MapReduce

Programmiermodell für parallele Berechnungen

Jennifer Wagner (01643907)
Lukas Truppe (01617081)

February 20, 2020

Inhalt

1. Einführung
2. Definition
3. Beispielhaftes Problem
4. Fazit

Einführung

Motivation

- Von Google entwickelt
- Absicht: In einem schnell wachsenden World Wide Web einen Index aller Seiten erstellen und pflegen
- Wie sollen so große Datenmengen verarbeitet werden?
- Benötigt: Skalierbarer Algorithmus bei unbekannter Anzahl an verarbeitenden Knoten in einem verteilten System

Motivation

- Modell für verteilte Dateisysteme: Global File System
- Kein Scheduling, keine Parallelisierung, keine Fehlerbehandlung und selbstständige Verteilung der Rechenkapazitäten
- Google nutzt eine große Anzahl an absichtlich billiger, fehleranfälliger Hardware, deshalb ist Fehlertoleranz eine wichtige Eigenschaft

Apache Hadoop

- Open-Source-Implementierung von MapReduce
- Framework zur Verarbeitung von großen Mengen an Daten
- Unterstützung von verteilten Netzwerken
- Fehlertoleranz, Load Balancing, etc.
- Gilt immer mehr als veraltet und wird in Zukunft durch andere Ausführungsverfahren ersetzt

Heutiger Stand

- MapReduce wurde bis 2014 in Form von Apache Hadoop eingesetzt
- Danach hat Google beschlossen, MapReduce nicht mehr einzusetzen
- Apache Spark ersetzt MapReduce und verspricht, zwischen drei und 100 mal schneller zu sein
- Spark bietet spezielle Pakete für Machine Learning, Graphenberechnungen, etc.

Definition

Überblick

- MapReduce ist ein Konzept, das für die paralelle Verarbeitung von großen Datensätzen eingesetzt wird.
- Map: Knoten berechnen Schlüssel/Wert-Paare aus einem Teil der zu bearbeitenden Daten
- Shuffle: Sortieren und Gruppieren der Schlüssel/Wert-Paare nach Schlüssel
- Reduce: Vereinfachung der Gruppen

