

 FernUniversität in Hagen

-

Seminar 01912 / 19912  
im Sommersemester 2017

„Skallierbare verteilte Datenanalyse“

Thema 2.3

Spark

Referent: Lukas Wappler

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>3</b>
<b>2</b>	<b>Apache Spark</b>	<b>4</b>
2.1	Kern-Bibliotheken / Komponenten . . . . .	4
2.1.1	Grundlage des Systems (Spark-Core & RDD's) . . . . .	6
2.1.2	SQL-Abfragen mit (Spark-SQL & Data Frames) . . . . .	7
2.1.3	Verarbeitung von Datenströmen (Spark-Streaming) . . . . .	8
2.1.4	Berechnungen auf Graphen (GraphX) . . . . .	9
2.1.5	Maschinelles Lernen (MLlib) . . . . .	10
2.1.6	Skalierung von R Programmen (SparkR) . . . . .	11
2.2	Mehrere Komponenten im Verbund . . . . .	12
2.3	Performance . . . . .	13
2.3.1	Besonderheiten bei der Speichernutzung . . . . .	13
2.3.2	Netzwerk und I/O-Traffic . . . . .	14
2.4	Nutzung & Verbreitung . . . . .	14
<b>3</b>	<b>Fazit</b>	<b>16</b>
<b>4</b>	<b>Ausblick &amp; Weiterentwicklung</b>	<b>17</b>
<b>5</b>	<b>Anhang</b>	<b>18</b>
<b>6</b>	<b>Literaturverzeichnis</b>	<b>19</b>

# 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 Programmiermodell bzw. die API zum schreiben solcher Programme sehr einfach und elegant gehalten. [Ryz+15]

Das Framework ist im Rahmen eines Forschungsprojekts entstanden. Das Forschungsprojekt wurde 2009 in der University of California in Berkeley im sogenannten AMPLab<sup>1</sup> ins Leben gerufen. Seit 2010 steht es als Open Source Software unter der BSD-Lizenz<sup>2</sup> zur Verfügung. Das Projekt wird seit 2013 von der Apache Software Foundation<sup>3</sup> weitergeführt. Seit 2014 ist es dort als Top Level Projekt eingestuft. Zum aktuellen Zeitpunkt steht Apache Spark unter der Apache 2.0 Lizenz<sup>4</sup> 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. Zur Nutzung der Komponenten gibt es eine Umfrage aus dem Jahr 2015 in Abbildung 2.2 zu sehen.

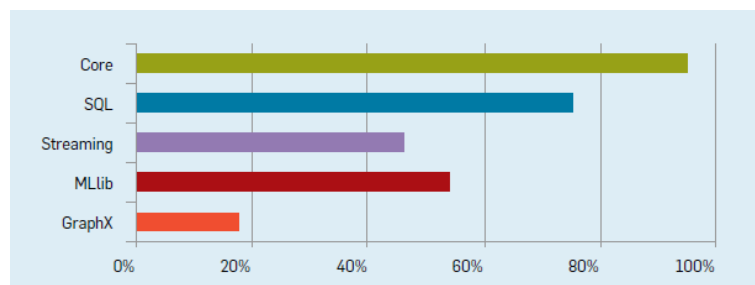


Abbildung 2.1: Nutzung der Komponenten [ZAH+15]

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 2.2 ist noch einmal eine Übersicht der Komponenten.

<sup>1</sup>AMPLab: ist ein Labor der Berkeley Universität in Californien, die sich auch Big-Data Analysen spezialisiert hat

<sup>2</sup>BSD-Lizenz (Berkeley Software Distribution-Lizenz): bezeichnet eine Gruppe von Lizenzen, die eine breitere Wiederverwertung erlaubt.

<sup>3</sup>Apache Software Foundation: Ist eine ehrenamtlich arbeitende Organisation, die die Apache-Projekte fördert.

<sup>4</sup>Apache 2.0 Lizenz: Die Software darf frei verwendet und verändert werden. Zusätzlich gibt es nur wenige Auflagen.

## 2 Apache Spark

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 Datenanalysen mit R zu optimieren bzw. zu beschleunigen.

