Hochschule für Technik, Wirtschaft und Kultur Leipzig

Abstrakt

Thema: Apache Flink

vorgelegt von: Martin Schallnahs

E-Mail: martin.schallnahs@stud.htwk-leipzig.de

Matrikelnummer: 70011

Studiengang: Informatik, 17INM/TZ

Lehrveranstaltung: Oberseminar
"Datenbanksysteme - Aktuelle Trends"

eingereicht am: 02.06.2019

in: Leipzig

Prüfer: Prof. Dr.-Ing. Thomas Kudraß

Inhaltsverzeichnis

1 Einleitung

Apache Flink ist ein *Stream Processing Framework* und damit ein Werkzeug zur Verarbeitung von *Big Data*. Es wird unter der Schirmherrschaft der *Apache Software Foundation* als Open-Source-Projekt entwickelt.

Die Motivation zur Entwicklung von Flink war und ist die Schaffung eines Werkzeuges, das die nachfolgenden Eigenschaften hat. Es soll für viele Anwendungsfälle einsetzbar sein (Fabian Hueske 2016, 13 ff.). Für die Verarbeitung von endlichen, aber auch unendlichen Datenströmen, gleichzeitig aber auch zur Ausführung von *Graphen*- und *Machine-Learning*-Algorithmen (Slim Baltagi 2016, 6 ff.). Es soll eine *Real-Time-Verarbeitung* ermöglichen. Dazu gehört die native Unterstützung von *Processing*- und *Event-Time* als Zeitstempel, sowie die Unterstützung verschiedener Mechaniken um Teilmengen in Datenströmen zu bilden – wie *Windows* oder *Sessions*. Es soll eine geringe Latenz im Millisekundenbereich und ein hoher Datendurchsatz gewährleistet werden. Gleichzeitig sollen die Ergebnisse genau sein, so dass eine *Exactly-Once-Semantik* eingehalten wird. Zusätzlich sollen die Entwicklung, Konfiguration und Betriebsoptimierung sehr einfach und gering im Aufwand sein und Flink soll rückwärts-kompatibel zu den älteren Werkzeugen *Apache Hadoop*/*MapReduce* und Apache *Storm* sein.

Typische Anwendungsfälle, die diese Eigenschaften erfordern stammen aus verschiedensten Bereichen. Gemein haben die meisten Anwendungsfälle, dass es darum geht mit geringer Latenz und geringem Aufwand einen unendlichen Datenstrom zu verarbeiten. Die Bank *ING* betreibt ihre *Missbrauchserkennung* (engl. *Fraud Detection*) mit Flink (Erik de Nooij 2017). *Zalando* nutzt Flink mehrfach, sowohl zum *Business Process Monitoring* (Mihail Vieru und Hung Chang 2017) als auch zum *Continuous ETL* (Mihail Vieru und Javier Lopez 2016).

2 Geschichte

Das erste erfolgreiche Werkzeug im Big-Data-Umfeld ist Hadoop und es wurde etwa 2006 populär. Die meisten weiteren Big-Data-Werkzeuge siedeln sich im Ökosystem von Hadoop an. Erst 2011 kam mit *Apache Spark* ein innovativer Generationssprung, es wird oft auch als die 3. Generation von Big-Data-Werkzeugen bezeichnet (raincent 2016). Etwa 2015 setzte sich dann die 4. Generation durch: *Apache Spark* mit Pseudo-Streaming-Unterstützung und Flink mit nativer Streaming-Unterstützung (Diego Calvo 2017; Jesús Domínguez 2018).

Die Entwicklung von Apache Flink hat etwa 2010 als Projekt *Stratosphere* an der *TU Berlin* gestartet (Aljoscha Krettek 2017, 39). Bis 2014 folgten wissenschaftliche Abhandlungen zum Projekt und dessen Grundprinzipien. Dann 2014 folgte die Umbenennung in Flink und die Ernennung zum *Apache Top Level Projekt*. Im Jahr 2016 gab es das erste 1.0 Release. Das derzeit aktuelle Release 1.8.

3 Technische Grundlagen

In diesem Abschnitt werden technische Grundlagen von Flink erläutert.

3.1 Grundbegriffe der Stream-Verarbeitung

Eine einheitliche Definition für den Begriff Stream gibt es nicht. Oft wird die namensgebende Parallele zu einem Strom oder Fluss an Daten gezogen. Dieser Strom an Daten kann unendlich fließen, dann wird von *Unbounded Data* gesprochen (Tyler Akidau 2015). Handelt es sich dagegen nur um eine endliche Menge an Daten wird von *Bounded Data* gesprochen. Wie in Abbildung 1 zu sehen, kann Bounded Data konzeptionell als Untermenge von Unbounded Data betrachtet werden.

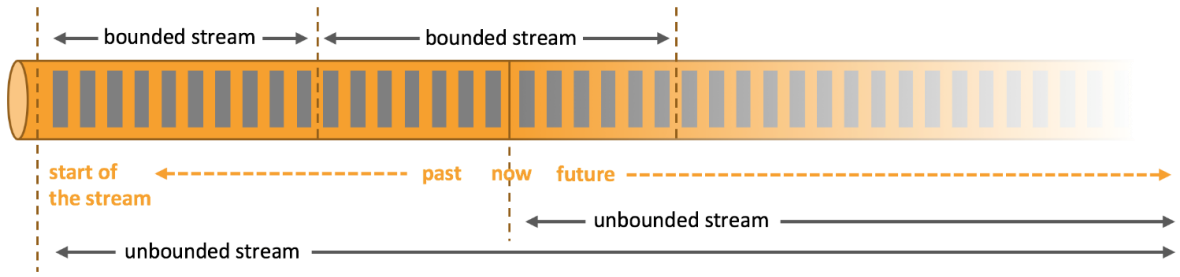

Abbildung 1: Ein Datenstrom unterteilt in Unbounded Data/Stream und Bounded Data/Stream.
(Quelle: Apache Flink 2019)

Es gibt verschiedene Zeitstempel, die das Auftreten eines Events terminieren. Es gibt die Event Time die den Zeitpunkt beschreibt, an dem das Event aufgetreten ist (Aljoscha Krettek 2016, 21). Und die Processing Time, dass den Zeitpunkt beschreibt, an dem das Event im System verarbeitet wird. Bei realen Business-Anwendungen sind in der Regel die Event-Time relevant, denn Daten werden oft in einer „falscher“ Reihenfolge im System verarbeitet. Wird dann bei der Datenanalyse nicht die Event-Time betrachtet sind die Ergebnisse möglicherweise falsch.

