $0,333 \cdot 10^3$).[9][10]

1.1.3 Horizontale Skalierung

Ein Cluster ist ein Verbund aus Computern (Nodes), die wie ein einziger, deutlich leistungsfähigerer Computer arbeiten. Aufgaben und Daten werden in kleinere Teile zerlegt und auf alle Nodes im Cluster aufgeteilt, welche dann parallel Teilaufgaben lösen. Ergebnisse werden zusammengefügt und zurückgegeben. Anders als bei der vertikalen Skalierung kann man gerade in Zeiten des Cloud Computings praktisch unendlich horizontal skalieren. Allerdings muss man dafür kompliziertere Anwendungslogik verwenden, die mit der parallelen Ressourcenverteilung eines Clusters funktioniert.[11]

Bei der horizontalen Skalierung ("scaling out") werden einem Cluster zur Leistungssteigerung zusätzliche Nodes hinzugefügt. So kann ein Rechner zum Beispiel 500MB/s von seiner Festplatte lesen und verarbeiten, zehn Rechner lesen und verarbeiten in dieser Zeit allerdings 5000MB/s und können ihre Teilergebnisse anschließend zu einer Antwort zusammenfügen. Hierbei entsteht zwar zusätzlicher Netzwerk- und Verwaltungsaufwand ("Overhead"), aber die Leistungsfähigkeit des Clusters wächst mit jedem hinzugefügten Node. Ein horizontal skalierbares System ist mit höheren anfänglichen Kosten verbunden, kann dann aber bei linearem Kostenaufwand praktisch unendlich skaliert werden.[7]
Wie im eingänglichen Zitat erwähnt, setzt Hadoop auf eben dieses Prinzip der Skalierbarkeit, um "die über Computercluster verteilte Verarbeitung großer Datensätze mit einfachen Programmiermodellen"[1] als Dienst mit hoher Verfügbarkeit anzubieten. Die Technologien, die konkret dahinter stecken, werden im nächsten Abschnitt behandelt.

1.1.4 Historie

2002 begannen Doug Cutting und Mike Cafarella ihre Arbeiten an Apache Nutch[1], einer Open Source Web Search Engine als Teil des Apache Lucene Projekts[2]. Die beiden mussten Wege finden, um ihr Projekt auf die Milliarden Webseiten des Internets zu skalieren. 2003 veröffentlichte Google ein Whitepaper zur Architektur des Google File System (GFS), Googles eigenem verteilten Dateisystem.[3] Als Google 2004 dann ein weiteres Whitepaper zum MapReduce Programmiermodell veröffentlichte[4], sahen Cutting und Cafarella darin die Lösung für Nutch's Skalierungsproblem. Sie implementierten eigene Versionen von MapReduce als Processing Engine und des GFS zur Datenhaltung (NDFS, Nutch Distributed File System) als Basis für Nutch. Da diese beiden Komponenten mannigfaltige Anwendungsfälle außerhalb der Web-Suche bedienen konnten, wurden sie 2006 als eigenes Projekt Apache Lucene unterstellt und erhielten den Namen **Hadoop**. Ungefähr zur gleichen Zeit wurde Doug Cutting von Yahoo! rekrutiert, um Hadoop dort mit zusätzlichen Ressourcen weiterzuentwickeln. 2008 wurde Hadoop schließlich zu einem Top Level Projekt der Apache Software Foundation.[5] [14]

1.2 Hadoop Core

Der Kern von Hadoop (Hadoop Core) besteht seit Hadoop 2.x aus vier Modulen, welche im offiziellen Download zusammengefasst sind[1]:

- Hadoop Distributed File System (HDFS™): Hadoops verteiltes Dateisystem
- Hadoop MapReduce: Hadoops Parallel Processing Engine für große Datenmengen
- Hadoop YARN: Ein Framework für Job Scheduling und Ressourcenverwaltung im Cluster
- Hadoop Common: Unterstützende Programme für die anderen Hadoop-Module

Diese Komponenten bringen alles mit, was man zur verteilten Verarbeitung und Speicherung großer Datenmengen benötigt. Dazu schreibt man in der Regel Java-Applikationen, die bestimmte Klassen aus den Bibliotheken Hadoop ableiten. Ein praktisches Beispiel dazu wird im Abschnitt 1.2.4 vorgestellt.

