FernUniversität in Hagen

–

Seminar 01912 / 19912
im Sommersemester 2017

„Skallierbare verteilte Datenanalyse“

Thema 2.3

Spark

Referent: Lukas Wappler

Inhaltsverzeichnis

1 Einleitung

2 Apache Spark

Apache Spark ist ein Open Source Framework, dass ermöglicht verteilt über ein Cluster Programme und Algorithmen auszuführen. Zusätzlich ist das Prgrammierodell bzw. die API zum schreiben solcher Programme sehr einfach und elegant gehalten. [Ryz+15]

Das Framework ist im Rahmen eine Forschungsprojekts entstanden. Das Forschungsprojekt wurde 2009 in der Universtiy of California in Berkeley im sogenannten AMPLab[1] ins Leben gerufen. Seit 2010 steht es als Open Source Software unter der BSD-Lizenz [2] zur Verfügung. Das Projekt wird seit 2013 von der Apache Software Foundation[3] weitergeführt. Seit 2014 ist es dort als Top Level Projekt eingestuft. Zum aktuellen Zeitpunkt steht Apache Spark unter der Apache 2.0 Lizenz[4] zur Verfügung.

2.1 Kern-Bibliotheken / Komponenten

Apache Spark besteht im wesentlichen aus fünf Modulen: Spark Core, Spark SQL, Spark Streaming, MLlib Machine Learning Library und GraphX.
Während Spark Core die Kern-Komponente bildet und alle notwendigen Bausteine für das Framework mitbringt sind die anderen Module auf dem Spark Core Module aufbauen und befassen sich mit spezielleren Bereichen wie SQL, Streaming, maschinelles Lernen oder Graphenberechnungen. In Abbildung 2.1 ist noch einmal eine Übersicht der Komponenten.
Die Module werden in den folgenden Kapitel von 2.1.1 bis 2.1.5 näher beleuchtet.

Darüber hinaus wird in Kapitel 2.1.6 SparkR vorgestellt. Das Module gehört nicht direkt, jedoch bietet es interessante Möglichkeiten Datenanylsen mit R zu optimieren bzw. zu beschleunigen.

[1] AMPLab: ist ein Labor der Berkeley Universität in Californien, die sich auch Big-Data Analysen spezialisiert hat

[2] BSD-Lizenz (Berkeley Software Distribution-Lizenz): bezeichnet eine Gruppe von Lizenzen, die eine breitere Wiederverwertung erlaubt.

[3] Apache Software Foundation: Ist eine ehrenamtlich arbeitende Organisation, die die Apache-Projekte fördert.

[4] Apache 2.0 Lizenz: Die Software darf frei verwendet und verändert werden. Zusätzlich gibt es nur wenige Auflagen.

