Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Daniel Kirchner

Skalierbare Datenanalyse mit Apache Spark

Fakultechnik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Daniel Kirchner

Skalierbare Datenanalyse mit Apache Spark

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultechnik und Informatik
der Hochschule fr Angewandte Wissenschaften Hamburg

Betreuender Prfer: Prof. Dr. Kahlbrandt
Zweitgutachter: Prof. Dr. Zukunft

Eingereicht am: 1. Januar 2345

Daniel Kirchner

Thema der Arbeit
Skalierbare Datenanalyse mit Apache Spark

Stichworte
Schlüsselwort 1, Schlüsselwort 2

Kurzzusammenfassung
Dieses Dokument …
Daniel Kirchner

Title of the paper
Scalable Data Analysis with Apache Spark

Keywords
keyword 1, keyword 2

Abstract
This document …

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Listings

1 Einführung

1.1 Motivation

Die Entwicklung und Verbesserung von Frameworks zur Verarbeitung großer Datenmengen ist zur Zeit hochaktuell und zunehmend im Fokus von Medien und Unternehmen. Verschiedene Programme und Paradigmen konkurrieren um die schnellste, bequemste und stabilste Art großen Datenmengen einen geschäftsfördernden Nutzen abzuringen Verweis??? .

Mit dem Begriff „große Datenmengen“ oder „Big Data“ werden in dieser Arbeit solche Datenmengen zusammengefasst, die die Kriterien Volume, Velocity, Variety[1] erfüllen oder „Datenmengen, die nicht mehr unter Auflage bestimmter Service Level Agreements auf einzelnen Maschinen verarbeitet werden können“ (Vgl. [SW14]).

Als Unternehmen, das früh mit solchen Datenmengen konfrontiert war implementierte Google das Map-Reduce Paradigma[2] als Framework zur Ausnutzung vieler kostengünstiger Rechner für verschiedene Aufgaben (u.a. Indizierung von Webseiten und PageRank[3]).

In Folge der Veröffentlichung dieser Idee im Jahr 2004 wurde Map-Reduce in Form der OpenSource Implementation Hadoop (gemeinsam mit einer Implementation des Google File Systems GFS, u.a.)[4] zum de-facto Standard für Big-Data-Analyseaufgaben.

Reines Map-Reduce (nach Art von Hadoop) als Programmierparadigma zur Verarbeitung großer Datenmengen zeigt jedoch in vielen Anwendungsfällen Schwächen:

- Daten, die in hoher Frequenz entstehen und schnell verarbeitet werden sollen erfordern häufiges Neustarten von Map-Reduce-Jobs. Die Folge ist kostspieliger Overhead durch Verwaltung/Scheduling der Jobs und gegebenenfalls wiederholtem Einlesen von Daten.

[1] [Lan01].
[2] [DG04].
[3] [Pag01].
[4] [SG03].

- Algorithmen die während ihrer Ausführung iterativ Zwischenergebnisse erzeugen und auf vorherige angewiesen sind (häufig bei Maschinenlernalgorithmen) können nur durch persistentes Speichern der Daten und wiederholtes Einlesen zwischen allen Iterationsschritten implementiert werden.
- Anfragen an ein solches Map-Reduce-System erfolgen imperativ in Form von kleinen Programmen. Dieses Verfahren ist offensichtlich nicht so intuitiv und leicht erlernbar wie deklarative Abfragesprachen klassischer Datenbanken (z.B. SQL).

In der Folge dieser Probleme entstanden viele Ansätze dieses Paradigma zu ersetzen, zu ergänzen oder durch übergeordnete Ebenen und High-Level-APIs zu vereinfachen[5].

Eine der Alternativen zu der klassischen Map-Reduce-Komponente in Hadoop ist die „general engine for large-scale data processing“ Apache Spark.