[1] https://nutch.apache.org/
[2] https://lucene.apache.org/
[3] 12, The Google File System.
[4] 13, MapReduce: Simplified Data Processing on Large Clusters.
[5] https://hadoop.apache.org/

1.2.1 HDFS

Das HDFS ist ein Dateisystem, welches dem Anwender eine Abstraktionsschicht über verteilt gespeicherte Daten bietet. Dateien lassen sich ganz normal über einen Dateipfad im HDFS ansprechen, auch wenn sie im Hintergrund in Einzelteilen über viele Nodes verteilt gespeichert sind. Das HDFS ist für den Betrieb auf Clustern aus sogenannter *Commodity Hardware* konzipiert. Commodity Hardware ist günstige, leicht zu ersetzende Hardware. Bei Commodity-Hardware-Clustern wird nicht etwa versucht, Ausfälle einzelner Nodes durch den Einsatz von besonders ausfallsicherer (und somit teurer) Hardware zu verhindern. Fällt ein Node aus, was in einem Cluster von hunderten Maschinen kein Sonderfall ist, übernimmt ein anderer Node dessen Arbeit, ohne dass dadurch die Verfügbarkeit des Clusters beeinträchtigt wird. Das HDFS setzt dafür auf die Konzepte von Blöcken, Replikation und Redundanz.[15]

Ein vollwertiger Hadoop Cluster (Hadoop im *fully-distributed Mode*) besteht aus mindestens einem Master, dem **NameNode**, und einem oder mehr Workern, den **DataNodes** (*vgl. Abb.* 1.1). Um Dateien im HDFS zu speichern (Beispiel siehe 1.2.4), teilt ein *Client*-Prozess die Dateien in Blöcke von standardmäßig 128MB auf und kontaktiert den NameNode. Der NameNode hat einen Überblick über den verfügbaren Speicherplatz aller DataNodes und designiert manche davon, um einige der Blöcke aufzunehmen. Der NameNode achtet außerdem darauf, dass jeder einzelne Block repliziert und auf unterschiedlichen DataNodes gespeichert wird. Standardmäßig verteilt Hadoop drei Kopien eines jeden Blocks im Cluster. Dadurch verbraucht man zwar drei mal so viel Speicher wie bei herkömmlichen, nicht redundanten Dateisystemen, erreicht dafür aber eine sehr hohe Verfügbarkeit. Der Einsatz von Commodity Hardware hält trotz des erhöhten Speicherbedarfs die Kosten niedrig.[15]

Die DataNodes senden in regelmäßigen Abständen sogenannte *Block Reports* an den NameNode. Dieser gleicht die Block Reports mit dem Soll-Zustand des Dateisystems ab. Ist zum Beispiel in einem Node eine Festplatte ausgefallen, so sind manche Blöcke unterrepliziert. Der NameNode veranlasst DataNodes, die Kopien der betroffenen Blöcke besitzen dazu, diese an andere DataNodes zu senden, bis der Soll-Zustand des Clusters wieder hergestellt ist.

1.2.2 MapReduce

MapReduce heißt sowohl ein Programmiermodell zur parallelisierten Verarbeitung von Datensätzen, als auch die konkrete Implementierung eben dieses Modells zur Verwendung mit dem HDFS. MapReduce macht sich mehrere Prinzipien zu Nutze, um effizient mit großen Datenmengen umzugehen[16]:
Aufteilung: Eingabedaten werden in **InputSplits** geteilt verarbeitet. Dadurch verarbeitet ein einzelner Prozess ein logisch zusammenhängendes Datenpaket.
Parallelisierung: InputSplits werden parallel auf mehreren Nodes bearbeitet und die Ausgaben zusammengeführt. Dadurch werden auch bei großen Datenmengen hohe Datendurchsatzraten erreicht.
Datenlokalität: Der erste Teil der Verarbeitungslogik, die Mapping-Phase, wird möglichst