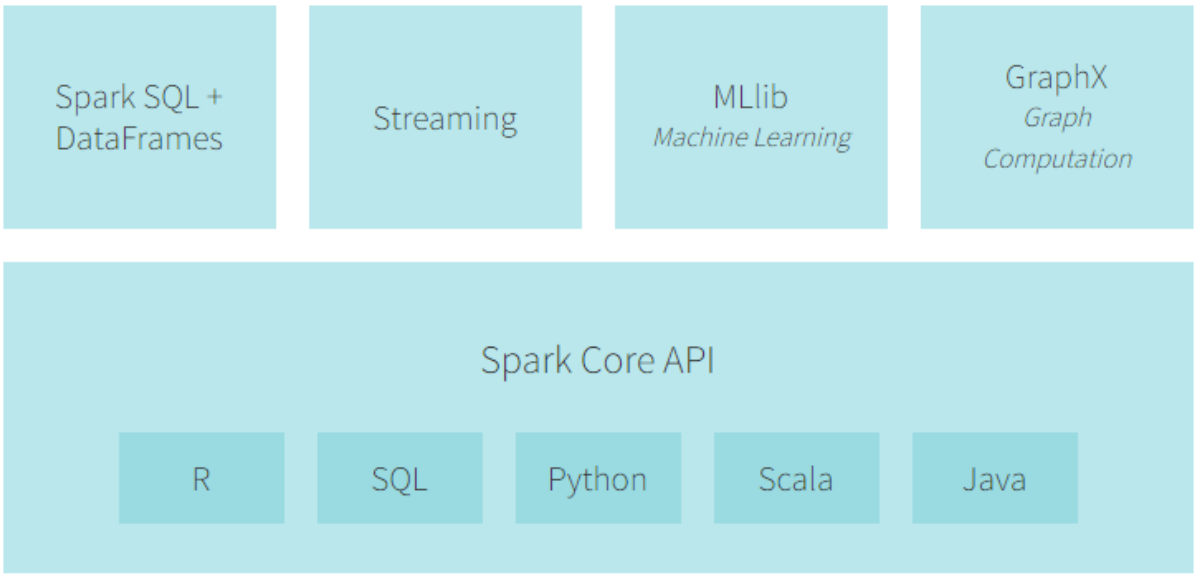

Abbildung 2.1: Spark Core

2.1.1 Grundlage des Systems (Spark-Core & RDD's)

Spark Core ist Grundlage der Spark Plattform. Alle anderen Komponenten bauen auf diesen Kern auf. Alle grundlegenden infrastrukturellen Funktionen sind darin enthalten. Darunter zählen diue Aufgabenverwaltung, das Scheduling sowie I/O Funktionen. Der Kern liefert zum Beispiel die Möglichkeit der Berechnungen direkt im Arbeitsspeicher. Das grundlegende Programmiermodell wie das Arbeiten mit den RDD's und die API's für die verschiedenen Sprachen (Java, Scala und Python). [5]
In der Abbildung 2.1 sind die einzelnen Bausteine innerhalb der Spark Core Komponenten / API zu sehen.
Die Paralelle Verarbeitung wird über den Spark Context realisiert. Der Spark Context wird im eigentlich Programm erzeugt und ist in der Regel dann mit einem Cluster Manager verbunden. Dieser wiederum kennst alle Workernode, die dann die eigentlichen Aufgaben ausführen. Die Abbildung 2.2 zeigt wie Spark Context, Cluster Manager und die Worker Node's zusammen agieren. Damit verteilt über viele Nodes Aufgaben wird eine Datenstrukturen benötigt, die dafür ausgelegt sind. [6]
Resilent Distirbuted Datasets (RDD's) zu deutsch elastische, verteilte Datensätze ist die primäre Datenabstraktion in Apache Spark. Ein RDD entspricht einer partitionierten Sammlung an Daten. Somit können die Partitionen auf verschiedene Systeme (bzw. Worker) verteilt werden.
Nach der Erstellung sind RDD's nur lesbar. Es ist also nur möglich ein einmal definiertes RDD durch Anwendungen von globalen Operationen in ein neues RDD zu überführen. Die Operationen werden dann auf allen Partitionen des RDD's angewendet.
Man unterscheidet bei den Operationen zwischen Transformationen (z.B.: filter oder join) und Aktionen (z.B.: reduce, count, collect oder foreach). Transformationen bilden ein RDD auf ein anderes RDD ab. Aktionen bilden ein RDD auf eine andere Domäne ab.

[5] Vgl. [Fou17b]
[6] Vgl. [ER16, S. 101]

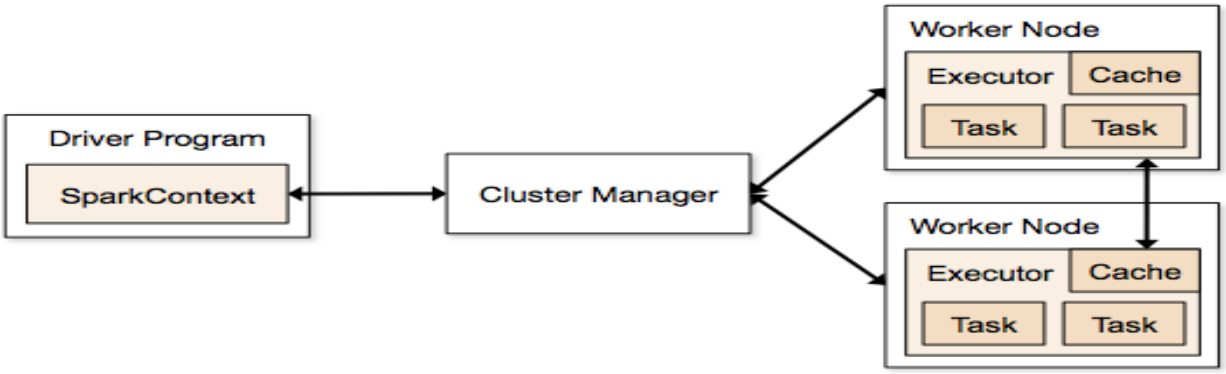

Abbildung 2.2: Spark Cluster aus [Fou17d]