Ein Indiz für das steigende Interesse an diesem Produkt liefert unter anderem ein Vergleich des Interesses an Hadoop und Spark auf Google:

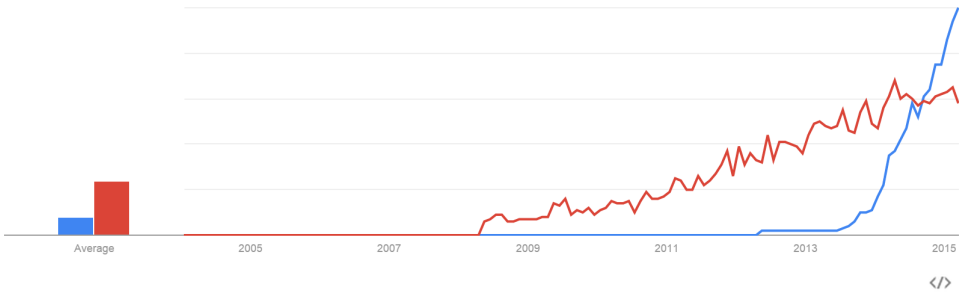

Abbildung 1.1: Suchanfragen zu Spark (*blau*) und Hadoop (*rot*), Stand 24.03.2014 [Goo]

1.2 Kontextabgrenzung

Das Ziel dieser Arbeit ist es die grundlegenden Konzepte und Möglichkeiten von Apache Spark zu untersuchen und ausgewählte Aspekte im Rahmen konkreter Anwendungen zu betrachten.

[5] [SR14].

Für ein tieferes Verständnis werden Installation, Cluster-Betrieb und die Modellierung und Implementation von Treiberprogrammen beispielhaft durchgeführt, dokumentiert und bewertet. Hierbei kommt Apache Spark in der Version 1.3.0 zum Einsatz.

Nur am Rande wird betrachtet:

- Der Vergleich mit ähnlichen Produkten
- Die empirische Analyse des Skalierungsverhaltens
- Details zu Installation und Betrieb

Apache Spark ist überwiegend in der Programmiersprache Scala geschrieben. Die Beispiele in dieser Arbeit werden ebenfalls in Scala verfasst um

1. einen einheitlichen Stil und Vergleichbarkeit zwischen Quellcode-Auszügen und eigenen Beispielen zu gewährleisten.
2. Ausdrücke in kurzer, prägnanter Form darzustellen.

2 Vorstellung von Apache Spark

Aus Sicht eines Nutzers ist Apache Spark eine API zum Zugriff auf Daten und deren Verarbeitung.

Diese API (wahlweise für die Programmiersprachen Scala, Java und Python verfügbar), kann im einfachsten Fall über eine eigene Spark Konsole mit Read Evaluate Print Loop[1] verwendet werden.
Die Zählung von Wortvorkommen in einem Text - das „Hello World“ der Big Data Szene - lässt sich dort mit zwei Befehlen realisieren (Listing 2.1).

```
$ ./spark-shell
[...]
      / __/__  ___ _____/ /__
     _\ \/ _ \/ _ `/ __/  '_/
    /___/ .__/\_,_/_/ /_/\_\   version 1.3.0
       /_/
Using Scala version 2.10.4 (OpenJDK 64-Bit Server VM, Java 1.7.0_75)
Type in expressions to have them evaluated.
[...]
scala> val text = sc.textFile("../Heinrich Heine - Der Ex-Lebendige")
[...]
scala> :paste
text.flatMap(line => line.split(" "))
.map(word => (word, 1))
.reduceByKey(_ + _)
.collect()
[...]
res0: Array[(String, Int)] = Array((Tyrann,,1), (im,2), (Doch,1) ...)
```

Listing 2.1: Word Count in der Spark Konsole

[1] [MH99].

Aus Sicht eines Administrators oder Softwarearchitekten ist Apache Spark eine Applikation auf einem Rechnercluster, die sich in der Anwendungsschicht befindet und charakteristische Anforderungen insbesondere an Lokalität des Storages und die Netzwerkperformance stellt.