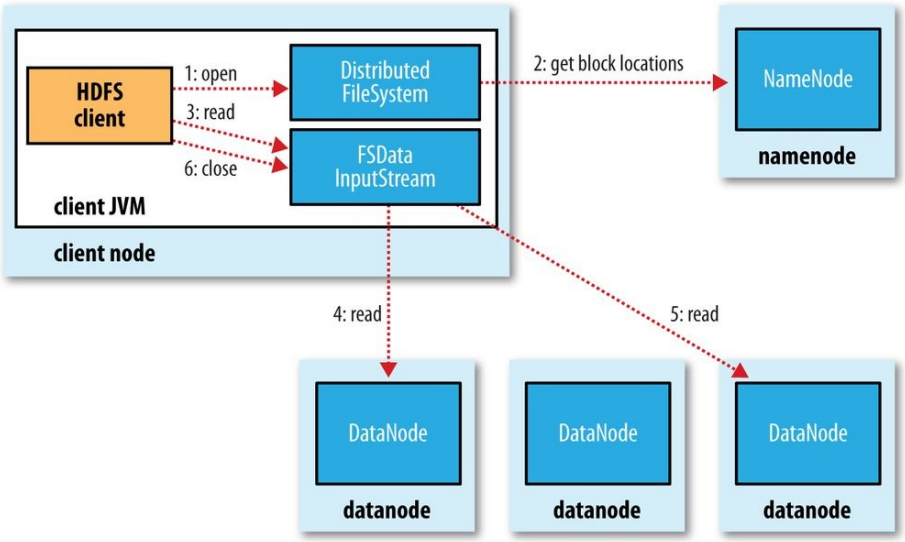

Abbildung 1.1: Architektur des HDFS(15, S.69, Figure 3-2)

nahe an den Daten durchgeführt; wenn möglich auf den Nodes, auf denen die Daten gespeichert sind. Ansonsten wird versucht, die Verarbeitung wenigstens auf dem gleichen Server Rack durchzuführen, um die Belastung der Netzwerkinfrastruktur so gering wie möglich zu halten.

Ein MapReduce-Job besteht aus zwei Phasen: der **Map-Phase** und der **Reduce-Phase**. Logisch kann man dazwischen noch die **Sort- und Shuffle-Phase** unterscheiden (siehe Abb. 1.2).

Wie eingangs erwähnt, wird die Eingabe in InputSplits zerteilt. Diese werden wiederum in einzelne Datensätze, die **Records**, aufgespalten. Wie diese Aufteilung abläuft, wird durch das **InputFormat** bestimmt, welches vom Anwender im Programmcode festgelegt und auf das Format der Eingabedaten abgestimmt werden muss. Dabei stehen zum Beispiel *TextInputFormat* oder *KeyValueTextInputFormat* zur Verfügung. Es ist auch möglich, durch Ableiten der abstrakten Java-Klasse *InputFormat* eigene InputFormats zu schreiben.[15]

Für jeden InputSplit wird ein eigener Map-Prozess (**Mapper**) gestartet. Dieser erhält alle Records des InputSplits in Form von **Key-Value-Paaren** als Eingabe. Auf jeden Record wird eine vom Anwender geschriebene Map-Funktion angewendet, die oftmals die Daten filtert und vorbereitet, zum Beispiel durch Parsen von Strings in Integer. Die Daten werden vom Eingabeformat in bereinigte Key-Value-Paare **gemappt**. Das Ergebnis wird an Reduce-Prozesse (**Reducer**) weitergegeben. Ein Beispiel dazu wird in Abschnitt 1.2.4 besprochen.

Bevor die Key-Value-Paare an die Reducer gegeben werden, werden sie nach Keys sortiert und gruppiert. Dies geschieht in der Sort- und Shuffle-Phase. Der Input für den Reducer ist dann eine Liste mit Key-Value-Paaren, wobei die Values wiederum Listen mit den Werten sind, die von den Mappern für jeden Key gefunden wurden (vgl. Abb. 1.3).

