Die Apache Flink Plattform zur parallelen Analyse von Datenströmen und Stapeldaten

Jonas Traub*, Tilmann Rabl*, Fabian Hueske†,
Till Rohrmann† und Volker Markl*§

*Technische Universität Berlin, FG DIMA, Einsteinufer 17, 10587 Berlin
†data Artisans GmbH, Tempelhofer Ufer 17, 10963 Berlin
§DFKI GmbH, Intelligente Analytik für Massendaten, Alt-Moabit 91c, 10559 Berlin
{jonas.traub,rabl,volker.markl}@tu-berlin.de,
{fabian,till}@data-artisans.com http://www.dima.tu-berlin.de

Abstract. Die Menge an analysierbaren Daten steigt aufgrund fallender Preise für Speicherlösungen und der Erschließung neuer Datenquellen rasant. Da klassische Datenbanksysteme nicht ausreichend parallelisierbar sind, können sie die heute anfallenden Datenmengen häufig nicht mehr verarbeiten. Hierdurch ist es notwendig spezielle Programme zur parallelen Datenanalyse zu verwenden. Die Entwicklung solcher Programme für Computercluster ist selbst für erfahrene Systemprogrammierer eine komplexe Herausforderung. Frameworks wie Apache Hadoop MapReduce sind zwar skalierbar, aber im Vergleich zu SQL schwer zu programmieren.
Die Open-Source Plattform Apache Flink schließt die Lücke zwischen herkömmlichen Datenbanksystemen und Big-Data Analyseframeworks. Das Top Level Projekt der Apache Software Foundation basiert auf einer fehlertoleranten Laufzeitumgebung zur Datenstromverarbeitung, welche die Datenverteilung und Kommunikation im Cluster übernimmt. Verschiedene Schnittstellen erlauben die Implementierung von Datenanalyseabläufen für unterschiedlichste Anwendungsfälle. Die Plattform wird von einer aktiven Community kontinuierlich weiter entwickelt. Sie ist gleichzeitig Produkt und Basis vieler Forschungsarbeiten im Bereich Datenbanken und Informationsmanagement.

Stichwörter: Big-Data, Datenstromverarbeitung, Stapelverarbeitung, Datenanalyse, Datenbanken, Datenanalyseabläufe

1 Einleitung

Dank der stark fallenden Kosten für die Datenspeicherung, Cloud-Speicherangeboten und der intensivierten Nutzung des Internets, steigt die Menge an verfügbaren Daten sehr schnell an. Während der theoretische Wert der Analyse dieser Daten unbestritten ist, stellt die tatsächliche Datenauswertung eine

 In: R. Bergmann, S. Görg, G. Müller (Eds.): Proceedings of the LWA 2015 Workshops: KDML, FGWM, IR, and FGDB. Trier, Germany, 7.-9. October 2015, published at http://ceur-ws.org

große Herausforderung dar. Konventionelle Datenbanksysteme sind nicht länger in der Lage mit den enormen Datenmengen und der dynamischen oder fehlenden Struktur der Daten umzugehen.

Das Forschungsprojekt Stratosphere[1] verfolgt das Ziel die Big-Data Analyseplattform der nächsten Generation zu entwickeln und damit die Analyse sehr großer Datenmengen handhabbar zu machen. Im Jahr 2014 wurde das im Stratosphere Projekt entwickelte System unter dem Namen Flink[1] zunächst ein Apache Incubator Projekt und später ein Apache Top Level Projekt.

Im Vergleich zu anderen verteilten Datenanalyseplattformen, reduziert Flink die Komplexität für Anwender durch die Integration von traditionellen Datenbanksystemkonzepten, wie deklarativen Abfragesprachen und automatischer Abfrageoptimierung. Gleichzeitig erlaubt Flink *schema-on-read*[2], ermöglicht die Verwendung von benutzerdefinierten Funktionen und ist kompatibel mit dem Apache Hadoop MapReduce Framework[3]. Die Plattform hat eine sehr gute Skalierbarkeit und wurde auf Clustern mit hunderten Maschinen, in Amazons EC2 und auf Googles Compute Engine erprobt.