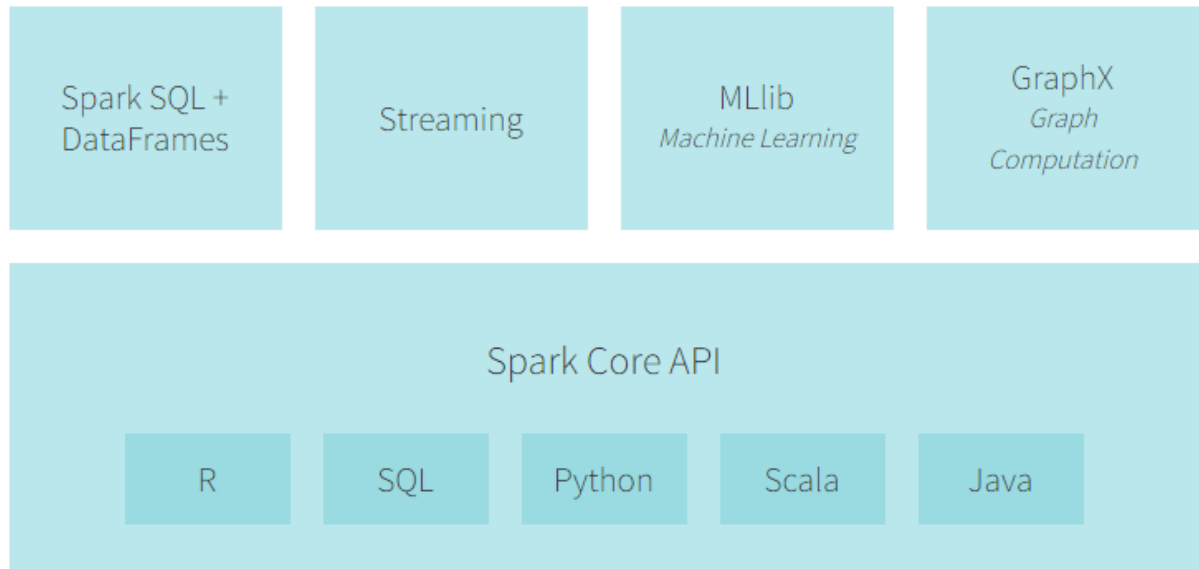


Abbildung 2.2: Spark Core

### 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 die 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).<sup>5</sup>

In der Abbildung 2.2 sind die einzelnen Bausteine innerhalb der Spark Core Komponenten / API zu sehen.

Die parallele Verarbeitung wird über den Spark Context realisiert. Der Spark Context wird im eigentlichen Programm erzeugt und ist in der Regel dann mit einem Cluster Manager verbunden. Dieser wiederum kennt alle Worker Nodes, die dann die eigentlichen Aufgaben ausführen. Die Abbildung 2.3 zeigt wie Spark Context, Cluster Manager und die Worker Nodes zusammen agieren. Damit die Aufgaben über viele Nodes verteilt werden können wird eine Datenstruktur benötigt, die dafür ausgelegt sind.<sup>6</sup>

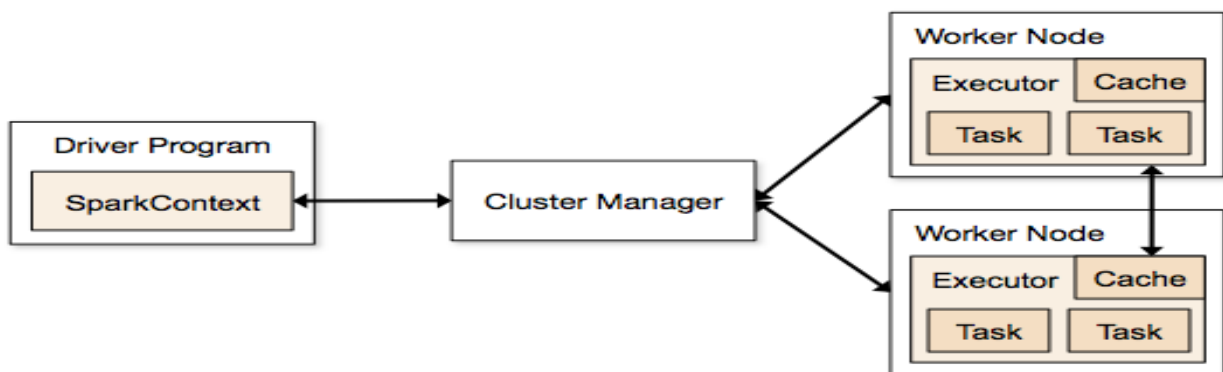


Abbildung 2.3: Spark Cluster aus [Fou17d]

Die Resilient Distributed Datasets (RDD's) zu deutsch elastische, verteilte Datensätze erledigen diese Aufgabe. Es 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 anwendung globaler 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.

Eine Folge von Operationen wird Lineage<sup>7</sup> eines RDD's genannt.<sup>8</sup>

<sup>5</sup>Vgl. [Fou17b]

<sup>6</sup>Vgl. [ER16, S. 101]

<sup>7</sup>RDD Lineage: Logischer Ablaufplan der einzelnen Operationen. Hilft Daten wiederherzustellen falls Fehler aufgetreten sind.