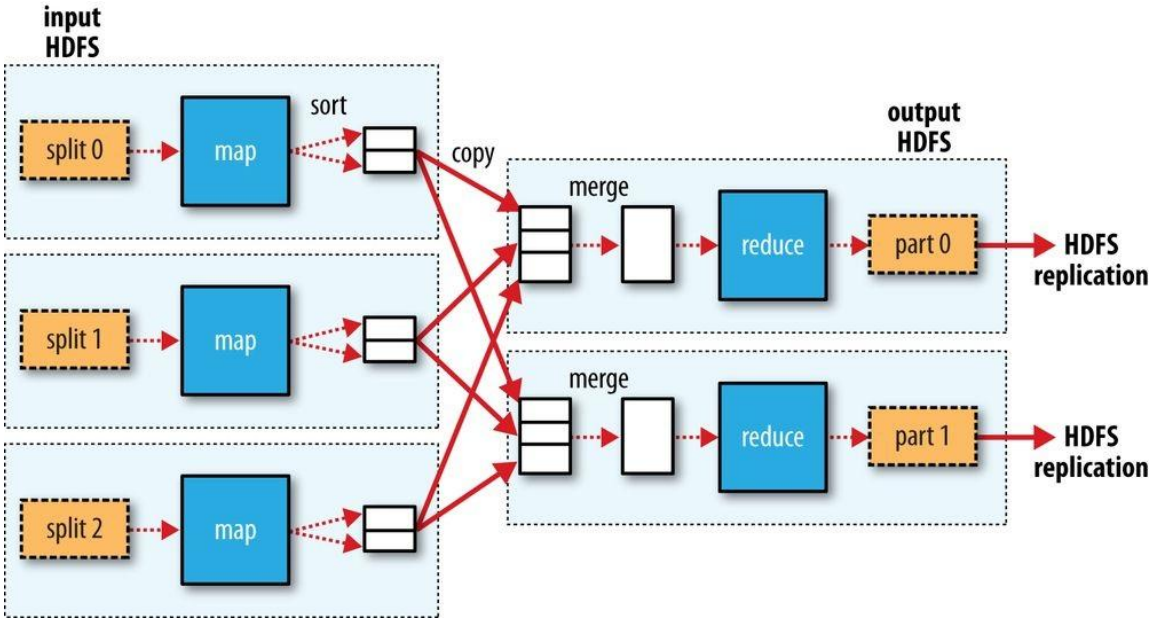

Abbildung 1.2: Die Phasen von MapReduce(15, Seite 34, Figure 2-4)

Der Reducer wendet eine, wiederum vom Anwender im Programmcode hinterlegte, Reduce-Funktion auf die ihm übergebenen Daten an. Für jeden Key wird die Liste aus Values zu einem einzigen Value **reduziert**, zum Beispiel durch Bestimmung des Maximums oder Aufsummierung aller Teilwerte. Die Anzahl der Reduce-Prozesse bestimmt die Anzahl der Ausgabedateien und *kann* im Programmcode festgelegt werden. Allerdings sollte man gute Gründe haben, um die von Hadoop gewählten Werte zu überschreiben, da dies katastrophale Folgen für die Performance haben kann.[17]

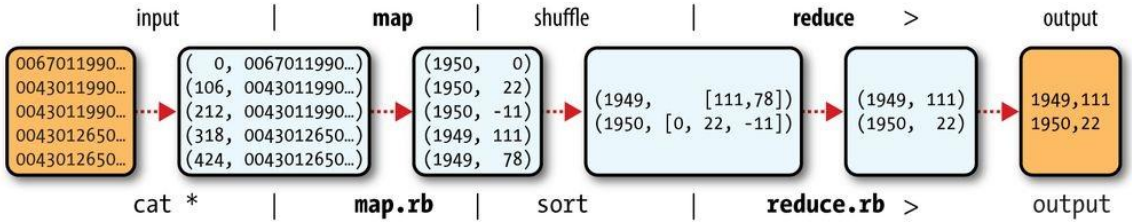

Abbildung 1.3: Der MapReduce Dataflow(15, Seite 24, Figure 2-1)

MapReduce hat immer noch eine Sonderstellung im Hadoop Ecosystem, da es die mitgelieferte Processing Engine ist. Vor Hadoop 2.x war es sogar die einzige Möglichkeit, Daten in einem Hadoop-Cluster zu verarbeiten. Eine MapReduce-Applikation zu entwickeln erfordert allerdings das Schreiben vielen Java-Codes und Problemstellungen müssen irgendwie in die Phasen von Mapping und Reducing übertragen werden, selbst wenn andere Modellierungen der Problemstellung intuitiver oder leichter zu bearbeiten wären. Außerdem ist MapReduce allein für **Batch Processing** ausgelegt. Das heißt, dass man eine MapReduce-Applikation