Was das konkret bedeutet, welche Mechanismen und Konzepte dahinterstehen und in welchem Ökosystem von Anwendungen sich Apache Spark bewegt wird in den folgenden Abschnitten dieses Kapitels beleuchtet.

2.1 Überblick

Im Allgemeinen Fall läuft eine Spark-Anwendung auf drei Arten von Rechnern (s. Abb. 2.1):

1. **Clientknoten**
   Auf Nutzerseite greift die Anwendung auf die API eines lokalen Spark-Kontextes zu, der die Kontaktdaten eines Clustermanagers sowie verschiedene Konfigurationseinstellungen enthält.

2. **Masterknoten**
   Der Masterknoten betreibt den *Clustermanager* läuft auf einem entfernten Rechner und ist der Einstiegspunkt in den Rechnercluster. Hier werden Aufträge des Anwenders an die Arbeitsknoten verteilt und Ergebnisse eingesammelt und zurückgereicht.

3. **Workerknoten**
   Die Workerknoten beherbergen die Spark *Executors* und sind die ausführenden Elemente der Aktionen und Transformationen. Die *Executors* können untereinander Zwischenergebnisse austauschen und melden ihre Ressourcenverfügbarkeit an den *Clustermanager*.

Um die Architektur und Optimierungskonzepte eines verteilten Systems bewerten zu können ist es offensichtlich wichtig, welche Eigenschaften der unterliegenden Hardware angenommen werden.

Weil Spark explizit für den Betrieb innerhalb eines Hadoop/YARN [VERWEIS auf Abschnitt Scheduling] geeignet ist und YARN wiederum für den Betrieb auf einem Rechnercluster auf Mittelklasse-Mehrzweckmaschinen (Commodity Hardware) optimiert ist[2], kann für Spark von einer vergleichbaren Hardwarekonfiguration ausgegangen werden.

[2] [AM14].

