Apache Spark

Ergänzung oder Konkurrenzprodukt zu Apache Hadoop?

1. Motivation

- Wichtige Einnahmequelle und Analysepotential steckt in der Auswertung von Big Data
- Viele Frameworks vertreten mit unterschiedlichen Funktionsumfang und Nutzen
  - Batch processing → Hadoop MapReduce
  - Stream processing → Apache Storm / Apache Flink
  - Graph processing → Neo4j / Apache Giraph
- Unterschiedliche Konzepte → kein Framework für alles vorhanden mit Eigenschaften:
  - Geschwindigkeit, Zuverlässigkeit, Skalierbarkeit, Fehlertoleranz

2. Apache Hadoop

- Framework zur parallelen Verarbeitung großer Datenmengen (Big Data) auf Clustern
- skalierbar, zuverlässig, fehlertolerant
- 4 Module:
  - **Hadoop Common**:
    - Core System
  - **Hadoop Distributed File System (HDFS**
    - Hochverfügbares Dateisystem auf einem Cluster
    - Datenblöcke werden in feste Länge zerteilt
    - Master- und Slaveknoten
    - Master organisiert die Ablage auf den Slavesknoten
  - **Hadoop YARN**:
    - Framework für Job Scheduling und das Ressourcen-Management im Cluster
  - **Hadoop MapReduce**:
    - System zur parallelen Verarbeitung großer Datenmengen (Algorithmus von Google)

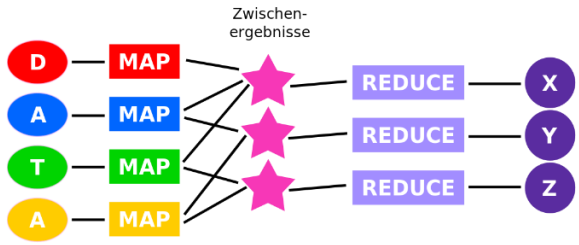


Abb. 1 - MapReduce Konzept [A1]

    - Beispiel: Preisvergleich (geringster Preis)
      - Map() → Produkte von allen Webseiten zu Preis zuordnen
      - Reduce() → Kleinsten Preis je Produkt ausgeben

3. Apache Spark

- Framework zur parallelen Verarbeitung großer Datenmengen (Big Data) auf Clustern
- Berkeley University 2013
- 2014 Top-Level-Projekt Apache
- skalierbar, zuverlässig, fehlertolerant
- Verfügbare Programmier-APIs in Java, Scala, Python, R and SQL
- Deployment auf vers. Systemen (Clustermanager) → Standalone, Hadoop Yarn, Apache Mesos, Kubernetes
- Hunderte Datenressourcen (Dateien, HDFS, Cassandra, HBase, …)
- **Batch-Processing**
- In-Memory Processing

3.1 Topologie

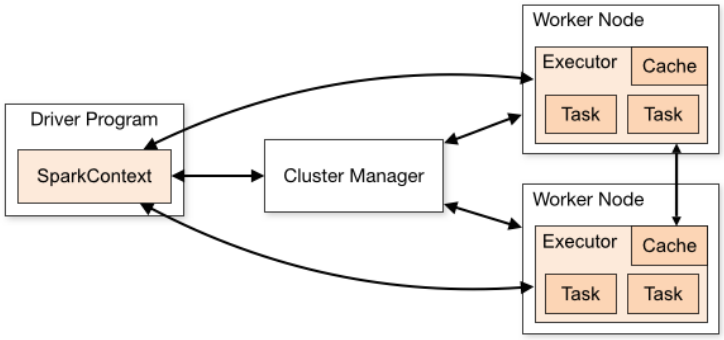


Abb. 2 - Topologie Apache Spark [A2]

- Driver Program: Der Prozess, welcher die Hauptanwendung ausführt und den SparkContext erzeugt
- SparkContext: Erstellt Directed Acyclic Graph (DAG), Partitioniert RDDs
- Cluster Manager: Job-Scheduling und Ressourcen-Management, koordiniert Worker Nodes
- Worker Node: Clusterknoten, welcher im Executer Aufgaben (Tasks) ausführt, welche vom Cluster-Manager empfangen werden

3.2 Konzept: RDD

- resilient distributed dataset
- Liste an Elementen im Cluster, die parallel verarbeitbar sind
- Operationen:
  - Transformations: Erstellt neues RDD aus dem bestehenden RDD (map(), …)
  - Actions: Gibt das Ergebnis zum Driver Program zurück (reduce(), …)
- Transformations nutzen *lazy loading* → werden erst ausgeführt, wenn eine Action auf diesem RDD aufgerufen wird

3.3 Konzept: Directed Acyclic Graph (DAG)

- Graph an Operationen auf RDDs (Knoten = RDDs, Kanten = Operationen)
- Fehlertoleranz, Optimierung und Effizienz
- Ablauf:

- Action-Aufruf übergibt DAG dem DAG Scheduler
- Graph in parallel verarbeitbare Stages aufteilen
- RDDs partitionieren (Tasks bilden)
- Verteilung über Clustermanager

- Vorteile:
  - Reihenfolge Optimierungen, Zwischenergebnisse im RAM

3.4 Libraries

3.4.1 Spark SQL

- Verarbeitung strukturierter Daten → Batch-Processing
- Verarbeitung von Datasets/Dataframes
- Interaktion über SQL oder die Dataset API
- Anbindung bestehender Business Intelligence Anwendungen
  - **Datasets** (distributed collection of data)
    - Typsicherheit + Lambda Funktionen
    - Spezieller Encoder zur Performancesteigerung (Sortierung, Filterung, Hashing ohne Deserialisierung der Objekte möglich)