schreibt, die eine ganz bestimmte Fragestellung zu einem Datensatz beantwortet. Diese wird ausgeführt und erst nachdem alle Daten verarbeitet wurden, sieht man ein Ergebnis. Dies kann viele Minuten, Stunden oder sogar Tage dauern. Will man nun einen Parameter der Fragestellung ändern (zum Beispiel nicht mehr nach Monaten sondern nach Wochen aufgeschlüsselt), muss man die gesamte Verarbeitung des Datensatzes noch einmal durchführen. Dies steht im Konflikt mit dem heute üblichen Bedürfnis nach **interaktiver Datenexploration**. Zuguterletzt ist MapReduce nach heutigen Standards eher langsam. Da es für die Ausführung auf Commodity Hardware entwickelt wurde, schreibt und liest es die Zwischenergebnisse der einzelnen Phasen immer wieder von der Festplatte des DataNodes. Neue Processing Engines (allen voran Apache Spark [6]), setzen viel auf **In-Memory Processing**, halten alle Daten also möglichst während der gesamten Bearbeitungszeit im Arbeitspeicher. Das ermöglicht bis zu 40 mal schnellere Abfragen bei gleichwertigem Arbeitsaufwand. [vgl. 16, Kap. 3.19]

1.2.3 YARN

In Version 1.x von Hadoop war MapReduce sowohl für die Verarbeitung der Daten, als auch für die Ressourcenzuteilung im Cluster zuständig. Das bedeutete, dass man zwingend das MapReduce-Programmiermodell nutzen musste, um die im Hadoop Cluster gespeicherten Daten auszuwerten. Die Ressourcenverwaltung war damit ein mögliches Bottleneck, da sie bei mehreren parallel laufenden Jobs auf einem Node um Rechenzeit mit der Datenverarbeitung konkurrieren musste und neue Jobs gegebenfalls lange nicht gestartet wurden.[16] Die größte Änderung in Hadoop 2.x war dann die Ausgliederung der Ressourcenverwaltung aus MapReduce und die Einführung einer dedizierten Ressourcenverwaltungsanwendung - **YARN** - *Yet Another Ressource Negotiator*. YARN teilt eingehenden Jobs Cluster-Ressourcen zu und startet fehlgeschlagene Jobs gegebenenfalls neu. Ähnlich wie das HDFS bringt YARN eine Reihe von Prozessen mit sich, die Master- und Worker-Rollen einnehmen. Auf dem vom HDFS designierten NameNode läuft der **Resource Manager**. Dieser unterteilt sich wiederum in **Application Manager** und **Scheduler**. Auf allen DataNodes laufen jeweils ein **Node Manager** und ein **Application Master**.[16]

Durch das Zusammenspiel dieser Prozesse bietet sich dem Anwender ein Interface zur verteilten Ausführung von Anwendungslogik, bei dem man sich nicht an das MapReduce-Programmiermodell halten muss. Startet man in Hadoop 2.x eine MapReduce-Applikation, ist diese eigentlich eine YARN-Applikation, bei der einem schon ein Teil des Programmieraufwands abgenommen wurde. Eine eigene YARN-Applikation zu schreiben bedeutet hingegen, sich selbst um die logische Aufteilung der Daten zu kümmern, Cluster-Ressourcen wie CPU und RAM in Form sogenannter **Container** von YARN anzufordern und dafür zu sorgen, dass der auszuführende Programmcode für alle DataNodes (am besten gespeichert im HDFS) verfügbar ist. YARN reiht die Ausführung der angeforderten Container auf verschiedenen DataNodes in Warteschlangen ein, kopiert den Anwendungscode aus dem HDFS auf diese Nodes und überwacht die erfolgreiche Ausführung der Anwendung.

[6] https://spark.apache.org/

1.2.4 Setup

Zur Installation von Hadoop kann man die offizielle Distribution[7] benutzen und komplett selbst konfigurieren. Dabei besteht die Möglichkeit, Hadoop in drei verschiedenen Modi zu betreiben: **Single Node**, **Pseudo-distributed** und **Fully-distributed**. Weiterhin haben diverse kommerzielle Anbieter wie Cloudera[8] eigene Hadoop Distributionen entwickelt, die sie in Form von vorkonfigurierten VM- oder Docker-Images teilweise kostenlos zur Verfügung stellen. Cloudera zum Beispiel ergänzt diese Distributionen aber mittlerweile durch Cloudlösungen[9]. Cloudanbieter wie Google und Microsoft bieten fertig konfigurierte und voll verwaltete Cluster in ihren jeweiligen Cloudumgebungen an (Google Dataproc[10] und Azure HDInsight[11]). Dabei kann man statt des HDFS zur Datenhaltung die jeweiligen Cloud Storage Systeme (Google Cloud Storage[12] und Azure Storage / Azure Data Lake Storage[13]) nutzen. In diesem Abschnitt sollen anhand eines fertigen VM Images von Cloudera die Interaktion mit dem HDFS und der Workflow für eine MapReduce-Applikation demonstiert werden.