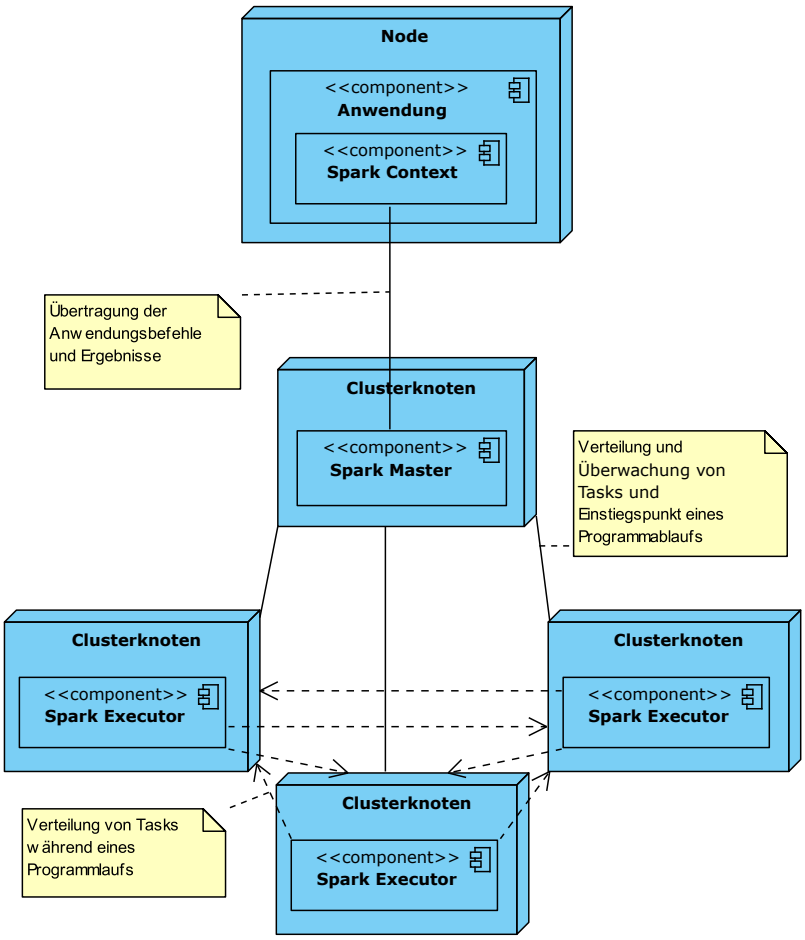

Abbildung 2.1: Verteilungsdiagramm einer typischen Sparkinstallation

Der Vergleich von drei aktuellen Rack Servern der 2000-Euro-Klasse (in der Grundausstattung) - hier als Mittelklasse-Geräte bezeichnet - liefert die folgenden Verhältnisse der wesentlichen Schnittstellen zueinander (Siehe Anhang 3.1).

Netzwerk	Festspeicher	Arbeitsspeicher
0,125 GB/s	1 GB/s	17 GB/s

Tabelle 2.1: Theoretische Spitzenleistungen bei Mittelklasse-Servern

Auf eine detaillierte Analyse des Zugriffsverhaltens wird im Rahmen dieser Arbeit verzichtet. Bei den folgenden Bewertungen der Kernkonzepte ist es wichtig sich die aus Tabelle

2.1 abgeleiteten Größenordnungen des Durchsatzes (D) der verschiedenen Datenkanäle zu vergegenwärtigen:

$$D_{Netzwerk} < D_{Festspeicher} < D_{Arbeitsspeicher}$$

Für eine effiziente Verarbeitung von Daten ist es - ganz allgemein - also wünschenswert den größten Anteil des Datentransfers im Arbeitsspeicher zu haben, einen kleineren Anteil auf der Festplatte und einen noch kleineren Anteil auf Netzwerkverbindungen.

Es ist das oberste Ziel der folgenden Kernkonzepte von Apache Spark unter diesen Bedingungen die effiziente Verarbeitung *großer Datenmengen*[3] zu gewährleisten.

2.2 Kernkonzepte

Die wichtigste Abstraktion des Applikationskerns sind sogenannte Resilient Distributed Datasets (RDDs)[4].

RDDs sind eine Variante von Distributed Shared Memory (DSM)[5] [MZ12].

2.2.1 Nutzung von Arbeitsspeicher

2.2.2 Nutzung von persistentem Speicher

2.2.3 Nutzung von CPUs

2.2.4 Scheduling/Shuffling

2.2.5 Kern-API

Transformationen

Aktionen

[3] [SW14].
[4] [MZ12].
[5] [NL91].

2.3 Standardbibliotheken

— Warum ist Spark so einfach (und wo vielleicht nicht)? —

Die vier Standardbibliotheken erweitern die Kern-API für bestimmte, häufig genutzte Aufgaben aus Bereichen der Datenanalyse.

Die bedienten Bereiche sind

- Deklaratives Abfragen auf strukturierten Datensätzen (*Spark SQL*)
- Maschinenlernverfahren (*MLlib*)
- Echtzeitbehandlung von eingehenden Daten (*Streaming*)
- Operationen auf Graph-Strukturen (*GraphX*)

2.3.1 Dataframes/Spark SQL

2.3.2 MLlib

2.3.3 Streaming

2.3.4 GraphX

2.4 Betrieb und Security

2.5 Spark im Kontext von Parallelisierungspattern

— Buch: Algorithms and Parallel Computing —

2.6 Entwicklergemeinschaft

— Herkunft, Apache Foundation, Entwicklungsphilosophien, Anzahl Entwickler, ... —

2.7 Verwandte Produkte

— Ergänzende oder konkurrierende Produkte —

2.7.1 YARN

2.7.2 Mesos

2.7.3 Flink

2.7.4 MPI

2.7.5 Kafka

2.7.6 HBase

3 Anwendung von Spark zur Datenanalyse

Im Folgenden wird Apache Spark im Rahmen zweier grundsätzlich verschiedener Anwendungsfälle betrachtet.