Beispiel

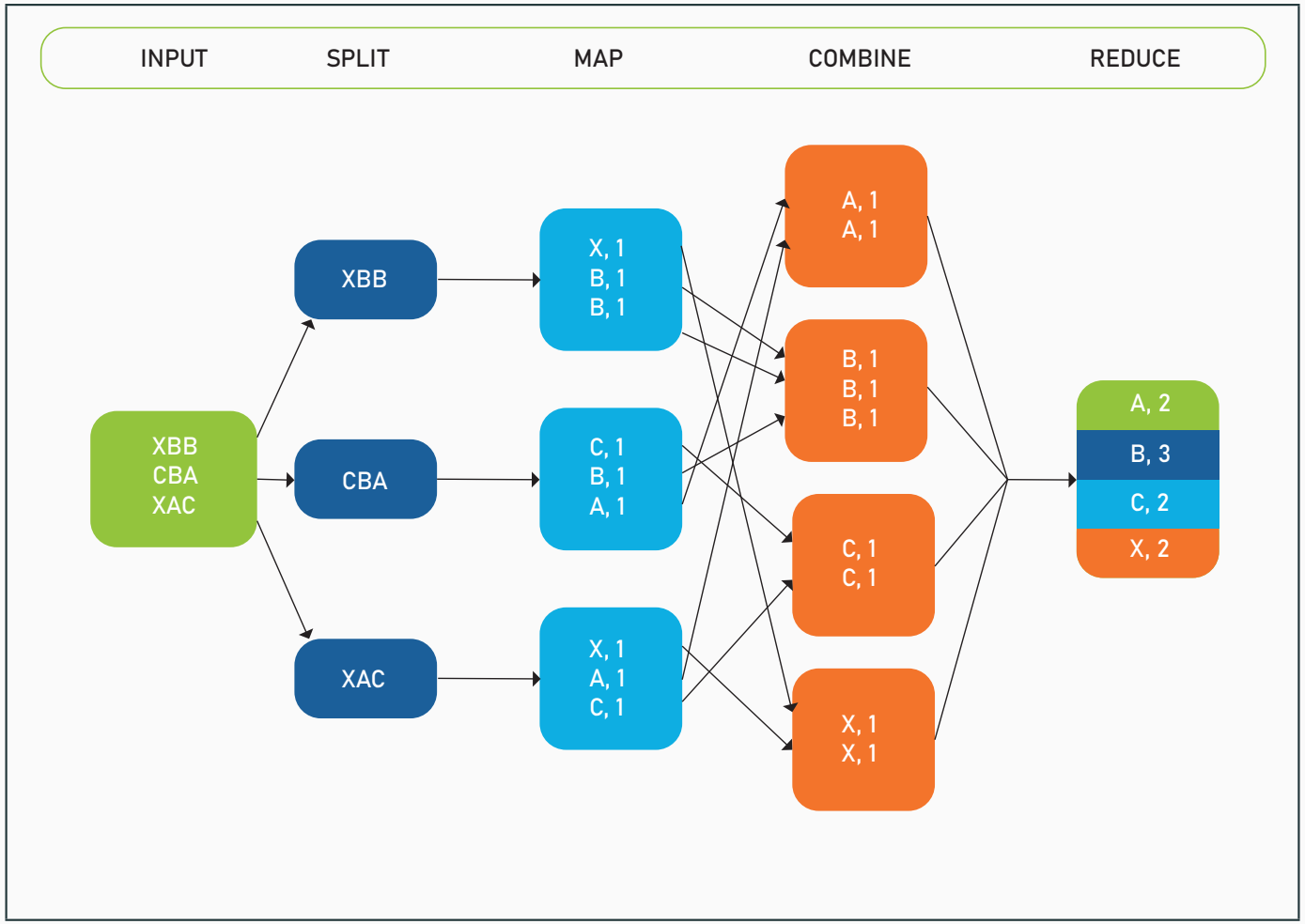

Figure 1: MapReduce für Zählaufgabe [1]

[1] Original Bild von https://de.talend.com/resources/what-is-mapreduce/

Formale Definition

Map: Der Map-Schritt besteht aus der Anwendung einer Funktion f, die als Eingabe ein Schlüssel/Wert-Paar $< k, v >$ akzeptiert und als Ausgabe eine Liste von Schlüssel/Wert-Paaren $< k_1, v_1 >$, $< k_2, v_2 >, \dots, < k_n, v_n >$ erzeugt. Ein- und Ausgabe sind voneinander unabhängig. Die Schlüssel in den Tupeln der Ausgabe von f müssen nicht eindeutig sein.

Visualisierung

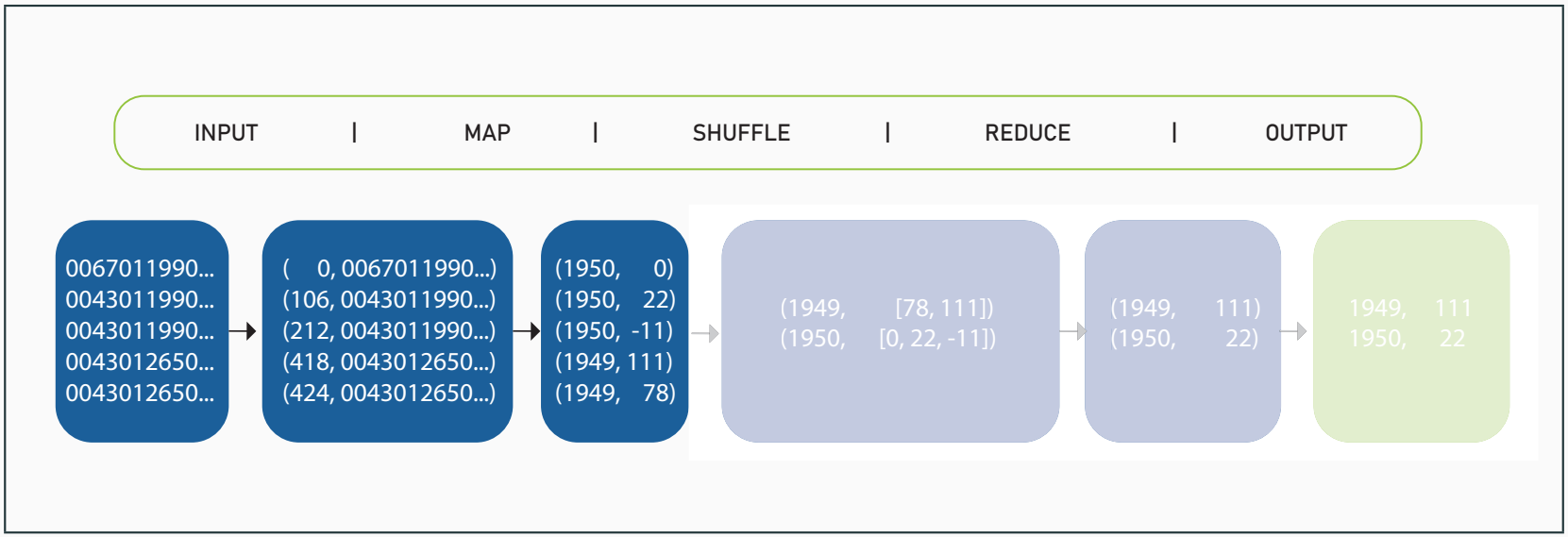

Figure 2: Map[2]