Single Node Setup

Defaulteinstellung des Hadoop Downloads Installation auf der einen beteiligten Maschine Start eines Single Node Clusters lokal Erste Übung zum Umgang mit dem HDFS MapReduce Job in depth

Fully-distributed Cluster

Installation von Hadoop auf allen beteiligten Maschinen Einrichtung von passwordless ssh auf allen Maschinen Evtl. Anpassung der /etc/hosts auf allen Maschinen Editieren der ganzen Konfigurationsdateien (XML) und kopieren der gleichen Dateien auf alle beteiligten Maschinen Editieren der Worker Datei auf dem NameNode NameNode (HDFS) formatieren Ausführen der Skripte auf dem NameNode

Hadoop in der Cloud

Google Dataproc, Azure HDInsights Oftmals eigenes Dateisystem, fully managed Grafische Oberfläche zum Submitten von Jobs, etc.

[7] https://hadoop.apache.org/releases.html
[8] https://de.cloudera.com/
[9] https://de.cloudera.com/products/cloudera-data-platform.html
[10] https://cloud.google.com/dataproc
[11] https://azure.microsoft.com/en-us/products/hdinsight/#overview
[12] https://cloud.google.com/blog/products/storage-data-transfer/hdfs-vs-cloud-storage-pros-cons-and-migration-tips
[13] https://learn.microsoft.com/en-us/azure/hdinsight/hdinsight-hadoop-architecture

2 ETL mit Pig

Auch wenn es vermehrt von Spark verdrängt wird

2.1 Anwendungsfälle

Welche neuen Dinge ermöglicht dieses Tool Eine Zeile Pig Latin entspricht vielen Zeilen MapReduce