Eine Folge von Operationen wird Lineage [7] eines RDD's genannt.[8]

[7] RDD Lineage: Logischer Ablaufplan der einzelnen Operationen. Hilft Daten wiederherzustellen falls Fehler aufgetreten sind.
[8] Vgl. [Zah+12]

2.1.2 SQL-Abfragen mit (Spark-SQL & Data Frames)

Spark-SQL wurde 2014 veröffentlicht. Die Komponente gehört zu den Komponenten aus der Spark-Familie, die am meisten weiterentwickelt werden.

Dabei kombiniert es zwei wesentliche Dinge. Zum einen ermöglicht es relationale Querys zu schreiben und zum anderen prozedurele Algorithmen einzusetzen. Bisher wurden beide Aktionen nacheinander von verschiedenen Systemen realisiert.

SparkSQL entstammt dem Apache-Shark. Man wollte die Probleme die es in Apache Shark gab lösen.

1. Mit Apache Shark ist es nur möglich auf Daten im Hive Katalog zuzugreifen.
2. Shark lässt sich nur über selbst geschriebene SQL's aufrufen.
3. Hive ist nur für MapReduce optimiert

Mit ApacheSQL hat man erreicht auf relationale Daten zuzugreifen. Es wurde eine hohe Performance aufgrund etablierter DBMS-Techniken erreicht. Neue Datenquellen lassen sich leicht anschließen und integrieren. Zusätzliche Erweiterungen wie Maschine Learning und Graph Processing sind nutzbar.

Todo, auf Dataframe API eingehen. Todo, auf Catalyst eingehen.

Gerade bei SQL ist es enorm wichtig sich für die richtigen Anweisungen zu entscheiden um keine langsamen Operationen zu haben. Hier gibt es sehr große Geschwindigkeitsunterschiede.

2.1.3 Verarbeitung von Datenströmen (Spark-Streaming)

Die Spark-Streaming Bilbliothek ermöglicht das Verarbeiten von Datenströmen. Auch hier dienen die RDD's als Grundlage. Die RDD's werden DStreams erweitert. DStreams (discretized streams) sind Objekte, die Informationen enthalten, die in Verbindung mit Zeit stehen. DStreams sind intern eine Sequenz von RDD's und werden aus diesem Grund diskrete Streams genannt. Auch DStreams haben die bereits aus 2.1.1 bekannten zwei Operationen (Transformation und Aktion).

Um Datenströme zu empfangen wird ein Empfänger (Receiver) auf einem Worker-Knoten gestartet. Die eingehenden Daten werden in keinen Datenblöcken gespeichert. Dafür werden die Daten innerhalb eines vorgegebenen Zeitfenster gepuffert. Pro Zeitfenster werden die Daten in dem Puffer in eine Partition eines RDD abgelegt.[9]

In der Spark-Streaming Bilbliothek sind bereits einige Empfänger wie Kafka[10], Twitter[11] oder TCP-Sockets[12] enthalten.
In der Abbildung 2.3 ist der Ablauf vom Eingang der Daten über die Verarbeitung bis hin zur Ausgabe bzw. Speicherung dargestellt.