Beispiel 1: Eine typische Anwendung mit verteilten lokalem Storage (HDFS) und Spark als „Client“ eines bestehenden Yarn Clustermanagers. — Commodity Hardware (Rasperry Pi Cluster). —

Beispiel 2: Eine untypische Anwendung mit verteiltem entfernten Storage und dem Spark Standalone Clustermanager. — HPC Hardware („Thunder“ des Hamburger KlimaCampus). —

3.1 Echtzeitbewertung von Twitter-Meldungen nach ihrer Relvanz für Spark

— Fusion von Tweets und Mailinglisten https://spark.apache.org/docs/1.3.0/mllib-feature-extraction.html Implementation auf einem Raspberry Pi Cluster mit HDFS und Yarn Clustermanager —

3.1.1 Problembeschreibung

— Es sollen die beiden Spark Mailingslisten (Developer, User) zur Identifikation relevanter und aktueller Themen genutzt werden. Mit den so bewerteten Begriffen können wiederum Tweets bewertet werden. Mit den Tweets können dann ganze Accounts nach ihrer Relevanz beurteilt werden. —
— Zwei Datenquellen: Tweets (Nahe-Echtzeit), Entwickler-Emails (Sporadisch) —
— Stichworte: HDFS, Yarn, Rasperri Pi Cluster, Machine Learning, Feature Extraction, Big Data Life Cycle —

Es sollen aktuelle Informationen bereitgestellt werden, die besonders für Nutzer und Entwickler des Softwareproduktes Apache Spark relevant sind.

3.1.2 Anforderungen

Für die Software soll folgende lose Sammlung funktionaler und nicht-funktionaler Anforderungen gelten. Mit *Information* ist jeweils eine Auflistung von Twitter-Meldungen gemeint.

- **A1: Zugriff auf die Information**
  Der Zugriff soll über eine grafische Benutzerschnittstelle erfolgen und keine Konfigurationen benötigen.
- **A2: Aktualität der Information**
  Es sollen stets Informationen dargestellt werden, die unmittelbar zuvor entstanden sind und in Quasi-Echtzeit[1] verarbeitet wurden.
- **A3: Relevanz der Information**
  Die Relevanz soll an aktuellen Themen der Entwicklergemeinschaft gemessen werden.

3.1.3 Hardwarekontext und Performance-Basisdaten

Als Testumgebung dient ein Rechnercluster aus vier identischen Workern und einem speziellen Masterknoten (Abb. 3.1).

Worker Rasperry Pi 2

- CPU: 900MHz Quad-Core ARM Cortex A7
- RAM: 1GB SDRAM
- Ethernet: 100MBit/s
- Festspeicher: SDHC Class 4 Speicherkarte 16GB

Als Betriebssystem kommt das Debian-Derivat Raspbian[2] 32-Bit zum Einsatz.

[1] Eine Latenz von unter einer Minute sei hier tolerabel
[2] [Ras].