2.2 Architektur

2.3 Pig Latin

2.4 Praxis

2.4.1 Hinzufügen zum Cluster

2.4.2 Anwendung auf dem Cluster

3 Data Ingestion

3.1 Sqoop

3.2 Flume

4 Datawarehousing mit Hive

4.1 Anwendungsfälle

Welche neuen Dinge ermöglicht dieses Tool

4.1.1 Unterschiede zu Pig

4.2 Architektur

4.3 Interaktion

4.3.1 HiveQL

4.3.2 CLI

4.3.3 Java API

4.4 Praxis

4.4.1 Hinzufügen zum Cluster

4.4.2 Einrichtung einer Datenbank

4.4.3 Einlesen von Daten im CLI

4.4.4 Einlesen von Daten mit Sqoop

4.4.5 Absetzen einer Query

5 NoSQL mit HBase

5.1 Anwendungsfälle

Welche neuen Dinge ermöglicht dieses Tool

5.1.1 CAP-Theorem

5.1.2 ACID und BASE

5.2 Architektur

5.3 Interaktion

5.3.1 HBase Shell

5.3.2 Java API

5.4 Praxis

5.4.1 Hinzufügen zum Cluster

5.4.2 Einrichtung einer Datenbank

5.4.3 Einlesen von Daten

5.4.4 Datenmigration aus einem RDBMS

5.4.5 Absetzen einer Query

6 Streaming mit Kafka

6.1 Anwendungsfälle

Welche neuen Dinge ermöglicht dieses Tool Ersetzt durch Spark Streaming

6.2 Architektur

6.3 Interaktion

6.3.1 Who knows

6.4 Praxis

6.4.1 Hinzufügen zum Cluster

6.4.2 Maybe, vielleicht kann man ja was zeigen

7 Hadoop heute

7.1 Aktuelle Anwendungsbeispiele zu Hadoop

7.1.1 AirBnB

7.2 Apache Spark als Gold Standard

7.2.1 Kann eh alles besser

Literatur

1. *Apache Hadoop* [Apache Hadoop Main Page] [online]. [besucht am 2022-12-07]. Abger. unter: https://hadoop.apache.org/.
2. *Big Data: was es ist und was man darüber wissen sollte* [online]. [besucht am 2022-12-05]. Abger. unter: https://www.sas.com/de_de/insights/big-data/what-is-big-data.html.
3. GRIFFITHS, Richard T. *Search Engines* [History of the Internet] [online]. 2007-06-21. [besucht am 2022-12-05]. Abger. unter: https://web.archive.org/web/20070621143859/http://www.internethistory.leidenuniv.nl/index.php3?m=6&c=7#how.
4. ZAKON, Robert H'obbes'. *Hobbes' Internet Timeline - the definitive ARPAnet & Internet history* [Hobbes' Internet Timeline] [online]. 2018-01-01. [besucht am 2022-12-05]. Abger. unter: https://www.zakon.org/robert/internet/timeline/#Growth.
5. BEAUMONT, David. *How to explain vertical and horizontal scaling in the cloud* [Cloud computing news] [online]. 2014-04-09. [besucht am 2022-12-07]. Abger. unter: https://www.ibm.com/blogs/cloud-computing/2014/04/09/explain-vertical-horizontal-scaling-cloud/.
6. GUSTAFSON, John L. Amdahl's Law. In: PADUA, David (Hrsg.). *Encyclopedia of Parallel Computing*. Boston, MA: Springer US, 2011, S. 53–60. ISBN 978-0-387-09766-4. Abger. unter DOI: 10.1007/978-0-387-09766-4_77.
7. *Horizontal Vs. Vertical Scaling Comparison Guide* [MongoDB] [online]. [besucht am 2022-12-07]. Abger. unter: https://www.mongodb.com/basics/horizontal-vs-vertical-scaling.
8. *Pricing - Linux Virtual Machines Scale Sets | Microsoft Azure* [online]. [besucht am 2022-12-07]. Abger. unter: https://azure.microsoft.com/en-us/pricing/details/virtual-machine-scale-sets/linux/.
9. ATHOW, Desire. *At 100TB, the world's biggest SSD gets an (eye-watering) price tag* [TechRadar] [online]. 2020-07-07. [besucht am 2022-12-07]. Abger. unter: https://www.techradar.com/news/at-100tb-the-worlds-biggest-ssd-gets-an-eye-watering-price-tag.
10. *who was who in SSD? - StorageTek* [online]. [besucht am 2022-12-07]. Abger. unter: http://www.storagesearch.com/storagetek.html.
11. *What is a Computer Cluster? | Answer from SUSE Defines* [SUSE Defines] [online]. [besucht am 2022-12-07]. Abger. unter: https://www.suse.com/suse-defines/definition/computer-cluster/.

12. GHEMAWAT, Sanjay; GOBIOFF, Howard; LEUNG, Shun-Tak. The Google File System. In: *Proceedings of the 19th ACM Symposium on Operating Systems Principles*. Bolton Landing, NY, 2003, S. 20–43.
13. DEAN, Jeffrey; GHEMAWAT, Sanjay. MapReduce: Simplified Data Processing on Large Clusters. In: *OSDI'04: Sixth Symposium on Operating System Design and Implementation*. San Francisco, CA, 2004, S. 137–150.
14. CUTTING, Doug; CAFARELLA, Mike; LORICA, Ben. *The next 10 years of Apache Hadoop* [online]. 2016. [besucht am 2022-12-08]. Abger. unter: https://www.oreilly.com/content/the-next-10-years-of-apache-hadoop/.
15. WHITE, Tom. *Hadoop: the definitive guide*. Fourth edition. Beijing: O'Reilly, 2015. ISBN 978-1-4919-0163-2. OCLC: ocn904818464.
16. FREIKNECHT, Jonas; PAPP, Stefan. *Big Data in der Praxis: Lösungen mit Hadoop, Spark, HBase und Hive ; Daten speichern, aufbereiten, visualisieren*. 2., erweiterte Auflage. München: Hanser, 2018. ISBN 978-3-446-45396-8.
17. INFRABOT, ASF. *HowManyMapsAndReduces* [HADOOP2 - Apache Software Foundation] [online]. 2019-07-09. [besucht am 2022-12-10]. Abger. unter: https://cwiki.apache.org/confluence/display/HADOOP2/HowManyMapsAndReduces.