3.2 Systemarchitektur

Apache Flink hat schematisch die in Abbildung 2 gezeigte Systemarchitektur: Anwendungen oder Geräte generieren einen unendlichen Datenstrom oder endliche Datenmengen in Form von Real-Time-Events, Datenbanken oder Dateien, die auf einem Filesystem gespeichert sind. Diese Daten werden in Flink verarbeitet. Dazu können, wenn notwendig, *Zustände* zwischengespeichert werden. Die Ausgabe kann dann genauso wie die Eingabe erfolgen. Die Hardware-Ressourcen können durch *YARN* oder *MESOS* verwaltet werden. Typischer Speicher sind das *Hadoop Filesystem* (*HDFS*) oder *Amazons Storage Service S3*.

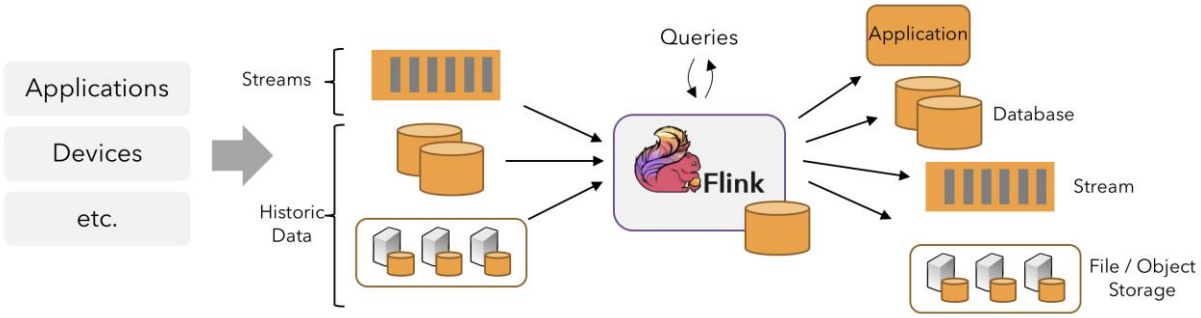

Abbildung 2: Schematische Systemarchitektur von Apache Flink
(Quelle: Ververica 2019b)

Typische Datenquellen und Datensenken sind Dateien aus dem Dateisystem, Nachrichten aus einer *Message Queue*, oder von *Apache Kafka*. Für weitere Beispiele siehe Abbildung 3.

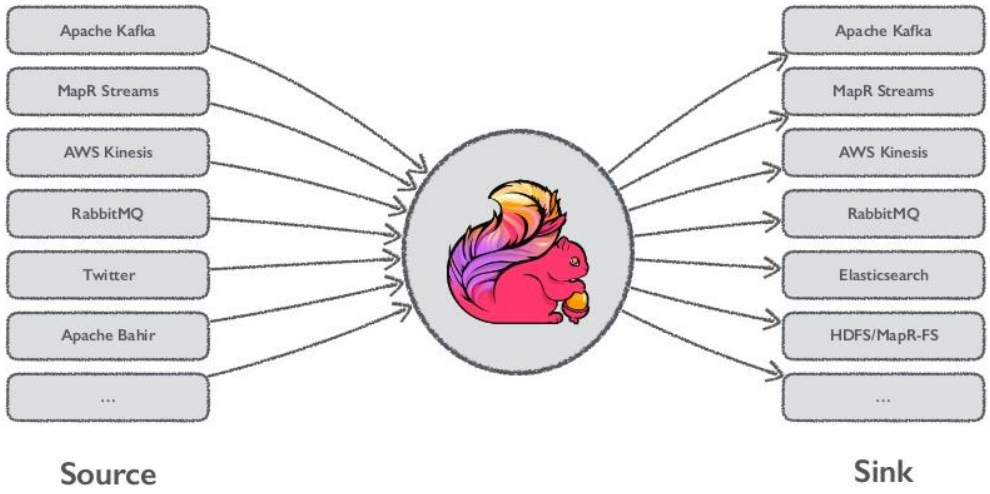

Abbildung 3: Typische Datenquellen und Datensenken
(Quelle: Tugdual Grall 2017)

3.3 Programmiermodel

In einem Programmiermodell, siehe Abbildung 4, bestehen Flink-Verarbeitungsabläufe aus einer oder mehrerer Datenquellen. Die über die Quellen erhaltenen Daten werden in einem oder mehreren Schritten transformiert. Diese Transformationen können zustandsbehaftet sein und damit einen Zwischenspeicher benötigen. Die Ergebnisse können über eine oder mehrere Datensenken ausgegeben werden. Die Verarbeitung verschiedener Teilschritte erfolgt parallel und verteilt.

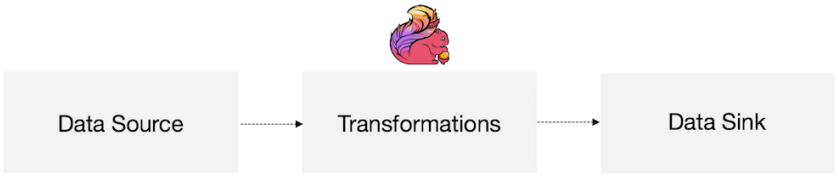

Abbildung 4: Programmiermodell

(Quelle: Ververica 2019b)

In Abbildung 5 ein Beispiel zum Zählen von Events je ID á 10 Sekunden *Windows* mit dessen Java-Quelltext (Apache Flink 2019e). Ein Eingangs-Datenstrom wird eingebunden, hier ein Kafka *Topic*. Als erste Transformation erfolgt ein *map()* zur Selektion von notwendigen Attributen aus den Eingangsdaten. Dann werden anhand des Attributes *id* Gruppen gebildet sowie *Fixed Windows* mit einer Länge von 10 Sekunden. Als Senke wird eine fortlaufend beschriebene Datei festgelegt.

```
DataStream<String> lines = env.addSource(
                                new FlinkKafkaConsumer<>(…));

DataStream<Event> events = lines.map((line) -> parse(line));

DataStream<Statistics> stats = events
        .keyBy("id")
        .timeWindow(Time.seconds(10))
        .apply(new MyWindowAggregationFunction());

stats.addSink(new RollingSink(path));
```