<sup>8</sup>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. Spark-SQL 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<sup>9</sup> Katalog zuzugreifen.
2. Shark lässt sich nur über selbst geschriebene SQL's aufrufen.
3. Hive ist nur für MapReduce optimiert

Es werden zwei wesentliche Anwendungsfälle kombiniert. Zum einen ermöglicht es relationale Querys zu schreiben und zum anderen prozedurale Algorithmen einzusetzen. Dafür werden neben den RDD's die DataFrames als weitere Datenstruktur eingeführt.

Die Abfragen werden zuerst in den DataFrame-Objekten gespeichert. Erst nach der Initialisierung werden diese SQL's dann ausgewertet. Für die Auswertung und Optimierung kommt Catalyst<sup>10</sup> zum Einsatz. Nach der Auswertung werden die Abfragen gegebenenfalls optimiert und danach in Spark-Optionen auf RDD's übersetzt. In Abbildung 2.4 sind die Phasen vom SQL-Query bis hin zu den RDD's dargestellt. In den Boxen mit abgerundeten Ecken befinden sich Catalyst-Trees.

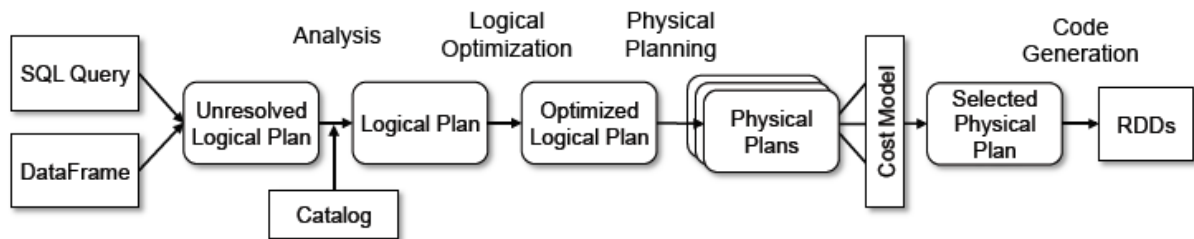


Abbildung 2.4: Phasen der Query Planung in Spark SQL [Arm+15b]

Mit Spark-SQL 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 maschinelles Lernen und Berechnungen von Graphen sind zusätzlich nutzbar.<sup>11</sup>

<sup>9</sup>Apache Hive: Ist eine Erweiterung für Hadoop und ermöglicht Abfragen über SQL zu nutzen.

<sup>10</sup>Catalyst: Ist eine Optimierungseengine für relationale Ausdrücke.

<sup>11</sup>Vgl. [Arm+15b]

### 2.1.3 Verarbeitung von Datenströmen (Spark-Streaming)

Die Spark-Streaming Bibliothek ermöglicht das Verarbeiten von Datenströmen. Auch hier dienen die RDD's als Grundlage. Die RDD's werden zu 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 kleinen 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.<sup>12</sup>

In der Spark-Streaming Bibliothek sind bereits einige Empfänger wie Kafka<sup>13</sup>, Twitter<sup>14</sup> oder TCP-Sockets<sup>15</sup> enthalten.

Abbildung 2.5 zeigt den Ablauf vom Eingang der Daten über die Verarbeitung bis hin zur Ausgabe auf zum Beispiel Dashboard oder der Speicherung in Datenbanken.

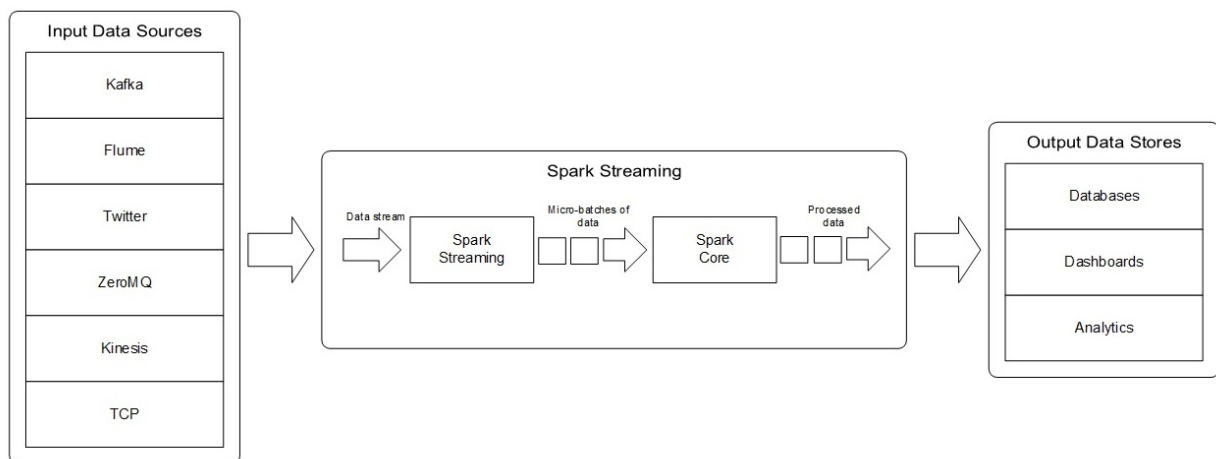


Abbildung 2.5: Spark Streaming Ablauf [Fou17c]

<sup>12</sup>Vgl. [ER16]

<sup>13</sup>Apache Kafka dient zur Verarbeitung von Datenströmen dient.