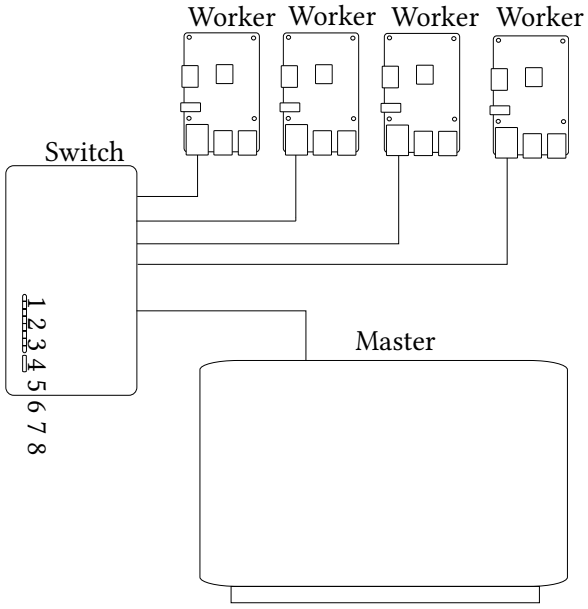

Abbildung 3.1: Hardwareumgebung des Programms zur Tweetanalyse

Master Dell d420

- CPU: 1,2 GHz Core2 Duo U2500
- RAM: 2GB DDR2 SDRAM
- Ethernet: 100MBit/s
- Festspeicher: 60GB 4200RPM Hard Drive

Als Betriebssystem komm Ubuntu[3] 14.04 32-Bit zum Einsatz.

Netzwerk Vernetzt sind die Rechner mit RJ45 über einen TP-Link TL-SF1008D Switch mit maximalem Durchsatz von 100MBit/s.

3.1.4 Lösungsskizze

Wahl des Dateisystems — HDFS —

Wahl des Cluster-Managers — YARN —

[3] [Ubu].

Tabelle 3.1: —DUMMY— Netzwerkdurchsatz

Nachrichtengröße	Worker → Worker	Master → Worker	Worker → Master
1kB	50ms	837ms	970ms
64kB	47ms	877ms	230ms
1MB	31ms	25ms	415ms
64MB	35ms	144ms	2356ms

Verteilungsdiagramm

Komponentendiagramm

Laufzeitdiagramme

3.1.5 Ergebnis und Bewertung

Dashboard

Laufzeitverhalten

Bewertung und Probleme

3.2 Evaluierung einer spark-basierten Implementation von CDOs auf einem HPC Cluster mit nicht-lokalem Storage

— Implementation ausgewählter CDOs (sehr wenige, möglicherweise nur 1-2) mit der Core-API von Spark. Testlauf auf einem HPC Cluster mit nicht-lokalem, allerdings per Infiniband angeschlossenen Storage. Insbesondere Betrachtung des Skalierungsverhaltens und der „Sinnhaftigkeit“. —

3.2.1 Beschreibung des Problems

— Erläuterung von CDOs (Climate Data Operators). —

3.2.2 Hardwarekontext und Performance-Basisdaten

— hier kommen die eingesetzten Systeme, und relevante Laufzeitmessungen (netzwerk, storage, cpu) hin —

3.2.3 Architekturübersicht

— hier kommen Verteilungs- und Komponentendiagramm hin —

3.2.4 Detailierte Lösungsbeschreibung

— hier kommen laufzeitdiagramme und codeschnipsel hin —

3.2.5 Ergebnisse

— Tabellen und Diagramme Ergebnissen, evt. Skalierungsverhalten — — Bewertung —

4 Schlussbetrachtung

4.1 Kritische Würdigung der Ergebnisse

4.2 Ausblick und offene Punkte

Acronyme

RDD Resilient Distributed Dataset. 7

Glossar

Master Host, der Verwaltungsaufgaben innerhalb eines Rechnerclusters übernimmt und dazu mit hierarchisch untergeordneten Rechnern kommuniziert. Zu den Aufgaben kann insbesondere das Verteilen von Arbeitsaufträgen oder Speicherblocks gehören. 11

Read Evaluate Print Loop Pattern zum Erzeugen einer Konsole, die in einer Endlosschleife Eingaben liest, die auswertet und das Ergebnis wieder ausgibt. 4

Rechnercluster Vernetzter Verbund aus vollwertigen Rechnern. 5, 11

RJ45 Achtpolige Modularsteckverbindung zur Datenübertragung. 12

Service Level Agreement Übereinkunft zwischen dem Anbieter und dem Nutzer eines Dienstes über dessen Qualität (z.B. Antwortzeiten, Durchsatz, Verfügbarkeit, etc.). 1

Worker Host, der als Arbeitsknoten in einem Rechnercluster dient. Falls nicht anders beschrieben ist hier ein Rechner gemeint, der seine Ressourcen einer Spark-Anwendung zur Verfügung stellt und mit seinem Festspeicher Teil eines verteilten Dateisystems ist. 11

Literatur

[AM14] Vinod Kumar Vavilapalli Arun Merthy. *Apache Hadoop YARN*. 2014, S. 42.