Source

Transformation

Transformation

Sink

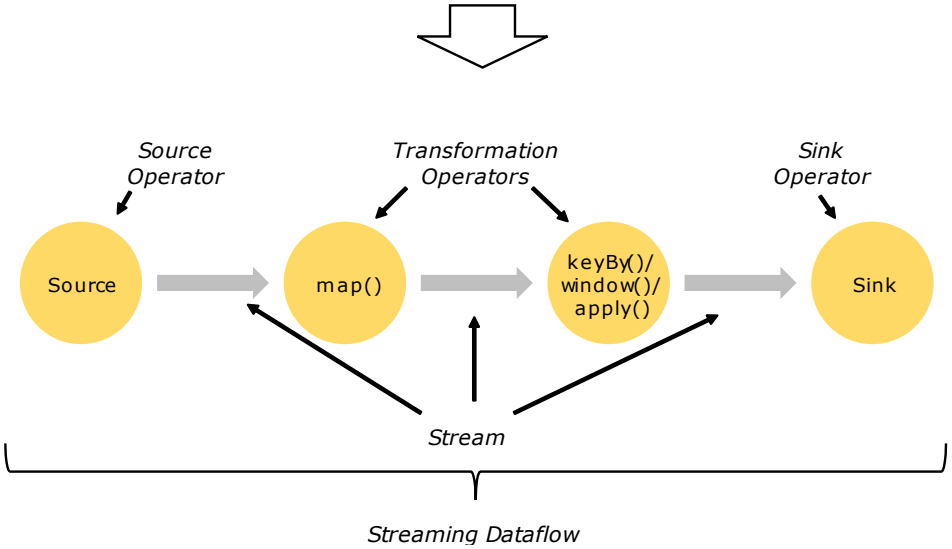

Abbildung 5: Beispiel zum Programmiermodell

(Quelle: Apache Flink 2019e)

3.4 Verteilte Laufzeitumgebung

In Flink werden standardmäßig alle Programme soweit wie möglich parallel und verteilt auf verschiedene *Nodes* eines *Clusters* ausgeführt (Apache Flink 2019f). In Abbildung 6 wird das Beispiel aus dem letzten Abschnitt granularer dargestellt. *Operatoren* haben je nach Parallelität ein bis n *Operator Subtasks*. Zwischen Operatoren kann es eine 1:1 Beziehung geben, dabei werden die Sortierung/Partition der Daten beibehalten. Bei so genannten *Redistributing* Strömen ist das nicht der Fall. Wie zu sehen im Beispiel zwischen *map* und *keyBy*. Dabei werden Daten vom Ausgangs-Operator an mehr oder weniger Folge-Operatoren gesendet und dabei eben selektiert oder gedoppelt.

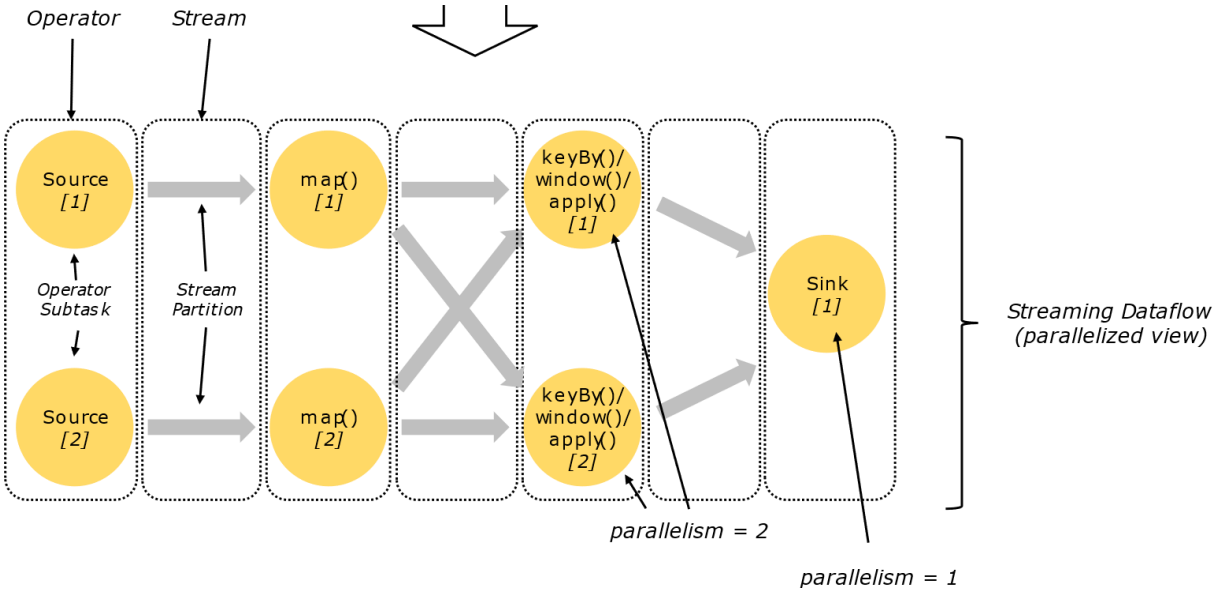

Abbildung 6: Verteilte Laufzeitumgebung
(Quelle: Apache Flink 2019e)

Dieses konzeptionelle Schema wird wie in Abbildung 7 gezeigt technisch realisiert: Der Flink Programm Code wird bei der Übersetzung optimiert und es wird, wie in Abbildung 6 gezeigt, ein gerichteter azyklischer Graph gebaut. Die *Flink Runtime* besteht aus zwei unterschiedlichen Prozess-Typen: Der *Job-Manager* koordiniert die verteilte Ausführung, also z.B. *Job-Scheduling*, und das Schreiben von *Checkpoints*, die für eine Fehlertoleranz notwendig sind. Die *TaskManager* führen *Subtasks* aus und übergeben sich die Datenströme. Jeder TaskManager ist ein Prozess der *Java Virtual Machine* (*JVM*). Dieser führt eine oder mehrere Subtasks in separaten Threads aus. Über *Task Slots* kann gesteuert werden, wie viele Tasks ein TaskManager gleichzeitig ausführt. Slots ermöglichen auch Memory Isolation, bei 3 TaskSlots wird der verfügbare Heap Speicher der JVM also gedrittelt.

