Review – „Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing“

Seminararbeit zu der Veranstaltung
Skalierbare Datenbanken

Felix Ortmann
{0ortmann}@informatik.uni-hamburg.de

6. Juni 2016

Zusammenfassung

Dieses Review von „Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-memory Cluster Computing“ [1] stellt zunächst die Autoren des Artikels kurz vor und setzt sie in Relation zu dem Thema des Artikels. Es folgt eine knappe inhaltliche Zusammenfassung des Artikels sowie eine kritische Beurteilung und eigene Meinung.

1 Autoren

Der Artikel wurde in Zusammenarbeit von insgesamt neun Wissenschaftlern der Universität von Kalifornien, Berkeley erarbeitet. Auf der neunten *USENIX Conference on Networked Systems Design and Implementation* wurde die Arbeit veröffentlicht. Im Folgenden werden alle Autoren kurz vorgestellt und ihr fachlicher Bezug zum Thema des Artikels knapp umrissen.

Matei Zaharia [1] ist Assistenzprofessor bei CSAIL[2] und Co-Founder der Firma Databricks. Während seiner Doktorarbeit an der UC Berkeley war er unteranderem maßgeblich an der Entwicklung von *Spark* beteiligt. Er wurde laut Google Scholar insgesamt ca. 19880 mal zitiert.

Mosharaf Chowdhury [3] ist Assistenzprofessor bei der Universität Michigan, Department Computer Science and Engineering.[4]. Auch er schrieb seine Doktorarbeit an der UC Berkeley. Momentan forscht er im Bereich Big Data und Cloud Computing. Laut Google Scholar wurde Chowdhury ca. 5720 mal zitiert.

[1] https://people.csail.mit.edu/matei/
[2] MIT: Computer Science and Artificial Intelligence Laboratory
[3] http://www.mosharaf.com/
[4] http://cse.umich.edu/

Tathagata Das ist zur Zeit bei Databricks angestellt, wo er am Spark Projekt und *Spark Streaming* arbeitet. Er hat seinen Master Abschluss an der UC Berkeley gemacht und war an der Entstehung von Spark beteiligt[5]. Insgesamt wurde er 2336 mal zitiert (Google Scholar).

Ankur Dave [6] befindet sich momentan im dritten Jahr seines Doktoriats an der UC Berkeley. Er war drei Jahre lang wissenschaftlicher Mitarbeiter dort und während dieser Zeit arbeitete er mit an dem Spark Projekt. Mittlerweile hat er bei fast allen namhaften Bigdata Firmen Praktika absolviert. Laut Google Scholar wurde er 1361 mal zitiert.

Justin Ma [7] ist seit 2012 Angestellter bei Google. Davor war er graduierter wissenschaftlicher Mitarbeiter an der UC Berkeley. Mittlerweile liegt sein Schwerpunkt mehr auf Netzwerksicherheit als auf Bigdata.[8] Laut SemanticScholar wurde er ca. 3470 mal zitiert.[9]

Murphy McCauley war Master Student an der UC Berkley, wo sein Fokus auf Software Networking lag.[10]

Michael J. Franklin [11] ist Professor for Computer Science und Bigdata an der UC Berkley. Seine drei Forschungsschwerpunkte sind Datenbanken, Betriebssysteme und Netzwerke. Er wurde laut Google Scholar ca. 4701 mal zitiert.

Scott Shenker [12] ist Professor an der UC Berkeley und eine Koryphäe in der Bigdata-Szene. Er hält viele Ehrenämter an der UC Berkeley. Laut Google Scholar wurde er ca. 102547 mal zitiert und ist einer der fünf meist zitiertesten Wissenschaftler weltweit. Er war unter anderem Doktorvater von Matei Zaharia und vielen anderen.

Ion Stoica [13] ist Professor für Computer Science an der UC Berkeley. Er war maßgeblich beteiligt an der Entstehung von Spark und war Doktorvater für Matei Zaharia und andere der Autoren. Momentan arbeitet auch er bei Databricks. Laut ResearchGate[14] wurde er ca. 30231 mal zitiert.

2 Inhaltliche Zusammenfassung

Die Autoren beschreiben ein generelles Vorgehen, um effizient mit verteilten in-memory-Datasets zu arbeiten, welche RDDs genannt werden. RDD steht für „Resilient Distributed Dataset“. Der Artikel beginnt mit einer anschaulichen Motivation – der Wiederverwendbarkeit von Rechen-Zwischenergebnissen. Es wird deutlich gemacht, dass viele der gän-

[5] https://www.linkedin.com/in/tathadas
[6] http://ankurdave.com/
[7] https://amplab.cs.berkeley.edu/author/jma/
[8] https://www.linkedin.com/in/justin-ma-73897b2
[9] https://www.semanticscholar.org/author/Justin-Ma/2932392
[10] https://www.usenix.org/system/files/login/articles/zaharia.pdf
[11] https://people.eecs.berkeley.edu/ franklin/
[12] https://www.eecs.berkeley.edu/Faculty/Homepages/shenker.html
[13] http://people.eecs.berkeley.edu/ istoica/
[14] https://www.researchgate.net/profile/Ion_Stoica/citations

gigen Batchprocessing- oder Clustercomputing-Frameworks ineffizient mit intermediären Daten verfahren.

