Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Daniel Kirchner

Skalierbare Datenanalyse mit Apache Spark

Architekturanalyse und Performancetests in verschiedenen Anwendungsfällen

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Daniel Kirchner

Skalierbare Datenanalyse mit Apache Spark

Architekturanalyse und Performancetests in verschiedenen Anwendungsfällen

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 Architekturanalyse und Performancetests in verschiedenen Anwendungsfällen

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

Listings

1 Einführung

1.1 Motivation

Die Entwicklung und Verbesserung von Frameworks zur Verarbeitung großer Datenmengen ist zur Zeit hochaktuell und sehr im Fokus von Medien und Unternehmen [VERWEIS]. Verschiedene Programme und Paradigmen konkurrieren um die schnellste, bequemste und stabilste Art großen Datenmengen einen geschäftsfördenden Nutzen abzuringen.

Unter dem Begriff „große Datenmengen“ oder „Big Data“ werden solche Datenmengen zusammengefasst, die die Kriterien Volume, Velocity, Variety [VERWEIS, Doug Laney] erfüllen oder „Datenmengen, die nicht mehr unter Auflage bestimmter SLAs auf einzelnen Maschinen verarbeitet werden können“ [VERWEIS, Hadoop/Yarn Entwickler].

Als Unternehmen, das früh mit solchen Datenmengen konfrontiert war implementierte Google das Map-Reduce Paradigma [VERWEIS] als Framework zur Ausnutzung vieler „Mittelklasse“-Rechner um Webseiten einzustufen und für andere Aufgaben [VERWEIS].

In Folge der Veröffentlichung ihrer Idee im Jahr 2005 [VERWEIS] wurde Map-Reduce in Form der OpenSource Implementation Hadoop (gemeinsam mit einer Implementation des Google File Systems GFS, u.a.) [VERWEIS] zum de-facto Standard für Big-Data-Analyseaufgaben [VERWEIS?].

Map-Reduce als Programmierparadigma zur effizienten Stapelverarbeitung großer Datenmengen zeigt jedoch in vielen Anwendungsfällen Schwächen:

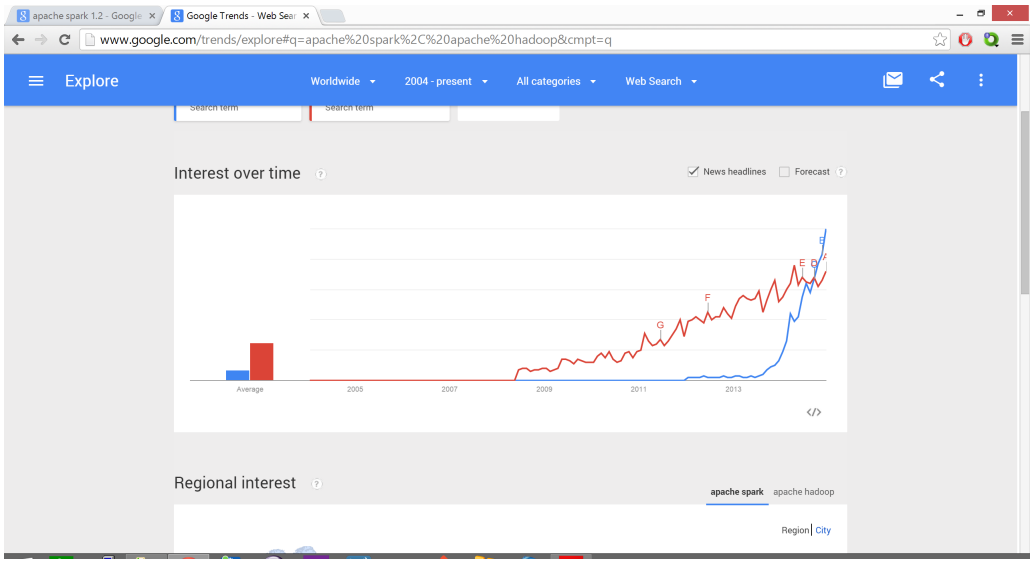
- Daten, die in hoher Frequenz verändert werden erfordern das ständige Neustarten eines Map-Reduce-Jobs. Iterative Algorithmen sind also nicht vorgesehen.
- Die Anfrage an ein solches System erfolgt in Form von kleinen Programmen. Dieses Verfahren ist offensichtlich nicht so deklarativ und leicht erlernbar wie SQL-Anfragen an klassische Datenbanken.

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

- [VERWEIS: A survey of large scale...] oder Aufzählung.

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

Zum aktuellen Zeitpunkt hat das Interesse an Apache Spark das Apache Hadoop Framework bereits überholt:



1.2 Kontextabgrenzung

1.3 Verwandte Produkte

1.3.1 Überblick

1.3.2 YARN

1.3.3 Flink

1.3.4 ...

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 produktionstaugliches 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 physiklischen 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