Im Folgenden stellen wir in Abschnitt 2 die Architektur der Flink Plattform näher vor und zeigen Bibliotheken, Schnittstellen und ein Programmbeispiel in Abschnitt 3. Abschnitt 4 beschreibt Besonderheiten in der Datenstromanalyse von Apache Flink im Vergleich zu anderen Plattformen. Abschließend stellen wir in Abschnitt 5 weiterführende Publikationen vor.

2 Architektur

Abbildung 1 zeigt eine Übersicht der Architektur der Apache Flink Plattform. Die Basis von Flink ist eine einheitliche Laufzeitumgebung in der alle Programme ausgeführt werden. Programme in Flink sind strukturiert als gerichtete Graphen aus parallelisierbaren Operatoren, welche auch Iterationen beinhalten können [6]. Bei der Ausführung eines Programms in Flink werden Operatoren zu mehreren parallelen Instanzen segmentiert, welche jeweils einen Teil der Datentupel verarbeiten (Datenparallelität). Im Gegensatz zu Hadoop MapReduce, werden Programme in Flink nicht in nacheinander auszuführende Phasen (Map und Reduce) geteilt. Alle Operatoren werden nebenläufig ausgeführt, sodass die Ergebnisse eines Operators direkt zu folgenden Operatoren weitergeleitet und dort verarbeitet werden können (Pipelineparallelität). Neben der verteilten Laufzeitumgebung für Cluster, stellt Flink auch eine lokale Laufzeitumgebung bereit. Diese ermöglicht es Programme direkt in der Entwicklungsumgebung auszuführen und zu debuggen. Flink ist kompatibel mit einer Vielzahl von Clustermanagement-

[1] `http://flink.apache.org`

[2] Bei *schema-on-read* werden Daten in ihrer ursprünglichen Form gespeichert, ohne ein Datenbankschema festzulegen. Erst beim Lesen der Daten werden diese in ein abfragespezifisches Schema überführt, was eine große Flexibilität bedeutet.

[3] `http://hadoop.apache.org`

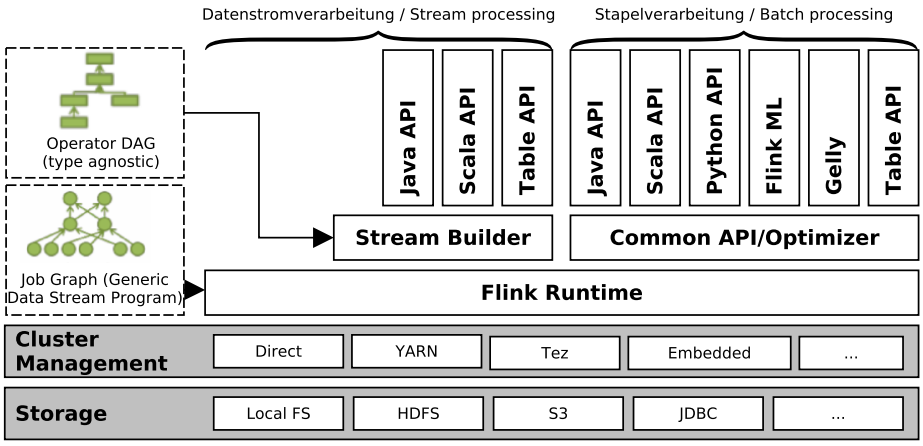

Abb. 1. Architektur- und Komponentenübersicht der Apache Flink Plattform.

und Speicherlösungen, wie Apache Tez[4], Apache Kafka[5] [9], Apache HDFS[3] [10] und Apache Hadoop YARN[3] [12].