<sup>14</sup>Twitter: ist ein Mikrobloggingdinst. Nutzer können über das Portal Kurznachrichten verbreiten.

<sup>15</sup>TCP-Sockets: Sind Kommunikationsendpunkte, die zur Netzwirkkommunikation genutzt werden.



### 2.1.4 Berechnungen auf Graphen (GraphX)

Das GraphX Framework ermöglicht die Berechnungen auf Graphen. Die Grundlage sind auch hier die RDD's. Also Graphenstrukturen werden Property-Graphen genutzt. Property-Graphen sind gerichtete Multigraphen. Das heißt der Graph besteht aus Ecken(Knoten) und Kanten. An den Kanten können Eigenschaften hinterlegt sein.

In dem GraphX Framework werden diese Graphen aus Tupeln aus RDDs gebildet. In dem ersten RDD sind die Ecken und in dem zweiten die Kanten enthalten. Um die Graphen auf mehrere Maschinen zu verteilen werden diese entlang der Kante geteilt. Man spricht hier vom sogenannten Edge Cut Verfahren. Eine einzelne Ecke kann somit auf mehreren Maschinen existieren. Um Änderungen an einer Ecke über alle Kopien auf den Maschinen zu propagieren wird zusätzlich eine Routing-Tabelle gepflegt. Über diese sind alle Kopien von Ecken bekannt und bei Änderungen einer Ecke werden alle Maschinen entsprechend informiert. In der folgenden Abbildung 2.6 ist ein verteilter Property-Graph abgebildet. Zusätzlich sind die verschiedenen RDD's für Knoten(Vertex), Kanten(Edge) und die Routing-Tabelle abgebildet.

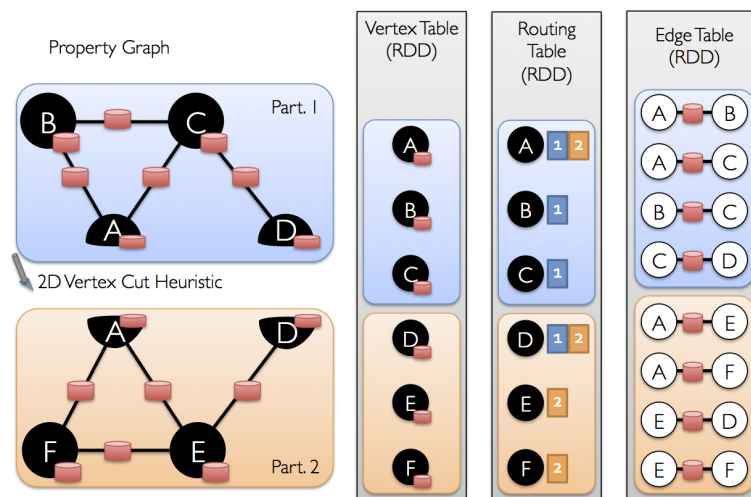


Abbildung 2.6: Property-Graph mit den dazugehörigen RDD's [Fou17e]

### 2.1.5 Maschinelles Lernen (MLlib)

Die Maschinen Lernen Bibliothek (Machine Learning library) bietet die Möglichkeit typische Maschinelle-Lern-Algorithmen auf verteilten Spark-Systemen zu nutzen. Zur Datenabstraktion wird das bereits in 2.1.2 erwähnte DataFrame genutzt.

In einem Maschinenlernprogramm läuft eine Sequenz von Algorithmen (Pipeline) ab um die Daten zu verarbeiten und davon zu lernen. Dafür gibt es in der MLlib Transformers und Estimator als Pipeline-Komponenten). Die Transformers verändern die DataFrames. Das DataFrame wird gelesen, die Daten werden anders strukturiert oder aufbereitet und in einem neuen DataFrame wieder ausgegeben. Diese nutzen die Methode *transform()*

Die Estimators sind Abstraktionen eines Lernalgorithmus. Sie erzeugen Transformer aus dem übergebenen DataFrame. Diese nutzen die Methode *fit()* Eine Pipeline selbst ist wiederum ein Estimator.