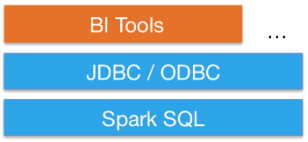


Abb. 3 - Anbindung von BI Tools [A3]

```
// Encoders are created for Java beans
Encoder<Person> personEncoder = Encoders.bean(Person.class);
Dataset<Person> javaBeanDS =
spark.createDataset(Collections.singletonList(person), personEncoder);
javaBeanDS.show();
```

Quellcode 1 – Dataset und Encoders [C1]

  - **DataFrames**
    - Datasets, wie in einer relationalen Datenbank (spaltenorientiert)
    - Dataset vom Typ Row → nicht typsicher und kein spezieller Encoder
    - Zugriff über SQL möglich

```
Dataset<Row> df = spark.read().json("/resources/people.json");
// Register the DataFrame as a SQL temporary view
df.createOrReplaceTempView("people");
Dataset<Row> sqlDF = spark.sql("SELECT * FROM people");
```

Quellcode 2 - DataFrames [C2]

3.4.2 Spark Streaming

- Streaming Bibliothek für Spark → nutzt Micro-Batch-Processing
- Kontinuierliche Inputstreams werden nach zeitlichen Intervall in Micro-Batches „zerschnitten“, damit Spark Engine diese verarbeiten kann

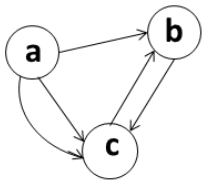
Abb. 4 - Spark Streaming Overview [A4]

3.4.3 MlLib

- Maschine Learning Bibliothek
- optimiert für iterative Berechnungen
- mögliche Anwendungsszenarien:
    - Classification: logistic regression, naive Bayes,...
    - Regression: generalized linear regression, survival regression,...
    - Statistics: summary statistics, hypothesis testing,...
    - …

3.4.4 GraphX

- Graphen als Abstraktionsebene
- Property Graph (gerichteter Multigraph mit Attributen an Knoten und Kanten)
- Datenobjekte = Knoten, Beziehungen = Kanten
- Graphenalgorithmen
    - PageRank
    - Triangle Counting
    - Shortest Path Algorithmen
    - Backtracking Algorithmen
    - …



4. Fazit

- Apache Spark (unabhängig) von Apache Hadoop
    - YARN **kann** als Clustermanager genutzt werden
    - HDFS **kann** als Datenquelle verwendet werden
- Weite Verbreitung (große Community)
    - Gute APIs, Libraries
    - Erweiterte Funktionalität
    - Ein Framework für viele Use cases
    - Bessere Performance als Hadoop durch DAG und In-Memory Processing
- Architektur bedingt keine reine Stream-Processing Unterstützung
    - Aktuelle Konkurrenz: Apache Flink

Abbildungen:

[A1] Wikipedia: MapReduce, https://de.wikipedia.org/wiki/Datei:MapReduce2.svg, zuletzt besucht 02.05.2018

[A2] SlideShare: clairvoyantllc - Intro to Apache Spark, https://de.slideshare.net/clairvoyantllc/intro-to-apache-spark-68926267, zuletzt besucht 02.05.2018

[A3] Apache Spark: SQL Library Overview, https://spark.apache.org/sql/, zuletzt besucht 02.05.2018

[A4] Apache Spark: Spark Streaming Programming Guide, https://spark.apache.org/docs/latest/streaming-programming-guide.html, zuletzt besucht 02.05.2018

Quellcode:

[C1] Apache Spark: Creating-datasets, https://spark.apache.org/docs/latest/sql-programming-guide.html#creating-datasets, zuletzt besucht 02.05.2018

[C2] Apache Spark: Running SQL Queries Programmatically, https://spark.apache.org/docs/latest/sql-programming-guide.html#running-sql-queries-programmatically, zuletzt besucht 02.05.2018

Internetquellen:

[1] Apache Spark: Spark Overview, https://spark.apache.org/, zuletzt besucht 02.05.2018

[2] Apache Spark: SQL Programming Guide, https://spark.apache.org/docs/latest/sql-programming-guide.html, zuletzt besucht 02.05.2018

[3] Apache Spark: Streaming Programming Guide, https://spark.apache.org/docs/latest/streaming-programming-guide.html, zuletzt besucht 02.05.2018

[4] Apache Spark: ML Guide, https://spark.apache.org/docs/latest/ml-guide.html, zuletzt besucht 02.05.2018

[5] Apache Spark: GraphX Programming Guide, https://spark.apache.org/docs/latest/graphx-programming-guide.html, zuletzt besucht 02.05.2018

[6] Reynold S. Xin, Joseph E. Gonzalez, Michael J. Franklin, Ion Stoica: GraphX: A Resilient Distributed Graph System on Spark, http://www.istc-cc.cmu.edu/publications/papers/2013/grades-graphx_with_fonts.pdf, zuletzt besucht 02.05.2018

[7] Data Flair: Apache Spark vs. Hadoop MapReduce, https://data-flair.training/blogs/apache-spark-vs-hadoop-mapreduce/, Stand 19.09.2016, zuletzt besucht 02.05.2018

[8] Data Flair: Directed Acyclic Graph DAG in Apache Spark, https://data-flair.training/blogs/dag-in-apache-spark/, Stand: 08.04.2017, zuletzt besucht 02.05.2018

[9] Michael Pisula, Konstantin Knauf: Spark versus Flink – Rumble in the (Big Data) Jungle, https://www.heise.de/developer/artikel/Spark-versus-Flink-Rumble-in-the-Big-Data-Jungle-3264705.html?seite=all, Stand: 15.07.2016, zuletzt besucht 02.05.2018