Zwischen der Laufzeitumgebung und den Programmierschnittstellen (API), sorgen *Stream Builder* und *Common API* für die Übersetzung von gerichteten Graphen aus logischen Operatoren in generische Datenstromprogramme, welche in der Laufzeitumgebung ausgeführt werden. In diesem Schritt erfolgt auch die automatische Optimierung des Datenflussprogramms. Während der Anwender beispielsweise lediglich einen Join spezifiziert, wählt der integrierte Optimierer den für den jeweiligen Anwendungsfall besten konkreten Join-Algorithmus aus.

Der folgende Abschnitt gibt eine Übersicht über die oberste Schicht der Flink Architektur, welche aus einem breiten Spektrum von Bibliotheken und Programmierschnittstellen besteht.

3 Bibliotheken und Schnittstellen

Nutzer von Apache Flink können Abfragen in verschiedenen Programmiersprachen spezifizieren. Zur Analyse von Datenströmen und zur Stapelverarbeitung stehen jeweils eine Scala und eine Java API zur Verfügung. Stapeldaten können außerdem mit einer Python API verarbeitet werden. Alle APIs stellen dem Programmierer generischen Operatoren wie Join, Cross, Map, Reduce und Filter zur Verfügung. Dies steht im Gegensatz zu Hadoop Map Reduce wo komplexe Operatoren als Folge von Map- und Reducephasen implementiert werden müssen. Listing 1 zeigt eine Wordcount-Implementierung in der Scala Stream Processing API. Eine Implementierung zur Stapelverarbeitung ist analog zu diesem Beispiel unter Auslassung der Window-Spezifikation möglich.

[4] `http://tez.apache.org`
[5] `http://kafka.apache.org`

```
1 case class Word (word: String, frequency: Int)
2 val lines: DataStream[String] = env.fromSocketStream(...)
3 lines.flatMap{line => line.split(" ")}
4     .map(word => Word(word,1))}
5     .window(Time.of(5,SECONDS)).every(Time.of(1,SECONDS))
6     .groupBy("word").sum("frequency").print()
```

Listing 1. Eine Wordcount-Implementierung unter Verwendung der Scala Stream Processing API von Apache Flink.

In der ersten Zeile wird ein Tupel bestehend aus einem String und einer Ganzzahl definiert. Programmzeile 2 gibt einen Socketstream an von dem ein Textdatenstrom zeilenweise eingelesen wird. In Zeile 3 wird ein FlatMap-Operator angewendet, welcher Zeilen als Eingabe erhält, diese an Leerzeichen trennt und die resultierenden Einzelwörter in das zuvor definierte Tupelformat mit dem Wort als String und 1 als Zahlenwert konvertiert. Da es sich um eine Datenstromabfrage handelt, wird ein Fenster spezifiziert, hier ein gleitendes Fenster mit einer Länge von fünf Sekunden und einer Schrittweite von einer Sekunde. Abschließend erfolgt eine Gruppierung nach Wörtern und die Zahlenwerte werden innerhalb der Gruppen aufsummiert. Die Printmethode sorgt für die Ergebnisausgabe auf der Konsole.

Zusätzlich zu den klassischen Programmierschnittstellen, bietet die *Flink ML* Bibliothek eine Vielzahl an Algorithmen des maschinellen Lernens. *Gelly* ermöglicht die Graphenanalyse mit Flink. Die *Table API* bietet die Möglichkeit der deklarativen Spezifikation von Abfragen ähnlich zu *SQL* und steht als Java- und Scalaversion zur Verfügung. Listing 2 zeigt eine Wordcount-Implementierung mit der Java Table API zur Stapelverarbeitung.

```
1 DataSet<WC> input = env.fromElements(new WC("Hello",1),new WC("Bye",1),new WC("Hello",1));
2 Table table=tableEnv.fromDataSet(input).groupBy("word").select("word.count as count, word");
3 tableEnv.toDataSet(table, WC.class).print();
```

Listing 2. Eine Wordcount-Implementierung unter Verwendung der Java Table API zur Stapelverarbeitung von Apache Flink.