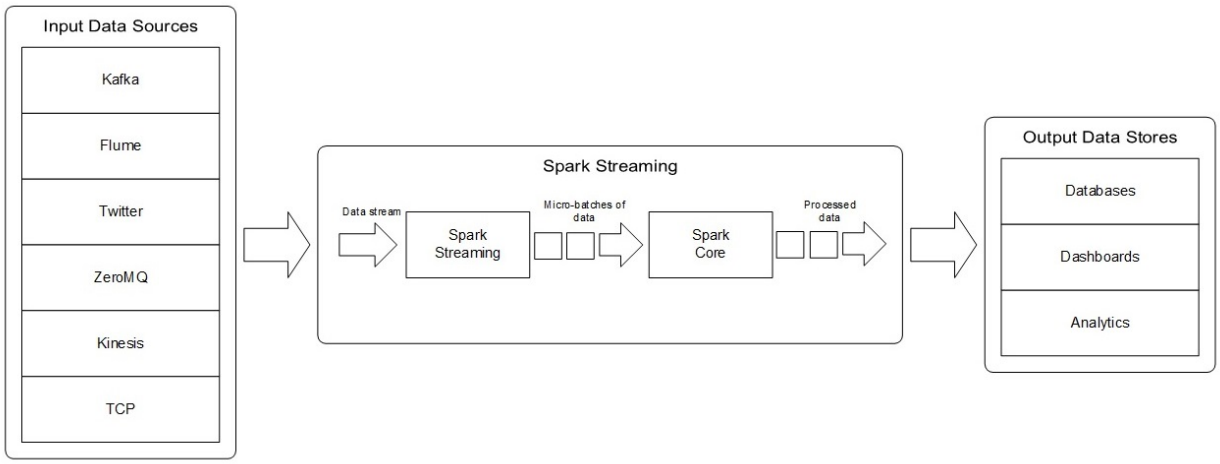

Abbildung 2.3: Spark Streaming Ablauf [Fou17c]

[9] Vgl. [ER16]
[10] Apache Kafka dient zur Verarbeitung von Datenströmen dient.
[11] Twitter: ist ein Mikrobloggingdinst. Nutzer können über das Portal Kurznachrichten verbreiten.
[12] TCP-Sockets: Sind Kommunkationsendpunkte, die zur Netzwerkkommunikation genutzt werden.

2.1.4 Berechnungen auf Graphen (GraphX)

2.1.5 Maschinelles Lernen (MLlib)

2.1.6 Skalierung von R Programmen (SparkR)

2.2 Mehrere Komponenten im Verbund

2.3 Performance

Analysen von Performance Probleme erweisen sich mitunter als sehr schwierig. Apache Spark bringt zwar die seiteneffektfreie API mit, jedoch kann trotzdem eine Menge schief gehen. Es ist schwer immer im Hinterkopf zu behalten, dass Operationen auf vielen verteilten Rechnern ablaufen.

Über eine Webbasierte Übersicht ist es Möglich Informationen zu dann aktuell laufenden Auswertungen und Dauer von Ergebnissen etc. zu bekommen. [13]

Todo Beispiel Bild erstellen und erklären. Es gibt ein Live-Dashboard. Es gibt einen Stack-Trace Button

2.3.1 Besonderheiten bei der Speichernutzung

Zusätzlich wird oft unterschätzt, dass die Wahl einer geeigneten bzw. speichereffizienten Datenstruktur sehr viel bewirken kann. Spark geht davon aus, eine Datei in Blöck einer bestimmten Größe geladen wird. In der Regel 128MB. Zu beachten ist jedoch, dass beim dekomprimieren größer Blöcke entstehen können. So können aus 128Mb schnell 3-4GB große Blöcke werden.

Um das Speichermanagement zu verbessern wurde ein per-node allocator implementiert. Dieser verwaltet den Speicher auf einer Node. Der Speicher wir in drei Breiche geteilt. Speicher zum verarbeiten der Daten. Speicher für die hash-tables bei Joins oder Aggretaions Speicher für ünrollingBlöcke, um zu prüfen ob die einzulesenden Blöcke nach dem entpacken immer noch klein genug sind damit diese gecached werden können.

Damit läuft das System robust über eine großen Bereich.

2.3.2 Netzwerk und I/O-Traffic

Mit Spark wurde schon Operationen bei denen über 8000 Nodes involviert waren und über 1PB an Daten verarbeitet wurden durchgeführt. Das beansprucht natürlich die I/O Schicht enorm.

Um I/O Probleme zu vermeiden, bzw. diese besser in den Griff zu bekommen wurde als Basis das Netty-Framework[14] verwendet.