Das Zusammenspiel zwischen Transformers und Estimators ist in der Abbildung 2.7 beispielhaft dargestellt. Ein Text wird eingelesen. In den ersten zwei Schritten (Tokenizer und HashingTF) arbeiten Transformatoren. In dem dritten Schritt arbeitet ein Estimator (Logistic Regression)

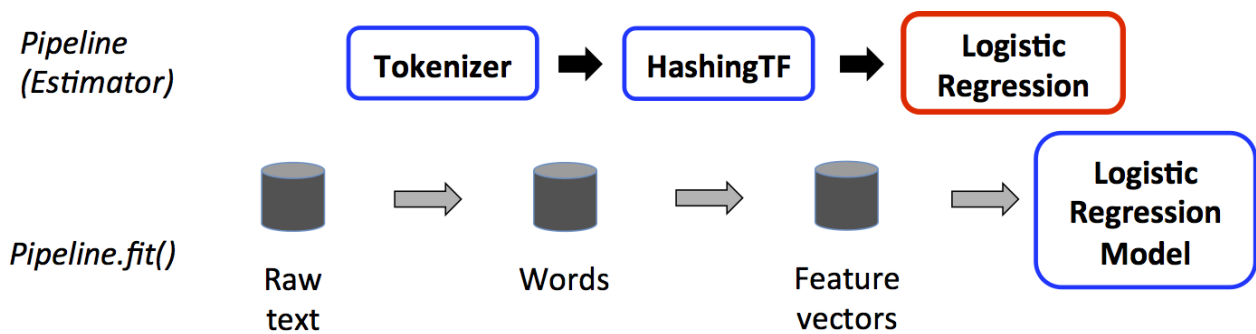


Abbildung 2.7: MLlib Pipeline [Fou17f]

### 2.1.6 Skalierung von R Programmen (SparkR)

SparkR ist eine R Packet das es ermöglicht eine einfache Obefläche bereitstellt um Apache Spark von R aus zu nutzen. Spark R stellt das bereits bekannte DataFrame bereit welche die Operationen wie *selection*, *filtering* oder *aggregation* bereitstellt. Also genau die Operationen die aus R dem Anwender bereits bekannt sind. Für große Datensätze kann SparkR zusätzlich auf maschinelles Lernen über MLlib zurückgreifen.

Um das zu ermöglichen ist eine Brücke von R hin zum Spark Context bzw. den Nodes / Workern notwendig. Das Architekturschaubild in Abbildung 2.8 zeigt diesen Ansatz.

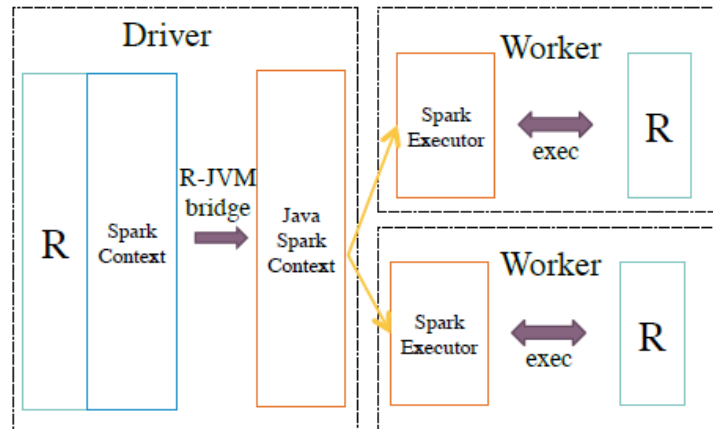


Abbildung 2.8: SparkR Architektur [Ven+15]

R hat den Nachteil, dass es zur Laufzeit nur auf einem einzelnen Thread arbeitet. Diese Hürde kann mit der SparkR Erweiterung genommen werden. Mithilfe der Verarbeitung auf vielen Kernen und zusätzlich des In-Memory Cachings von Spark kann ein sehr großer Laufzeitgewinn erzielt werden <sup>16</sup>

<sup>16</sup>Vgl. [Ven+15]

## **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. Für Entwickler ist es immer schwer im Hinterkopf zu behalten, dass Operationen auf vielen verteilten Rechnern ablaufen.

Über eine Webbasierte Übersicht, die in Abbildung 2.9 zu sehen ist, ist es Möglich Informationen zu dann aktuell laufenden Auswertungen und Dauer von Ergebnissen etc. zu bekommen.