In Zeile 1 werden Eingabewörter explizit angegeben. Zeile 2 konvertiert das *DataSet* zunächst zu einer Tabelle, die anhand des Attributs *word* gruppiert wird. Die Selectanweisung wählt wie in SQL das Wort sowie die Summe der Zähler aus. Abschließen wird die Ergebnistabelle zurück zu einem *DataSet* konvertiert und ausgegeben.

4 Datenstromverarbeitung

Die Datenstromverarbeitung unterscheidet sich signifikant von der Stapelverarbeitung: Programme haben lange (theoretisch unendliche) Laufzeiten, konsumieren Daten kontinuierlich von Eingabeströmen und produzieren im Gegenzug Ausgabeströme. Aggregationen können jedoch nur für abgeschlossene Datenblöcke berechnet werden. Sie folgen in Datenstromprogrammen daher auf eine Diskretisierung, die einen Datenstrom in abgeschlossene, potentiell überlappende Fenster unterteilt. Eine Aggregation erfolgt dann fortlaufend per Fenster.

Im Gegensatz zu vielen anderen Datenanalyseplattformen ist Flink durch seine Laufzeitumgebung nicht an Limitationen gebunden, die aus Micro-Batching Techniken [14] entstehen. Beim Micro-Batching wird ein Datenstrom als Serie von Datenblöcken fester Größe interpretiert, die separat als Stapel verarbeitet werden. Die Größen aller Fenster müssen Vielfache der Blockgröße sein, sodass ein Gesamtergebnis aus den Blockergebnissen berechnet werden kann. Flink stellt weitaus flexiblere Diskretisierungsoptionen bereit, welche eine Generalisierung von IBM SPLs Trigger und Eviction Policies [7] sind. Eine Trigger Policy gibt an, wann ein Fenster endet und die Aggregation für dieses Fenster ausgeführt wird. Die Eviction Policy gibt an, wann Tupel aus dem Fernsterpuffer entfernt werden und spezifiziert so die größe von Fenstern. Anwender können aus einer Vielzahl von vordefinierten Policies wählen (z.B. basierend auf Zeit, Zählern oder Deltafunktionen) oder benutzerdefinierte Policies implementieren. Flink erreicht damit bei geringeren Latenzen eine größere Expressivität als micro-batch-abhängige Systeme und vermeidet die Komplexität von Lambda-Architekturen.

Operatoren in Flink können statusbehaftet sein. Ein Schnappschussalgorithmus stellt sicher, dass jedes Tupel auch im Fehlerfall exakt einmal im Operatorstatus repräsentiert ist und verarbeitet wird.

Flink bietet somit eine bei Open-Source-Systemen einmalige Kombination aus Stapelverarbeitung, nativer Datenstromverarbeitung ohne Beschränkungen durch Micro-Batching, statusbehafteten Operatoren, ausdrucksstarken APIs und *exactly-once* Garantien.

5 Weiterführende Publikationen

Flink ist sowohl Produkt als auch Basis vieler Forschungsarbeiten. Im Folgenden werden die wichtigsten Publikationen genannt. Warnecke et al. stellen die Nephele Laufzeitumgebung vor [13], auf der Flinks Laufzeit ursprünglich basierte. Battré et al. ergänzen sie mit dem PACT Modell [3], einer Erweiterung von MapReduce [4]. Alexandrov et al. geben eine detaillierte Beschreibung der Stratosphere Plattform [1]. Hueske et al. befassen sich mit der Optimierung von Programmen mit benutzerdefinierten Funktionen [8]. Ewen et al. führen die native Unterstützung von Iterationen ein [6]. Aktuelle Arbeiten befassen sich mit Fehlertoleranz [5] und implizitem Parallelismus mittels eingebetteter Sprachen [2]. Spangenberg et al. vergleichen die Performance von Flink und Spark für unterschiedliche Algorithmen [11].