- Zero-copy I/O:
  Daten werden direkt von der Festplatte zu dem Socket kopiert. Das vermeidet Last an der CPU bei Kontextwechseln und entlastet zusätzlich den JVM[15] garbage collector[16]
- Off-heap network buffer management:
  Todo
- Mehrfache Verbindungen:
  Jeder Sparker worker kann mehrere Verbdingungen parallel bearbeiten.

[13] Vgl. [Ryz+15, S. 12]
[14] Netty: High-Performance Netzwerk Framework
[15] JVM: Todo
[16] garbage collector: Todo

2.4 Nutzung & Verbreitung

Durch die Unterstützung der drei Programmiersprachen skala, pathon und java ist arbeit mit Apache Spark einfacher, als wenn es nur einen einzige exotische Programmiersprache zur Nutzung gäbe.
Apache Spark unterstützt zudem noch verschiedene Datenquellen und Dateiformate. Zu den Datenquellen zählen die das Dateisystem S3[17] von Amazon und das HDFS[18]. Die Dateiformate können strukturiert (z.B.: CSV, Object Files), semi-strukturiert (z.B.: JSON) und unstrukturiert (z.B.: Textdatei) sein.

Unter den Mitwirkenden(Contributors) zählen über 400 Entwickler aus über 100 Unternehmen, Stand 2014
Es gibt über 500 produktive Installationen.

Seit einigen Jahren finden weltweit jährlich unter dem Namen Spark Summit Konferenzen statt. [Fou17a]

Heise.de beauftrage 2015 eine Umfrage in der 2136 Teilnehmer befragt wurden [Sch15]. Diese gaben an, dass 31% Prozent den Einsatz derzeit prüfen. 13% Nutzen bereits Apache Spark und 20% planten den Einsatz noch in dem damaligen Jahr. Scala lag als Programmiersprache mit großen Abstand vorn. Die Nutzung innerhalb verschiedener Berufsgruppen war sehr ähnlich. Mit 16% lag bei den Telekommunikationsunternehmen der Einsatz am höchsten. Eine detaillierte Übersicht ist in Abbildung 2.4 zu sehen.