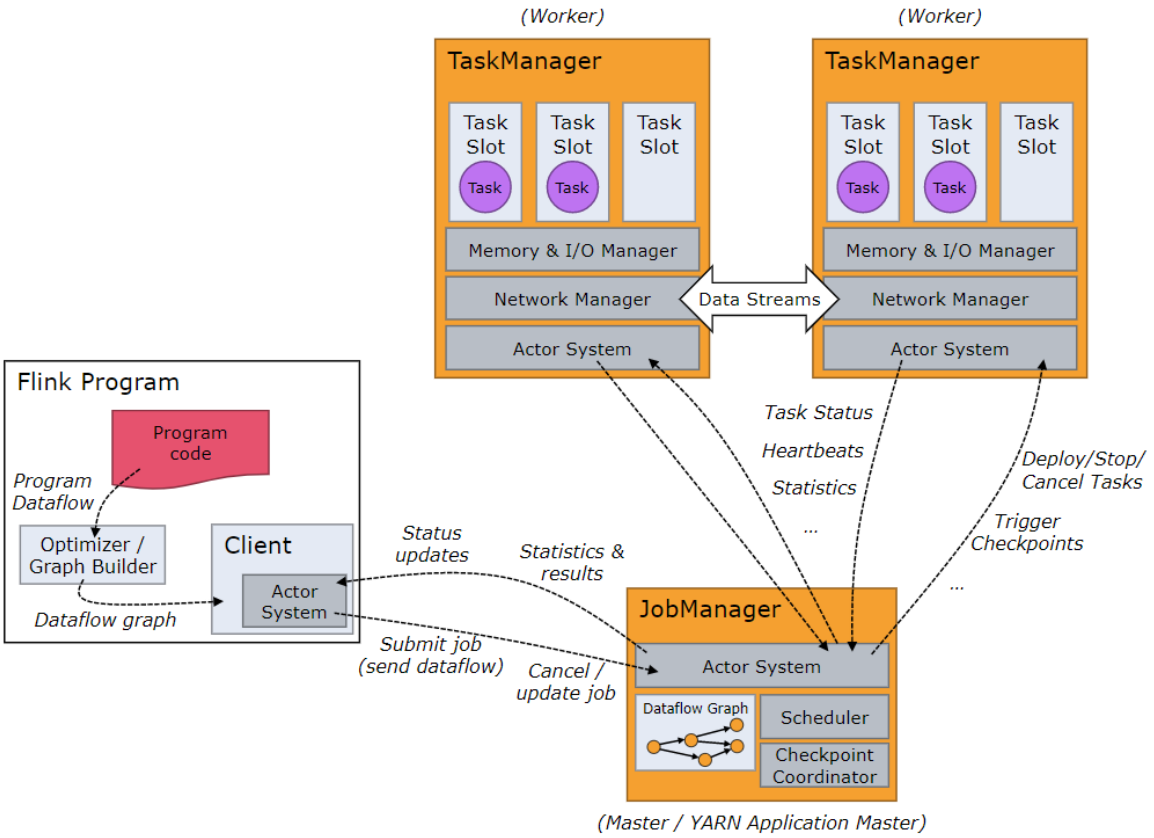

Abbildung 7: Technische Detail zur verteilten Laufzeitumgebung
(Quelle: Apache Flink 2019f)

3.5 Software Stack

Flink hat den in Abbildung 8 gezeigten Software Stack. Dabei nutzt eine Schicht immer nur die direkt nächste Schicht. Der Kern bildet die Runtime, also die verteilte *Streaming Dataflow Engine* und die Low-Level-API *ProcessFunction*. Die Runtime bekommt Programme als Graphen. Die *DataStream API* und *DataSet API* basierten auf der ProcessFunction-API generieren bei der Kompilierung solche *Job-Graphen*. Die Ausführung dieser Graphen erfolgt lokal, auf einem Cluster via YARN oder MESOS, oder in der Cloud. Weitere API‘s oder interne Bibliotheken generieren Programme über die DataStream API und DataSet API. Nennenswert sind die *Table API*, *SQL API* und *FlinkML* für *Machine Learning* (*ML*). *Third Party Programme* können an Flink angebunden werden oder werden von Flink genutzt. Nennenswert sind *Apache Zeppelin* zu Ad-Hoc Datenanalyse und *Apache SAMOA* als Werkzeug zum *Streaming Machine Learning* (Apache Flink 2019g). Die Programmierung erfolgt in Java oder Scala. Die kompilierten Programme beider Programmiersprachen werden in der JVM ausgeführt.

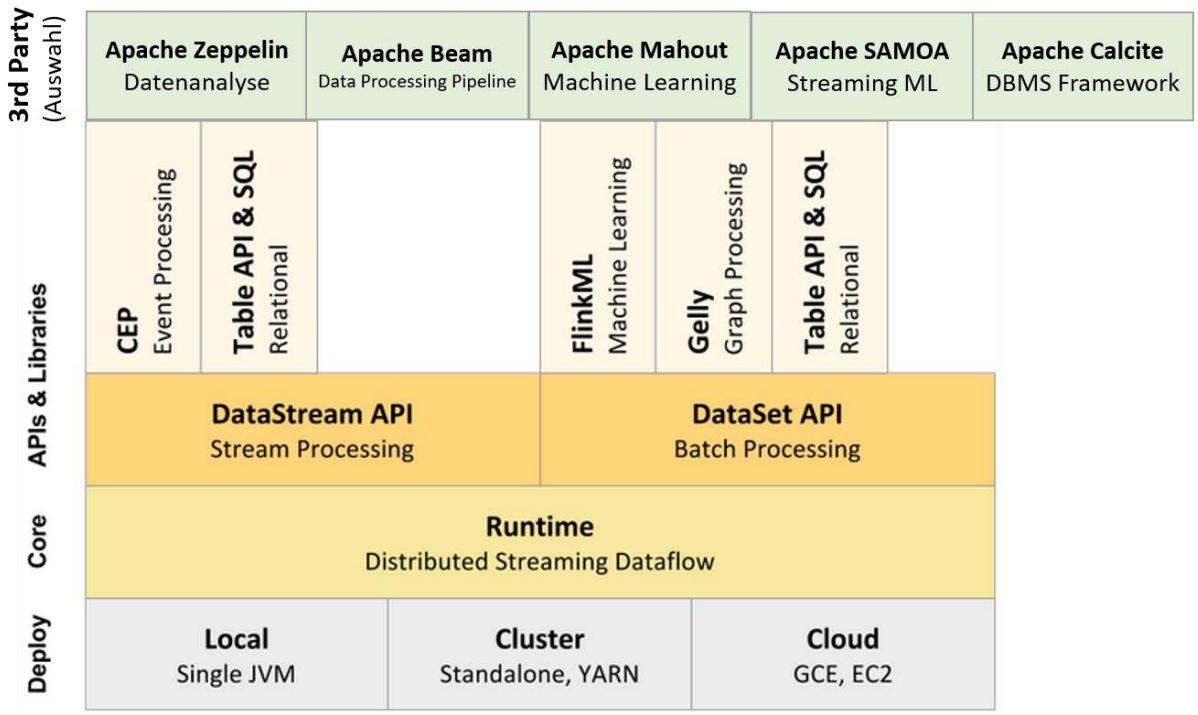

Abbildung 8: Software Stack
(Quelle: Eigene Darstellung in Anlehnung an Apache Flink 2019c)

4 Programmierschnittstellen

In diesem Abschnitt wird vertiefend auf die internen Programmierschnittstellen, also API's, von Flink eingegangen.

