



Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Daniel Kirchner

Skalierbare Datenanalyse mit Apache Spark
Beispielimplementation eines Influenza-Frühwarnsystems

*Fakultät Technik und Informatik
Studiendepartment Informatik*

*Faculty of Engineering and Computer Science
Department of Computer Science*

Daniel Kirchner

Skalierbare Datenanalyse mit Apache Spark
Beispielimplementation eines Influenza-Frühwarnsystems

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Kahlbrandt
Zweitgutachter: Prof. Dr. Zweitprüfer

Eingereicht am: 1. Januar 2345

Daniel Kirchner

Thema der Arbeit

Skalierbare Datenanalyse mit Apache Spark Beispielimplementation eines Influenza-Frühwarnsystems

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

| | | |
|----------|--|----------|
| 1 | Einführung | 1 |
| 1.1 | Motivation | 1 |
| 1.2 | Kontextabgrenzung | 2 |
| 1.3 | Relevante Produkte und Meilensteine | 2 |
| 1.3.1 | Überblick | 2 |
| 1.3.2 | Big Table | 2 |
| 1.3.3 | Map/Reduce | 2 |
| 1.3.4 | Hadoop | 2 |
| 2 | Vorstellung von Apache Spark | 3 |
| 2.1 | Übersicht | 3 |
| 2.1.1 | Architekturübersicht | 3 |
| 2.1.2 | Standardbibliotheken | 3 |
| 2.2 | Wesentliche Konzepte | 3 |
| 2.2.1 | Abgrenzung zu Hadoop | 3 |
| 2.2.2 | Resilient Distributed Datasets | 3 |
| 3 | Vorstellung des Beispiels | 4 |
| 3.1 | Aufgabenbeschreibung | 4 |
| 3.2 | Lösungsidee | 4 |
| 3.2.1 | 1. Schritt: Ähnlichkeitsmaß für Wörter erzeugen | 4 |
| 3.2.2 | 2. Schritt: Echtzeitbewertung von Textnachrichten aus einem Datenstrom | 4 |
| 4 | Implementation und Bewertung | 5 |
| 4.1 | Betriebsumgebung | 5 |
| 4.1.1 | Verteilungsdiagramm | 7 |
| 4.2 | Architekturübersicht | 7 |
| 4.3 | Architekturdetails | 7 |
| 4.3.1 | Modell für Ähnlichkeit von Wörtern mit MLlib erzeugen | 7 |
| 4.3.2 | Einlesen von Nachrichten aus dem Twitter Livestream | 7 |
| 4.3.3 | Verarbeiten und Bewerten der Nachrichten | 7 |
| 4.4 | Bewertung der Verfahren | 7 |
| 5 | Schlussbetrachtung | 8 |
| 5.1 | Kritische Würdigung der Ergebnisse | 8 |
| 5.2 | Ausblick und offene Punkte | 8 |

Listings

1 Einführung

1.1 Motivation

Der Bedarf auf großen Datenmengen zu operieren ist nicht neu. Spätestens seit in den späten Neunzigerjahren Suchmaschinenanbieter mit Mengen von Daten und Anfragen konfrontiert wurden, die nicht mehr wirtschaftlich durch einzelne Rechner oder kleine Rechnerverbünde zu bewältigen waren, wurden neue Verfahren benötigt. Algorithmen wurden auf Verteilbarkeit und Fehlertoleranz optimiert.

Die Menge verfügbarer Daten wuchs und wurde auch für Unternehmen und Einrichtungen interessant, deren Kerngeschäft nicht die Daten selbst sind. Regierungen, Wissenschaft, Industrieunternehmen, Militär, Börsenhändler und viele andere treffen Entscheidungen auf Grundlage von Daten die die Kapazitäten einzelner Maschinen weit überschreiten.

Häufige Veränderung und Unvorhersehbarkeit der Anforderungen sind alltäglich. Daten denen man zunächst keine Bedeutung beimisst, können sich später als sehr kritisch erweisen und in anderem Zusammenhang eine wichtige Rolle spielen. Ein Verfahren hat sich in diesem Rahmen als besonders wirksam herausgestellt: Daten werden schwach strukturiert gespeichert und erst zum Zeitpunkt einer Abfrage (oder beliebigen Operation) in eine jeweils sinnvolle Struktur gebracht.

Solche Abfragen werden bei aktuellen Systemen oft in Form kleiner Programme gestellt, die auf den Knoten eines Clusters verteilt und nach einem vorgegebenem Paradigma ausgeführt werden. Hier kommt das Cluster-Computing-Framework Apache Spark ins Spiel, das mehrere solcher Paradigmen vereint und als bisher vielseitigste Lösung für viele aktuelle Aufgaben der „Big Data“ Analyse antritt.