[DG04] Jeffrey Dean und Sanjay Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters". In: *OSDI* (2004).

[Fuj] *Fujitsu PRIMERGY SERVER - Basics of Disk I/O Performance*. 2011. URL: http://globalsp.ts.fujitsu.com/dmsp/Publications/public/wp-basics-of-disk-io-performance-ww-en.pdf.

[Goo] Google. *Google Trends*. URL: https://www.google.com/trends.

[Lan01] Doug Laney. "3D Data Management: Controlling Data Volume, Velocity and Variety". In: *Application Delivery Strategies* (2001).

[Law14] Jason Lawley. *Understanding Performance of PCI Express Systems*. 2014.

[MH99] Karl Knight Max Hailperin Barbara Kaiser. *Concrete Abstractions: An Introduction to Computer Science Using Scheme*. 1999, 278ff.

[MZ12] et al. Matei Zaharia Mosharaf Chowdhury. "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing". In: *NSDI* (2012).

[NL91] Bill Nitzberg und Virginia Lo. "Distributed Shared Memory: A Survey of Issues and Algorithms". In: *Computer* 24.8 (Aug. 1991), S. 52–60. ISSN: 0018-9162. DOI: 10.1109/2.84877. URL: http://dx.doi.org/10.1109/2.84877.

[Pag01] Lawrence Page. *Method for node ranking in a linked database*. US Patent 6,285,999. 2001. URL: http://www.google.com/patents/US6285999.

[Ras] Raspbian. *Raspbian Operating System*. URL: http://www.raspbian.org.

[SG03] Shun-Tak Leung Sanjay Ghemawat Howard Gobioff. *The Google File System*. Techn. Ber. Google, 2003.

[SR14] Dilpreet Singh und Chandan K Reddy. "A survey on platforms for big data analytics". In: *Journal of Big Data* (2014).

[SW14] Jasson Venner Sameer Wadkar Madhu Siddalingaiah. *Pro Apache Hadoop*. 2014, S. 1.

[Ubu] Ubuntu. *Ubuntu Operating System*. URL: http://www.ubuntu.com.

Anhang

1 Installation der Plattform

2 Quellcode (Auszüge)

3 Sonstiges

3.1 Einschätzung des theoretischen Spitzendurchsatzes von Mittelklasse-Servern

Um zu einer groben Einschätzung des möglichen Datendurchsatzes verschiedener Schnittstellen bei „Commodity Servern“ zu gelangen, wurden drei Systeme von großen Herstellern ausgewählt.
In der Grundkonfiguration kosten diese Systeme (zum Zeitpunkt dieser Arbeit) um die € 2000,- und lassen damit auf die Größenordnungen bei dem Datendurchsatz bestimmter Schnittstellen bei preisgünstigen Mehrzweck-Rechenknoten schließen.

Modell	Netzwerkschnittstelle	Interner Bus für Festspeicher	Arbeitsspeicher
Dell PowerEdge R530	1Gb/s Ethernet	PCIe 3.0	DDR4
HP Proliant DL160 Gen8	1Gb/s Ethernet	PCIe 3.0	DDR3
System x3650 M5	1Gb/s Ethernet	PCIe 3.0	DDR4

Tabelle 1: Theoretische Spitzenleistungen bei Mehrzweck-Servern der 2000 Euro Klasse

Mit [Law14] und [Fuj] lassen sich grobe obere Abschätzungen errechnen, die in Tabelle 2.1 angegeben sind.

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 1. Januar 2345 Daniel Kirchner