17

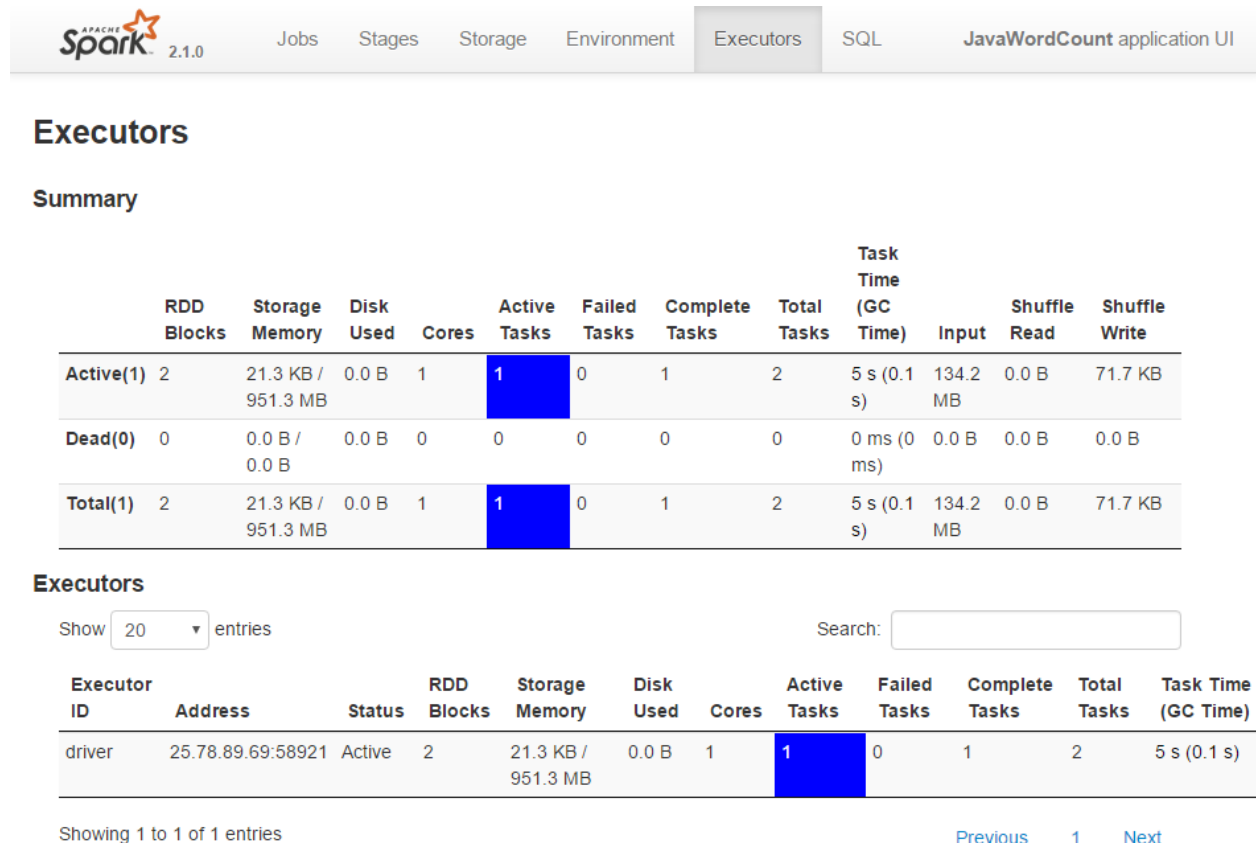


Abbildung 2.9: Spark Web UI: Zusammenfassung der Worker

Speziell beim Thema SQL-Abfragen ist es enorm wichtig sich für die richtigen Anweisungen zu entscheiden um keine langsamen Operationen zu haben. Hier gibt es sehr große Geschwindigkeitsunterschiede.

### 2.3.1 Besonderheiten bei der Speichernutzung

Die Wahl einer geeigneten bzw. speichereffizienten Datenstruktur wird oftmals unterschätzt. Spark geht davon aus, eine Datei in Blöcke einer bestimmten Größe geladen wird. In der Regel 128MB. Zu beachten ist jedoch, dass beim dekomprimieren größere Blöcke entstehen können. So können aus 128MB schnell 3-4GB große Blöcke in dekomprimierten Zustand

<sup>17</sup>Vgl. [Ryz+15, S. 12]

werden.

Um das Speichermanagement zu verbessern wurde ein per-node allocator implementiert. Dieser verwaltet den Speicher auf einer Node. Der Speicher wird in drei Bereiche geteilt. Speicher zum Verarbeiten der Daten. Speicher für die Hash-Tables bei Joins oder Aggregationsen. Speicher für Unrolling-Blöcke, 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 für Anwendungsbereiche mit sehr vielen Nodes sowie mit ganz wenigen.<sup>18</sup>

### 2.3.2 Netzwerk und I/O-Traffic

Mit Apache Spark wurden 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<sup>19</sup> 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<sup>20</sup> Garbage Collector<sup>21</sup>
- Off-heap network buffer management:  
Netty verwaltet einige Speichertabellen außerhalb des Java Heap Speichers um Probleme mit den JVM Garbage Collector zu vermeiden.
- Mehrfache Verbindungen:  
Jeder Spark worker kann mehrere Verbindungen parallel bearbeiten.