6 Resumé

Flink vereinfacht die parallele Analyse großer Datenmengen durch die Anwendung klassischer Datenbanktechniken wie automatischer Optimierung und deklarativen Abfragesprachen. Ausdrucksstarke, intuitive APIs ermöglichen sowohl Stapel- als auch Datenstromverarbeitung. Flink ist skalierbar und durch seine große Kompatibilität vielseitig einsetzbar. Operatoren werden nebenläufig, frei von Limitierungen durch Micro-Batching-Techniken, in einer Pipeline ausgeführt.

7 Danksagung

Für die Entwicklung der Plattform gilt unser besonderer Dank der gesamten Flink Community. Dieses Forschungsprojekt wird unterstützt durch Mittel des BMBF für das Berlin Big Data Center (BBDC) unter der Förderungsnummer 01IS14013 sowie der DFG Forschergruppe Stratosphere (FOR 1306).

Referenzen

1. Alexandrov, A., Bergmann, R., Ewen, S., Freytag, J. C., Hueske, F., Heise, A., ... & Warneke, D. (2014). The Stratosphere platform for big data analytics. The VLDB Journal—The International Journal on Very Large Data Bases, 23(6), 939-964.
2. Alexandrov, A., Kunft, A., Katsifodimos, A., Schüler, F., Thamsen, L., Kao, O., ... & Markl, V. (2015, May). Implicit Parallelism through Deep Language Embedding. In Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (pp. 47-61). ACM.
3. Battré, D., Ewen, S., Hueske, F., Kao, O., Markl, V., & Warneke, D. (2010, June). Nephele/PACTs: a programming model and execution framework for web-scale analytical processing. In Proceedings of the 1st ACM symposium on Cloud computing (pp. 119-130). ACM.
4. Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. Communications of the ACM, 51(1), 107-113.
5. Dudoladov, S., Xu, C., Schelter, S., Katsifodimos, A., Ewen, S., Tzoumas, K., & Markl, V. (2015, May). Optimistic Recovery for Iterative Dataflows in Action. In Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (pp. 1439-1443). ACM.
6. Ewen, S., Tzoumas, K., Kaufmann, M., & Markl, V. (2012). Spinning fast iterative data flows. Proceedings of the VLDB Endowment, 5(11), 1268-1279.
7. Gedik, B. (2014). Generic windowing support for extensible stream processing systems. Software: Practice and Experience, 44(9), 1105-1128.
8. Hueske, F., Peters, M., Sax, M. J., Rheinländer, A., Bergmann, R., Krettek, A., & Tzoumas, K. (2012). Opening the black boxes in data flow optimization. Proceedings of the VLDB Endowment, 5(11), 1256-1267.
9. Kreps, J., Narkhede, N., & Rao, J. (2011, June). Kafka: A distributed messaging system for log processing. In Proceedings of the NetDB (pp. 1-7).
10. Shvachko, K., Kuang, H., Radia, S., & Chansler, R. (2010, May). The hadoop distributed file system. In Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on (pp. 1-10). IEEE.
11. Spangenberg, N., Roth, M., & Franczyk, B. (2015, June). Evaluating New Approaches of Big Data Analytics Frameworks. In Business Information Systems (pp. 28-37). Springer International Publishing.
12. Vavilapalli, V. K., Murthy, A. C., Douglas, C., Agarwal, S., Konar, M., Evans, R., ... & Baldeschwieler, E. (2013, October). Apache hadoop yarn: Yet another resource negotiator. In Proceedings of the 4th annual Symposium on Cloud Computing (p. 5). ACM.
13. Warneke, D., & Kao, O. (2009, November). Nephele: efficient parallel data processing in the cloud. In Proceedings of the 2nd workshop on many-task computing on grids and supercomputers (p. 8). ACM.
14. Zaharia, M., Das, T., Li, H., Shenker, S., & Stoica, I. (2012, June). Discretized streams: an efficient and fault-tolerant model for stream processing on large clusters. In Proceedings of the 4th USENIX conference on Hot Topics in Cloud Ccomputing (pp. 10-10). USENIX Association.