Der Grundidee von RDDs ist, dass sie im nicht-persistenten Speicher eines Clusters vorliegen. Dadurch werden Zugriffe und Prozessierungen stark beschleunigt, denn es müssen (meistens) keine I/O-Zugriffe stattfinden.

Rechenergebnisse werden durch die wiederholbare Aneinanderkettung von Rechenschritten definiert. Dies wird im Artikel mit „Lineage“ – „Abstammungslinie“ betitelt. Dabei sind die Rechenschritte stets Transformationen, die auf eine Menge von Objekten angewandt werden, so wie *map, filter, join, etc.* RDDs sind derart partitioniert, dass sie geschickt über das Cluster verteilt werden können. Die Partitionierung kann auch vom User definiert werden. Anhand der Lineage ist es möglich eine gewisse Fehlertoleranz abzubilden. Sollten RDD-Partitionen durch Systemversagen verloren gehen, können sie durch erneute Anwendung der Lineage schnell und nebenläufig rekonstruiert werden. Die Anwendbarkeit von RDDs in der Praxis wurde mit der Implementation von *Spark* demonstriert.

2.1 Spark

Die Autoren präsentieren Spark, ein Framework das RDDs implementiert und somit Daten über viele parallele Operationen hinweg schnell zugreifbar macht. Spark ist besonders für iterative Aufgaben geeignet, wie interaktives Data-Mining. Das Framework garantiert Fehlertoleranz und Skalierbarkeit.

Die Autoren geben an, iteratives Machine-Learning sowie auch Google Pregel unter Verwendung von RDDs implementiert zu haben. Ferner behaupten sie, dass das Spark-Framework bis zu 20-mal schneller sein kann als Hadoop und dass die Datenanalyse mit Spark bis zu 40-mal schneller ist.

Diese Behauptungen werden durch Benchmarks gägngiger Algorithmen belegt. Es wurden einige Algorithmen wie *k-means*, *logische Regression* und *PageRank* via Spark implementiert, was nach eigenen Angaben mit bemerkenswert wenigen Zeilen Code geschehen ist. Anschließend wurden deren Laufzeiten in Kontrast gesetzt zu den herkömmlichen Laufzeiten auf Hadoop sowie auch auf HadoopBinMem, welches shared-memory Policies in Hadoop ermöglicht. Mit Spark ließ sich etwa bei PageRank eine Verdoppelung der Geschwindigkeit feststellen, bei iterativen Machinelearning Verfahren sogar eine bis zu 20-fache Beschleunigung.

Der Artikel weist darauf hin, dass viele der gängigen Hadoop-Anwendungsfälle sowie auch die Funktionen anderer Batchprocessing-Frameworks mit Spark und RDDs abgebildet werden können. Die Autoren beschreiben verschiedenste Anwendungsszenarien wie etwa Verkehrsmodellierung, Spam Klassifikation und interactive Data-Mining. Auch hierfür werden Implementationshinweise für Spark geliefert.

Bei allen Tests haben die Spark Implementationen bedeutend besser abgeschnitten als die Referenzimplementationen auf Hadoop / HadoopBinMem. Die Behauptungen der Autoren was Geschwindigkeitsverbessungen durch Spark angeht wurden bewiesen.

Für den Fall von wenig verfügbarem Hauptspeicher beschreiben die Autoren das Verhalten von *logischer Regression* unter Spark. Die Performance des Algorithmus nimmt linear mit dem verfügbaren Speicher ab.

3 Beurteilung

Der Artikel ist sehr zielorientiert gehalten und präsentiert RDDs und Spark als herausragende Idee und Technologie. Durch die ausfühliche Motivation wird dem Leser der Bedarf nach einem Framework wie Spark nahe gebracht und gut verständlich gemacht. Im Verlauf zeigt der Artikel zunächst immer wieder Fragestellungen oder Anwendungsfälle auf und motiviert den Leser für eine Verbesserung bestehender Verfahren. Anschließend wird eine (Geschwindigkeits-)Verbesserung für das zuvor beschriebene Verfahren präsentiert, die durch RDDs und Spark ermöglicht wird.

Alle Tests und Benchmarks sind unter idealen Bedingungen gemacht worden. Die Autoren diskutieren zwar kurz das Verhalten für *logische Regression* unter Spark bei wenig verfügbar RAM, aber vergleichen dieses Verhalten nicht mit anderen Frameworks oder Hadoop-basierten Implementationen.

Die Autoren erwähnen unter anderem Pregel, HaLoop und Twister als verwandte Arbeiten, vergleichen deren Performance aber nicht mit Spark. Stattdessen geben sie an, dass es möglich ist die Funktionen dieser Frameworks auch mit Spark umsetzen zu können – ohne durch Benchmarks zu beweisen, dass dies eine klare Verbesserung mit sich bringt. Hier sollten meiner Meinung nach Benchmarks zwischen den origanlen Implementationen und deren Spark-basierten Versionen angeführt werden. Durch die Menge der Tests wird ein wenig verschleiert, dass kein Test im Detail erklärt wird.

Allgemein ist das Paper sehr aussagekräftig und verdeutlicht die Überlegenheit von Spark (in bestimmten Anwendungsfällen) gegenüber anderen Verfahren und Frameworks. Die Autoren stellen gut heraus, warum es einer derartigen Lösung bedarf und vermuten begründet, warum es bisher noch keine gab.

Literatur

[1] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pages 2–2, Berkeley, CA, USA, 2012. USENIX Association.