4.1 DataStream & DataSet API

Die DataStream API und DataSet API sind die übliche API für typische Flink-Programme. Die DataStream API wird für Unbounded Data und die DataSet API für Bounded Data genutzt (Apache Flink 2019a). Beide API's bieten ähnliche Funktionen für die Programmierenden. Die interne Struktur beider API's unterscheiden sich aber deutlich umso die unterschiedlichen Optimierungsmöglichkeiten zu realisieren (Kostas Tzoumas 2015).

In Abbildung 9 wird das klassische *Word Count* Problem mit der DataStream API und der DataSet API vergleichend gelöst.

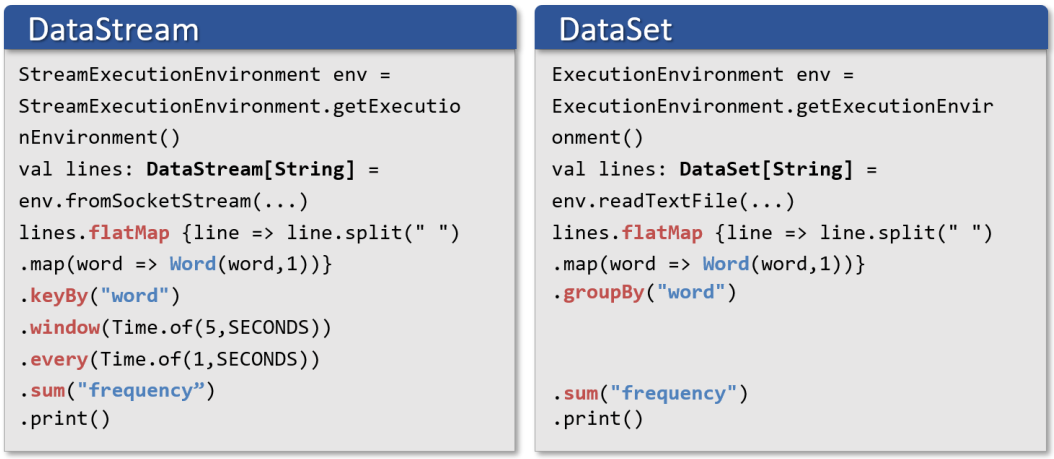

Abbildung 9: Beispiel für DataStream API und DataSet API (Scala)
(Quelle: Eigene Darstellung in Anlehnung an Tilmann Rabl 2017)

4.2 SQL & Table API

Eine Abstraktionsschicht höher sind die *SQL API* und *Table API* im Software Stack von Flink angesiedelt. Sie sind jeweils eine Unified API für Unbounded Data und Bounded Data (Apache Flink 2019k). Die Syntax von der SQL API orientiert sich an *ANSI SQL*. Die Syntax der Table API ist im *LINQ-Style* von *C# (Fabian Hueske 2017, 5 ff.)*. Beide API's liefern bei (quasi) gleichen Anfragen auch die gleichen Ergebnisse zurück. Die SQL API und Table API sind jeweils interoperabel mit der DataStream API und DataSet API. Dadurch kann im Quelltext an jeder Stelle zwischen den jeweiligen API's gewechselt werden. In Abbildung 10 wird für beide API's die gleiche Anfrage gezeigt. In diesem Beispiel wird die Anzahl der Webseitenaufrufe je Webseite gruppiert nach dem Benutzer gezählt.

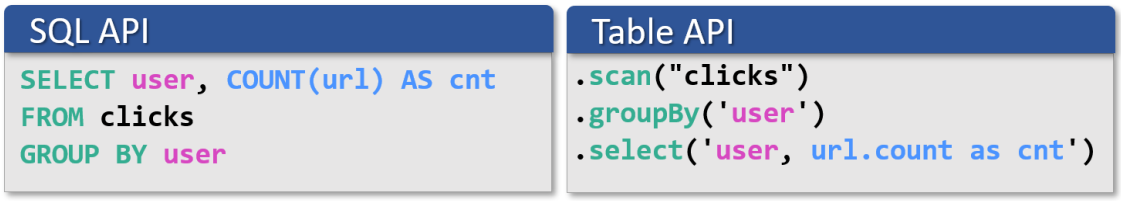

Abbildung 10: Beispiel für SQL API und Table API
(Quelle: Ververica 2019a)

5 Fehlertoleranz

In Flink gibt es zwei Mechanismen, die die Fehlertoleranz gewährleisten. Die *Checkpoints* für unerwartete Programmabbrüche und die *Savepoints* für geplante Wartungsarbeiten.

5.1 Checkpoints

Checkpoints speichern den Zustand des aktuellen Programms und die aktuelle Position in dem zu verarbeitendem Datenstrom. Checkpoints werden automatisch, kontinuierlich und teilweise

asynchron erzeugt (Apache Flink 2019d). Checkpoints sind nur notwendig bei *stateful*-Operationen wie z.B. *keyBy*, *map*, *filter*, dem *Windowing* oder der Kommunikation mit speziellen Quellen/Senken wie Apache Kafka. Sie sind auch nur bei Unbounded Data, also der DataStream API, notwendig. Bei Bounded Data, also der DataSet API, gibt es kein Checkpointing (Apache Flink 2019b). Tritt dort ein Fehler auf werden einfach alle Daten erneut verarbeitet. Flink garantiert fehlerfreie Ausführung mit Exactly-Once-Semantik. Das ist wichtig, wenn ein genaues und korrektes Ergebnis notwendig ist. In den Szenarien in denen Flink eingesetzt wird, also in einer Kappa-Architektur, ist das zwingend erforderlich. Denn die Daten werden nur von Flink verarbeitet und nicht noch durch ein langsameres, aber genaueres Werkzeug ein zweites Mal. Die Exactly-Once-Semantik wird sogar Ende-zu-Ende, inklusive Quelle und Senke, garantiert wenn die Quelle und Senke kompatibel sind (z.B. Apache Kafka). Nach einem Programmabbruch und Neustart der Verarbeitung befindet sich das Programm im Zustand, der vor dem Fehler bestand.