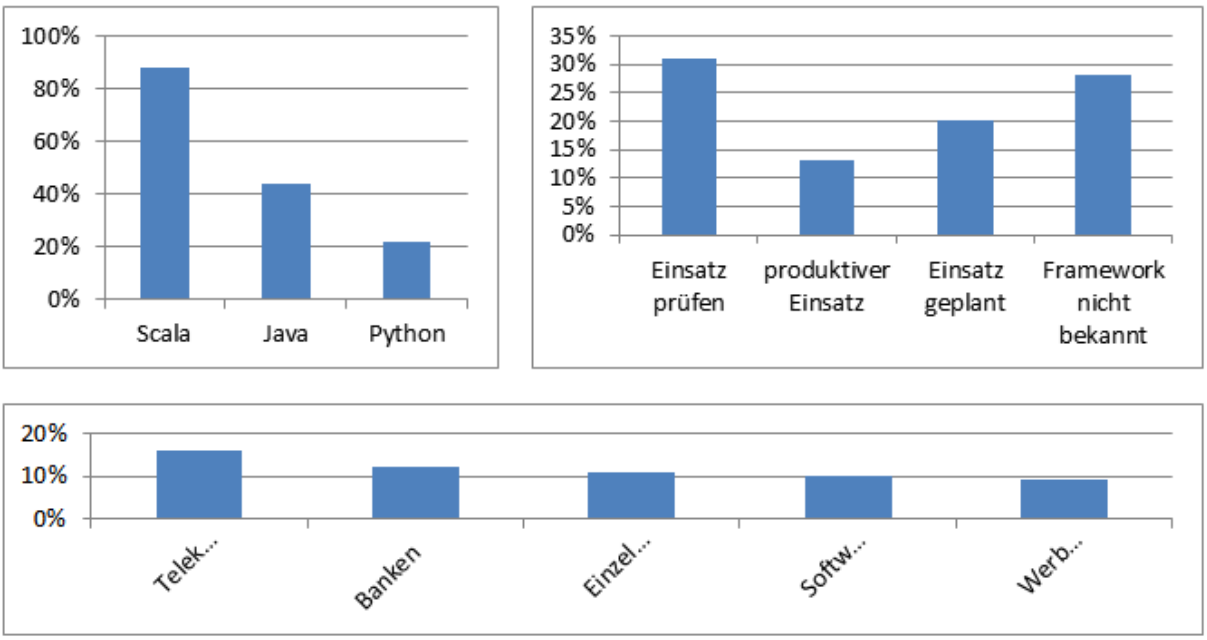

Abbildung 2.4: Einsatz & Verbreitung

[17] S3: Todo
[18] HDFS (Hadoop Distributed File System): Todo

3 Fazit

4 Ausblick & Weiterentwicklung

Immer mehr Firmen führen Apache Spark ein oder Nutzen es bereits. Dieser Trend sollte auch weiterhin so bleiben.
Seit der Einführung von Apachae Spark im Jahr 2010 wird die Software kontinuierlich verbessert und weiterentwickelt. Vieles hat die Community dazu beigetragen, die aufgrund der Open-Source Software dazu in der Lage ist aktiv daran mit zu arbeiten. Auch das wird zukünftig weiter gehen. Im ersten Quartal 2017 gab es über 717 commits.
Der Code liegt auf GitHub[1] und ist öffentlich für jeden zugänglich. Bis zum 10.04.2017 gab es bereits 51 Releases, 19,365 commits und 1,053 contributors[2].
Von der Version 1.6 auf die Version 2.0 gab es nochmal einen relativ starken Performancegewinn. Vermutlich wird man solche Performance steigerungen nicht mehr so leicht erreichen, aber dennoch sollten sich an den Werten auch zukünftig noch etwas nach unten verändern. Eine Übersicht der Performanceänderungen ist in der Tabelle 4.1 zu sehen.[3]

primitive	Spark 1.6	Spark 2.0
filter	15ns	1.1ns
sum w/o group	14ns	0.9ns
sum w/ group	79ns	10.7ns
hash join	115ns	4.0ns
sort (8-bit entropy)	620ns	5.3ns

Tabelle 4.1: cost per row (single thread)

Zukünftig ist denkbar, das noch weitere Komponenten so wie zum Beispiel SparkR dazu kommen. Auch das Anbinden weiterer Datenquellen wird sicherlich weiter gehen

[1] GitHub: Todo
[2] contributors: Todo
[3] Vgl. [Inc17]

5 Anhang

6 Literaturverzeichnis

[Zah+12] Matei Zaharia u. a. *Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing*. Forschungsspapier. University of California, Berkeley, 2012.

[Ryz+15] Sandy Ryza u. a. *Advanced Analytics with Spark*. 1005 Gravenstein Highway North: O'Reilly Media, Inc., 2015.

[Sch15] Julia Schmidt. *Big Data: Umfrage zur Verbreitung zu Apache Spark*. Jan. 2015. URL: https://www.heise.de/developer/meldung/Big-Data-Umfrage-zur-Verbreitung-zu-Apache-Spark-2529126.html.

[ER16] Raul Estrada und Isaac Ruiz. *Big Data SMACK*. New York, 233 Spring Street: Springer Science + Business Media, 2016.

[Fou17a] Apache Software Foundation. *Apache Spark Community*. Apr. 2017. URL: http://spark.apache.org/community.html.

[Fou17b] Apache Software Foundation. *Apache Spark Ecosystem*. Apr. 2017. URL: https://databricks.com/spark/about.

[Fou17c] Apache Software Foundation. *Apache Spark Ecosystem*. Apr. 2017. URL: https://www.infoq.com/articles/apache-spark-streaming.

[Fou17d] Apache Software Foundation. *Cluster Mode Overview*. Apr. 2017. URL: https://spark.apache.org/docs/1.1.0/cluster-overview.html.

[Inc17] Databricks Inc. *Technical Preview of Apache Spark 2.0 Now on Databricks*. Apr. 2017. URL: https://databricks.com/blog/2016/05/11/apache-spark-2-0-technical-preview-easier-faster-and-smarter.html.