## 2.4 Nutzung & Verbreitung

Durch die Unterstützung der drei Programmiersprachen Scala, Python und Java ist die Arbeit mit Apache Spark einfacher, als wenn es nur eine einzige exotische Programmiersprache zur Nutzung gäbe.

Apache Spark unterstützt zudem noch verschiedene Datenquellen und Dateiformate. Zu den Datenquellen zählen die das Dateisystem S3<sup>22</sup> von Amazon und das HDFS<sup>23</sup>. 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.

---

<sup>18</sup>Vgl. [Arm+15a]

<sup>19</sup>Netty: High-Performance Netzwerk Framework

<sup>20</sup>JVM: Todo

<sup>21</sup>garbage collector: Todo

<sup>22</sup>S3 (Simple Storage Service): ist ein Filehosting-Dienst von Amazon der beliebig große Datenmengen speichern kann

<sup>23</sup>HDFS (Hadoop Distributed File System): Ist ein hochverfügbares Dateisystem zur Speicherung sehr großer Datenmengen

Es gibt über 500 produktive Installationen. [Arm+15a]

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

Heise.de beauftragte 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 2.10 zu sehen.

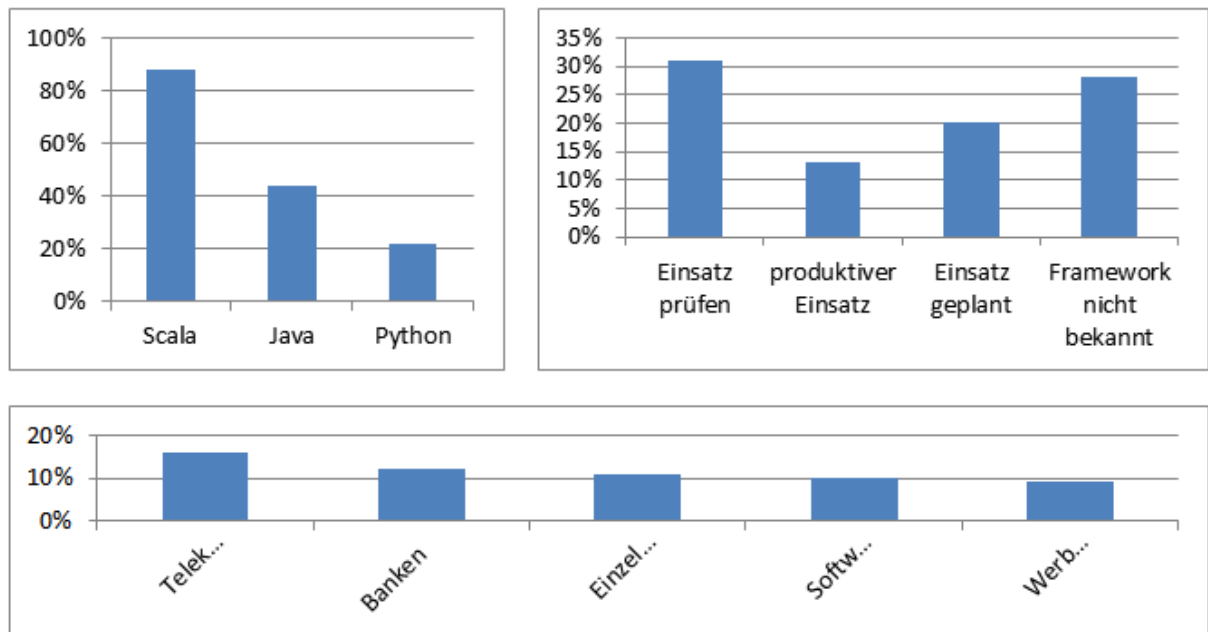


Abbildung 2.10: Einsatz & Verbreitung

## 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 Apache 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. Die Kommunikation innerhalb der Community findet im wesentlichen über offizielle Mailinglisten und einem Ticket-System der Apache Foundation statt. Der Code liegt auf GitHub<sup>1</sup> und ist öffentlich für jeden zugänglich. Bis zum 10.04.2017 gab es bereits 51 Releases oder Release-Kandidaten, 19,365 commits und 1,053 contributors<sup>2</sup>. Auch das wird zukünftig weiter gehen. Im ersten Quartal 2017 gab es über 717 commits. Ein Einbruch der Aktivität ist momentan nicht zu erkennen.<sup>3</sup>