Technisch sind Checkpoints wie nachfolgend beschrieben und in Abbildung 11 gezeigt implementiert. Zusätzlich zu den echten Events fügt Flink zyklisch *Barrier* (*Trenner*) in den Datenstrom ein (Apache Flink 2019d). Diese werden genutzt um an jeder Stelle des gesamten Verarbeitungsstroms die gleiche Unterteilung in *Delta-Checkpoints* zu machen. Checkpoints enthalten die Position bezogen auf die Eingabe-Ströme und *Pointer* auf die Zustände im jeweiligen aktiven *State Backend*.

Es gibt aktuell drei verschiedene State Backends: Nur auf dem JVM Heap, auf dem Heap (für aktuelle Checkpoints) und Dateien (für „etwas“ ältere Checkpoints) oder via Third-Party-Werkzeug *RockDB* als Datei (Apache Flink 2019j). Welches State Backend genutzt werden soll kommt auf die Balance zwischen Performance, Hardware-Anforderung (Arbeitsspeicher) und Ausfallsicherheit an.

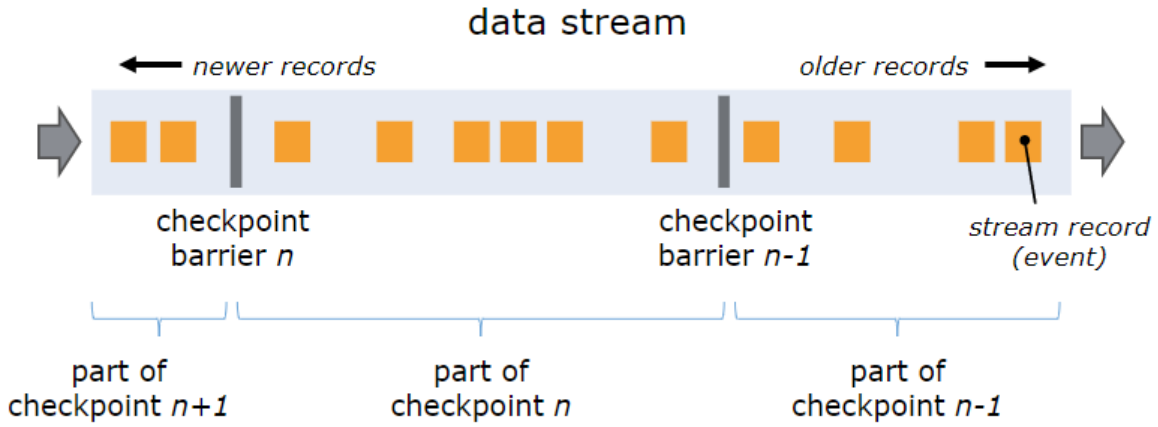

Abbildung 11: Checkpointing
(Quelle: Apache Flink 2019d)

5.2 Savepoints

Savepoints sind persistent gespeicherte Checkpoints (Apache Flink 2019i). Technisch nutzt Flink den selben Mechanismus wie beim Checkpointing. Savepoints werden aber im Gegensatz zu Checkpoints nicht automatisch erzeugt. Die Erzeugung von Savepoints muss von einem Administrator manuell getriggert werden. Die Verarbeitung kann dann nach einem Anwendungsstopp und Anwendungsstart wieder an einem Savepoint beginnen. Das geht bei Checkpoints nicht, da diese nur für automatische Neustarts nach Programmabbrüchen gedacht sind.

Anwendungsfälle für Savepoints sind das Update des eigenen Flink-Programms, einem Update der Flink-Software oder der Reskalierung des Flink-Clusters (Ververica 2016, 24 f.).

6 Machine Learning

Flink unterstützt Machine Learning über die Bibliothek FlinkML Mit dem aktuellen Release (Stand Flink 1.8) aber nur via Batch-Verarbeitung, also für Bounded Data über die DataSet API (Apache Flink 2019h). Die Verarbeitung von Unbounded Data unterstützt FlinkML also nicht. In der Praxis liegen oft aber Unbounded Data vor, z.B. bei der Fraud-Detection von Banken oder der Email-Spam-Erkennung.

Die Ziele von FlinkML orientieren sich an den Zielen von Flink: Eine parallele, verteilte Ausführung unterstützt die freie Skalierung, es soll eine einfache API geben und Werkzeuge zur automatische Code-Erzeugung umso den Code-Fußabdruck klein zu halten.

Die Vorteile des Einsatzes von Flink bei ML sind dessen Unterstützung von *Piplining*, welches die Materialisierung von großen Datenmengen (Zuständen) zwischen Funktionen reduziert (Till Rohrmann 2015). Sowie die native Unterstützung von, für ML sehr wichtigem, *Stateful Iterations* sowie *Delta Iterationen*. Im Gegensatz zur normalen Iteration werden hier bei jeder Iteration nicht alle Daten erneut zur Berechnung einbezogen, sondern es werden nur relevante Daten selektiert. Dies führt zu einem enormen Performance-Vorteil.

FlinkML unterstützt aktuell einige der wichtigsten ML-Algorithmen. Beispielsweise im Bereich *Supervised Learning*, also Klassifikation oder Vorhersage, die *Algorithmen Multiple Linear Regression*, *CoCoA* und *Optimization* Framework (Apache Flink 2019h). Im Bereich *Unsupervised Learning*, also der Mustererkennung und Erkennung von Unregelmäßigkeiten, den Algorithmus *K-Nearest Neighbors Join*. Die Roadmap von FlinkML sieht weitere neue Algorithmen vor, aber auch größere Änderungen. Die API soll einfacher, schneller und gleichzeitig auch die Verarbeitung von Unbounded Data unterstützen (Apache Flink 2019l). Alternativ arbeiten die Apache Projekte *Mahout* und *SAMOA* ebenfalls an Machine Learning in Kombination mit Flink (Apache Flink 2019g). SAMOA ist dabei eine Alternative für Streaming Machine Learning.