1.2 Kontextabgrenzung

1.3 Relevante Produkte und Meilensteine

1.3.1 Überblick

1.3.2 Big Table

1.3.3 Map/Reduce

1.3.4 Hadoop

2 Vorstellung von Apache Spark

2.1 Übersicht

2.1.1 Architekturübersicht

2.1.2 Standardbibliotheken

Spark SQL

MLlib

Streaming

GraphX

2.2 Wesentliche Konzepte

2.2.1 Abgrenzung zu Hadoop

2.2.2 Resilient Distributed Datasets

3 Vorstellung des Beispiels

3.1 Aufgabenbeschreibung

3.2 Lösungsidee

3.2.1 1. Schritt: Ähnlichkeitsmaß für Wörter erzeugen

3.2.2 2. Schritt: Echtzeitbewertung von Textnachrichten aus einem Datenstrom

4 Implementation und Bewertung

4.1 Betriebsumgebung

Die Wahl einer Betriebsumgebung für Cluster-Computing-Frameworks stellt besondere Anforderungen. Obwohl Installationen einer lauffähigen Instanz von Apache Spark auch für einzelne Maschinen möglich ist, werden solche Setups einer Demonstration und Untersuchung von verteiltem Rechnen offensichtlich nicht gerecht.

Auf der anderen Seite bringen verteilte und auf physikalischen Rechnern laufende Installationen zusätzliche Probleme mit. Ausfälle von Hardwarekomponenten auf Rechenknoten und Netzwerken, Stromkosten für Betrieb und Kühlung, sowie Beschaffungs- oder Mietkosten für Räumlichkeiten und Komponenten sind nur die offensichtlichsten.

Im Rahmen dieser Arbeit sollen Konzepte demonstriert und bewertet werden. Ein produktionsstaugliches System ist nicht gefordert. Es ergeben sich die folgenden Vorgaben.

Anforderungen:

- echte Verteilung der Komponenten
- volle Kontrolle über die lokalen Abläufe

Keine Anforderungen:

- hohe Performance
- authentische Bedingungen eines Rechenzentrums

Gemäß dieser Anforderungen kommt ein KVM-virtualisierter Host zum Einsatz der als Plattform für eine Cloud Computing Software dient, um ein kleines Cluster von virtuellen Maschinen zu Verfügung zu stellen.

Technische Daten des Hosts

- CPU Intel® Xeon® E5-2670V2
- 4 dedizierte Kerne
- 16GB DDR3 RAM
- Betriebssystem Fedora 20 GNU/Linux
- Cloud Software OpenStack (Release Juno)

Das Cluster selbst besteht aus 3 gleichartigen virtuellen Maschinen innerhalb eines OpenStack Tenants und dem Host selbst. Um die untersuchte Installation in andere Betriebsumgebungen zu bringen oder schnell wiederherstellen zu können, sind die Installationen der Apache Spark Komponenten jeweils in Docker Containern (basierend auf LXC) durchgeführt. Dadurch daraus ergeben sich drei Laufzeitumgebungen für den Entwicklungs- und Untersuchungsprozess:

- Lokale Single-Node-Installation: Exploration, Entwicklung, Debugging
- [Docker] Virtuelles Cluster auf einzelner physikalischen Maschine: Entwicklung und Debugging im verteilten Betrieb
- [Docker] Virtuelles Cluster auf verteilten physikalischen Maschinen: Performance-Messungen, Analyse von Skalierungsverhalten

Technische Daten der virtuellen Cluster-Knoten

- 2 virtuelle CPUs
- 2GB RAM
- Betriebssystem CoreOS
- Netzwerkvirtualisierung mit der OpenStack-Komponente Neutron

4.1.1 Verteilungsdiagramm

4.2 Architekturübersicht

4.3 Architekturdetails

4.3.1 Modell für Ähnlichkeit von Wörtern mit MLlib erzeugen

4.3.2 Einlesen von Nachrichten aus dem Twitter Livestream

4.3.3 Verarbeiten und Bewerten der Nachrichten

4.4 Bewertung der Verfahren

5 Schlussbetrachtung

5.1 Kritische Würdigung der Ergebnisse

5.2 Ausblick und offene Punkte

See also [One und Two](#) (2010).

Literaturverzeichnis

[One und Two 2010] ONE, Author ; TWO, Author: A Sample Publication. (2010)

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