Von der Version 1.6 auf die Version 2.0 gab es nochmal eine relativ starke Performancesteigerung. Vermutlich wird man solche Performancesteigerungen nicht mehr so leicht erreichen, aber dennoch sollten sich an den Geschwindigkeiten auch zukünftig noch etwas nach unten verändern. Eine Übersicht der Performanceänderungen ist in der Tabelle 4.1 zu sehen.<sup>4</sup>

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: Kosten pro Zeile (cost per row) auf einem einzelnen Thread

Zukünftig ist denkbar, dass noch weitere Komponenten so wie zum Beispiel SparkR dazu kommen. Auch das Anbinden weiterer Datenquellen wird sehr wahrscheinlich weiter vorangetrieben werden.

<sup>1</sup>GitHub: Ist ein webbasierter Onlinedienst, der die Möglichkeit bietet Softwareprojekt mit der Versionsverwaltung Git dort zu speichern.

<sup>2</sup>contributors: Sind Beitragende, die zum Projekt mit Schreiben von Code beigetragen haben.

<sup>3</sup>Vgl. [Git17]

<sup>4</sup>Vgl. [Dat17]

## 5 Anhang

```
//create spark context
String master = "local[4]";
String appName = "SeminararbeitTop20Woerter";
JavaSparkContext javaSparkContext = new JavaSparkContext(master, appName);

convertPdfToTextfile(PATH_TO_PDF_FILE, PATH_TO_TXT_FILE);

//read lines and split to words
JavaRDD<String> lines = javaSparkContext.textFile(PATH_TO_TXT_FILE);
JavaRDD<String> words = lines.flatMap(s -> Arrays.asList(SPACE.split(s)).iterator());
JavaRDD<String> filteredWords = words.filter(word -> word.length() != 0);
filteredWords = filteredWords.filter(word -> !word.equals("."));

//count words
JavaPairRDD<String, Integer> ones = filteredWords.mapToPair(s -> new Tuple2<>(s, 1));
JavaPairRDD<String, Integer> counts = ones.reduceByKey((i1, i2) -> i1 + i2);

//swap the tupel
JavaPairRDD<Integer, String> map = counts.mapToPair(item -> item.swap());

//sort the key (word count)
boolean ascending = false;
JavaPairRDD<Integer, String> sortByKey = map.sortByKey(ascending);
List<Tuple2<Integer, String>> output = sortByKey.collect();

int numberOfLines = 20;
printToConsole(output, numberOfLines);

javaSparkContext.close();
```

Abbildung 5.1: Quellecode um Wörter zu zählen und Top 20 Ranking auszugeben

## 6 Literaturverzeichnis

### Bücher

- [Ryz+15] Sandy Ryza u. a. *Advanced Analytics with Spark*. 1005 Gravenstein Highway North: O'Reilly Media, Inc., 2015.
- [ER16] Raul Estrada und Isaac Ruiz. *Big Data SMACK*. New York, 233 Spring Street: Springer Science + Business Media, 2016.

### Papers

- [Zah+12] Matei Zaharia u. a. *Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing*. Forschungsspapier. University of California, Berkeley, 2012.
- [Arm+15a] Michael Armbrust u. a. *Scaling Spark in the Real World: Performance and Usability*. Forschungsspapier. Databricks Inc.; MIT CSAIL, 2015.
- [Arm+15b] Michael Armbrust u. a. *Spark SQL: Relational Data Processing in Spark*. Forschungsspapier. Databricks Inc.; MIT CSAIL; AMPLab, UC Berkeley, Juni 2015.
- [Ven+15] Shivaram Venkataraman u. a. *SparkR: Scaling R Programs with Spark*. Forschungsspapier 11. AMPLab UC Berkeley, Databricks Inc., MIT CSAIL, Nov. 2015.
- [ZAH+15] MATEI ZAHARIA u. a. *Apache Spark: A Unified Engine for Big Data Processing*. Forschungsspapier 11. COMMUNICATIONS OF THE ACM, Nov. 2015.

### Internet

- [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>.
- [Dat17] Inc. Databricks. *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>.
- [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>.
- [Fou17e] Apache Software Foundation. *GraphX - Spark 2.1.0 Documentation*. Apr. 2017. URL: <http://spark.apache.org/docs/latest/graphx-programming-guide.html>.
- [Fou17f] Apache Software Foundation. *ML Pipelines - Spark 2.1.0 Documentation*. Apr. 2017. URL: <http://spark.apache.org/docs/latest/ml-pipeline.html>.
- [Git17] Inc. GitHub. *apache/spark: Mirror of Apache Spark*. Apr. 2017. URL: <https://github.com/apache/spark>.