7 Vergleich von Flink mit Spark

Der größte Konkurrent von Apache Flink ist das ebenfalls von Apache betreute Spark. Darum wird in diesem Abschnitt in Tabelle 1 ein kurzer Vergleich zwischen beiden Big-Data-Werkzeugen vorgenommen.

Tabelle 1: Vergleich von Flink mit Spark

	Flink	**Spark**
Stream	Native Streaming	Micro-Batching (alle paar Sekunden ein Lauf)
Batch	Stream-First (Kappa)	Native Batch
Programmiersprachen	Scala, Java, (Python)	Scala, Java, Python, R
Fehlertoleranz	Exactly-Once-Semantik	
Durchsatz	Hoch	
Latenz	Millisekunden	Sekunden
Zustände	Native Unterstützung	stateless
Community	Innovationsführer, aber weniger Contributors	Große Community mit vielen Contributors

(Quelle: Eigene Darstellung in Anlehnung an Melanie Däschinger 2017; Chandan Prakash 2018)

Die Performance bzw. der mögliche Durchsatz werden von den Entwicklern von Flink und Spark mit hoch angegeben. Verschiedene Studien, die die Performance genauer untersuchten, kommen zu verschiedenen Ergebnissen. Dies liegt auch daran, dass es keinen eindeutigen Sieger gibt. Je nach Anwendungsfall und derzeitigem Entwicklungsstand beider Werkzeuge ist das eine oder das andere Werkzeug schneller. Hinzu kommt, dass sich beide Werkzeuge so rapide weiterentwickeln, dass Informationen schnell veraltet sein können.

8 Fazit & Ausblick

Flink ist ein modernes, fehlertolerantes Big-Data-Werkzeug. Durch eine Exactly-Once-Semantik kann Flink in einer Kappa-Architektur eingesetzt werden und ermöglicht so eine Real-Time-Verarbeitung mit geringer Latenz und hohem Datendurchsatz. Es ist einfach optimierbar und skalierbar. Flink ist seit seinem Projektstart Innovationsführer im Bereich der Big-Data-Werkzeuge. Flink hat oft innovative Features eingeführt, die die Konkurrenz später auch eingeführt hat. Jedoch ist die Entwickler-Community von Flink deutlich kleiner im Vergleich zu etablierten Konkurrenten wie Spark. Dadurch geht die Entwicklung langsamer voran, vor allem in Nischenbereichen wie Machine Learning. Außerdem sind aufgrund des aktuellen Reifegrades auch *Breaking-Changes* möglich die den Entwicklungsaufwand erhöhen können.

In naher Zukunft stehen große Veränderungen in der internen Struktur von Flink an. Die API's für Unbounded und Bounded Data werden zu einer Unified Batch and Streaming API zusammengeführt (Xiaowei Jiang 2018). Diese neue API-Schicht ersetzt also die bestehenden DataStream und DataSet API in ihrer derzeitigen Verwendung. Sie sind dann gleichwertig mit der Table API und SQL API. Weiterhin soll die Performance im Batch Bereich stark zu nehmen und so eine Lücke zu Spark verkleinert werden (Apache Flink 2019m). Letztlich soll Flink nicht nur ein Stream Processing Framework sein, sondern eine Stream Processing Bibliothek werden die einfach in andere Werkzeuge integriert werden kann (Ververica 2019).

Literaturverzeichnis

Aljoscha Krettek (2016): Apache Flink for IoT. How Event-Time Processing Enables Easy and Accurate Analytics. Online verfügbar unter https://www.slideshare.net/BigDataSpain/apache-flink-for-iot-how-eventtime-processing-enables-easy-and-accurate-analytics-by-aljoscha-krettek, zuletzt geprüft am 01.06.2019.

Aljoscha Krettek (2017): Apache Flink for IoT. How Stateful Event-Time Processing Enables Accurate Analytics. Online verfügbar unter https://www.slideshare.net/dataArtisans/aljoscha-krettek-apache-flink-and-iot-how-stateful-eventtime-processing-enables-accurate-analytics, zuletzt geprüft am 01.06.2019.

Apache Flink (2019a): Basic API Concepts. Online verfügbar unter https://ci.apache.org/projects/flink/flink-docs-release-1.8/dev/api_concepts.html, zuletzt aktualisiert am 2019, zuletzt geprüft am 02.06.2019.

Apache Flink (2019b): Batch (DataSet API): Fault Tolerance. Online verfügbar unter https://ci.apache.org/projects/flink/flink-docs-release-1.8/dev/batch/fault_tolerance.html, zuletzt aktualisiert am 2019, zuletzt geprüft am 02.06.2019.

Apache Flink (2019c): Component Stack. Online verfügbar unter https://ci.apache.org/projects/flink/flink-docs-release-1.8/internals/components.html, zuletzt aktualisiert am 2019, zuletzt geprüft am 02.06.2019.

Apache Flink (2019d): Data Streaming Fault Tolerance. Online verfügbar unter https://ci.apache.org/projects/flink/flink-docs-release-1.8/internals/stream_checkpointing.html, zuletzt aktualisiert am 2019, zuletzt geprüft am 02.06.2019.

Apache Flink (2019e): Dataflow Programming Model. Online verfügbar unter https://ci.apache.org/projects/flink/flink-docs-release-1.8/concepts/programming-model.html, zuletzt aktualisiert am 2019, zuletzt geprüft am 01.06.2019.

Apache Flink (2019f): Distributed Runtime Environment. Online verfügbar unter https://ci.apache.org/projects/flink/flink-docs-stable/concepts/runtime.html, zuletzt aktualisiert am 2019, zuletzt geprüft am 01.02.2019.

Apache Flink (2019g): Ecosystem. Online verfügbar unter https://flink.apache.org/ecosystem.html, zuletzt aktualisiert am 2019, zuletzt geprüft am 02.06.2019.

Apache Flink (2019h): FlinkML - Machine Learning for Flink. Online verfügbar unter https://ci.apache.org/projects/flink/flink-docs-release-1.8/dev/libs/ml/, zuletzt aktualisiert am 2019, zuletzt geprüft am 02.06.2019.

Apache Flink (2019i): Savepoints. Online verfügbar unter https://ci.apache.org/projects/flink/flink-docs-release-1.8/ops/state/savepoints.html, zuletzt aktualisiert am 2019, zuletzt geprüft am 02.06.2019.

Apache Flink (2019j): State Backends. Online verfügbar unter https://ci.apache.org/projects/flink/flink-docs-release-1.8/ops/state/state_backends.html, zuletzt aktualisiert am 2019, zuletzt geprüft am 02.06.2019.

Apache Flink (2019k): Table API & SQL. Online verfügbar unter https://ci.apache.org/projects/flink/flink-docs-stable/dev/table/index.html, zuletzt aktualisiert am 2019, zuletzt geprüft am 02.06.2019.

Apache Flink (2019): What is Apache Flink? Architecture, zuletzt aktualisiert am 2019, zuletzt geprüft am 01.06.2019.

Apache Flink (2019l): Projectmanagement tool (JIRA) of Apache Flink. Current open Issues for FlinkML. Online verfügbar unter https://bit.ly/2GYTCV4, zuletzt aktualisiert am 13.04.2019, zuletzt geprüft am 02.06.2019.

Apache Flink (2019m): Roadmap. Online verfügbar unter https://flink.apache.org/roadmap.html, zuletzt aktualisiert am 08.05.2019, zuletzt geprüft am 02.06.2019.

Chandan Prakash (2018): Spark Streaming vs Flink vs Storm vs Kafka Streams vs Samza. Choose Your Stream Processing Framework. Online verfügbar unter https://medium.com/@chandanbaranwal/spark-streaming-vs-flink-vs-storm-vs-kafka-streams-vs-samza-choose-your-stream-processing-91ea3f04675b, zuletzt geprüft am 02.06.2019.

Diego Calvo (2017): Big Data definition. Online verfügbar unter http://www.diegocalvo.es/en/big-data-definition/, zuletzt geprüft am 01.06.2019.

Erik de Nooij (2017): StreamING models, how ING adds models at runtime to catch fraudsters. Hg. v. Flink Forward. Online verfügbar unter https://sf-2017.flink-forward.org/kb_sessions/streaming-models-how-ing-adds-models-at-runtime-to-catch-fraudsters/, zuletzt aktualisiert am 01.06.2019.

Fabian Hueske (2016): Data Stream Processing with Apache Flink. Online verfügbar unter https://www.slideshare.net/fhueske/data-stream-processing-with-apache-flink, zuletzt geprüft am 01.06.2019.

Fabian Hueske (2017): Stream Analytics with SQL on Apache Flink. Ververica. Online verfügbar unter https://de.slideshare.net/dataArtisans/fabian-hueske-stream-analytics-with-sql-on-apache-flink-76998033, zuletzt geprüft am 02.06.2019.

Jesús Domínguez (2018): From Lambda to Kappa: evolution of Big Data architectures. Online verfügbar unter https://en.paradigmadigital.com/dev/from-lambda-to-kappa-evolution-of-big-data-architectures, zuletzt geprüft am 01.06.2019.

Kostas Tzoumas (2015): Batch is a special case of streaming. Online verfügbar unter https://www.ververica.com/blog/batch-is-a-special-case-of-streaming, zuletzt geprüft am 02.06.2019.

Melanie Däschinger (2017): Apache Spark vs. Apache Flink. Online verfügbar unter https://www.woodmark.de/blog/apache-spark-vs-apache-flink/, zuletzt geprüft am 02.06.2019.

Mihail Vieru; Hung Chang (2017): Complex Event Generation for Business Process Monitoring using Apache Flink. Online verfügbar unter https://jobs.zalando.com/tech/blog/complex-event-generation-for-business-process-monitoring-using-apache-flink/, zuletzt geprüft am 01.06.2019.

Mihail Vieru; Javier Lopez (2016): Apache Showdown: Flink vs. Spark. Online verfügbar unter https://jobs.zalando.com/tech/blog/apache-showdown-flink-vs.-spark/, zuletzt geprüft am 01.06.2019.

raincent (2016): Big Data Processing-Technologie. Online verfügbar unter http://www.raincent.com/content-85-7204-1.html, zuletzt geprüft am 01.06.2019.

Slim Baltagi (2016): Overview of Apache Flink: the 4G of Big Data Analytics Frameworks. Hg. v. DataWorks Summit/Hadoop Summit. Online verfügbar unter https://www.slideshare.net/HadoopSummit/overview-of-apache-flink-the-4g-of-big-data-analytics-frameworks, zuletzt geprüft am 01.06.2019.

Till Rohrmann (2015): Machine Learning with Apache Flink at Stockholm Machine Learning Group. Online verfügbar unter https://www.slideshare.net/tillrohrmann/machine-learning-with-apache-flink, zuletzt geprüft am 02.06.2019.

Tilmann Rabl (2017): Practical Big Data Processing. An Overview of Apache Flink. TU Berlin. Online verfügbar unter https://www.bbdc.berlin/fileadmin/news/documents/BBDC_DEUK_Slides/Tilmann_Rabl.pdf , zuletzt geprüft am 02.06.2019.

Tugdual Grall (2017): Introduction to Streaming with Apache Flink. Hg. v. Slideshare. Online verfügbar unter https://www.slideshare.net/tgrall/introduction-to-streaming-with-apache-flink-77391749, zuletzt geprüft am 01.06.2019.

Tyler Akidau (2015): Streaming 101: The world beyond batch. A high-level tour of modern data-processing concepts. Hg. v. O'Reilly. Online verfügbar unter https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-101, zuletzt geprüft am 01.06.2019.

Ververica (2016): Apache Flink Training - DataStream API - Fault Tolerance. Online verfügbar unter https://de.slideshare.net/dataArtisans/apache-flink-training-datastream-api-state-failure-recovery, zuletzt geprüft am 02.06.2019.

Ververica (2019a): Introduction to SQL on Apache Flink. Online verfügbar unter https://github.com/ververica/sql-training/blob/master/slides/sql-training-01-intro-to-Flink-SQL.pdf, zuletzt aktualisiert am 2019, zuletzt geprüft am 02.06.2019.

Ververica (2019): Key Takeaways from Flink Forward San Francisco 2019. Online verfügbar unter https://www.ververica.com/blog/key-takeaways-from-flink-forward-san-francisco-2019, zuletzt geprüft am 02.06.2019.

Ververica (2019b): Streaming with Apache Flink. Online verfügbar unter https://training.ververica.com/intro/intro-1.html, zuletzt aktualisiert am 2019, zuletzt geprüft am 01.06.2019.

Xiaowei Jiang (2018): Flink Forward Berlin 2018: Unified Engine for Data Processing and AI. Online verfügbar unter https://de.slideshare.net/FlinkForward/flink-forward-berlin-2018-xiaowei-jiang-keynote-unified-engine-for-data-processing-and-ai, zuletzt geprüft am 02.06.2019.