[2] Original Bild von https://dbs.uni-leipzig.de/file/seminar_0910_findling_König.pdf

Formale Definition

Shuffle: In einem Zwischenschritt namens Shuffle wird das Ergebnis aus dem Map-Schritt in einer Liste aus Schlüssel/Werte-Paaren $< k_1, l_1 >, < k_2, l_2 >, \dots, < k_m, l_m >$ zusammengefasst, wobei m der Anzahl distinkter Schlüssel k entspricht und jeder der Werte l_i jedes Paares eine Liste aus Werten $\{v_1, v_2, \dots, v_j\}$ ist, für die gilt, dass sie den Schlüssel k_i haben.

Visualisierung

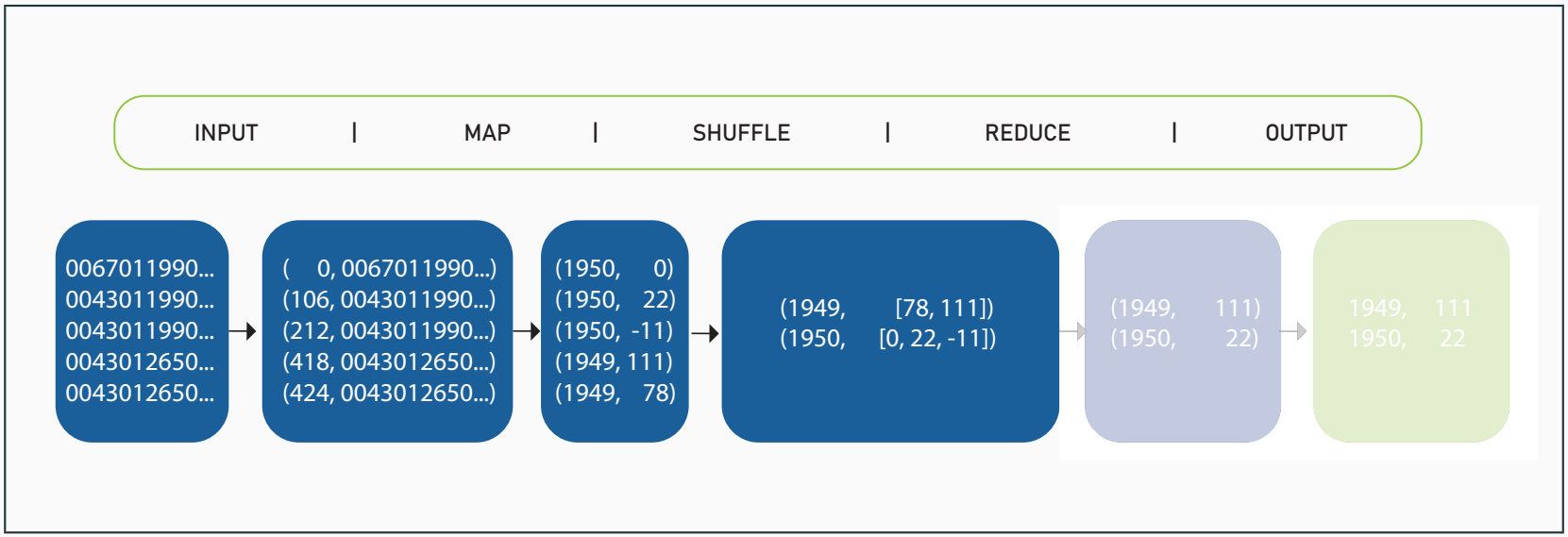

Figure 3: Shuffle[3]

[3] Original Bild von https://dbs.uni-leipzig.de/file/seminar_0910_findling_König.pdf

Formale Definition

Reduce: Im Reduce-Schritt wird eine Funktion f angewendet, die als Eingabe einen Schlüssel und eine Liste an dazugehörigen Werten $< k, \{v_1, v_2, \dots, v_j\} >$ akzeptiert. Das Ergebnis dieses Reduce-Schrittes ist dann ein Paar aus einem Schlüssel und einem dazugehörigen Wert $< k, v' >$, wobei die Werte $v_1, v_2, \dots, v_j$ auf v' reduziert wurden.

Visualisierung

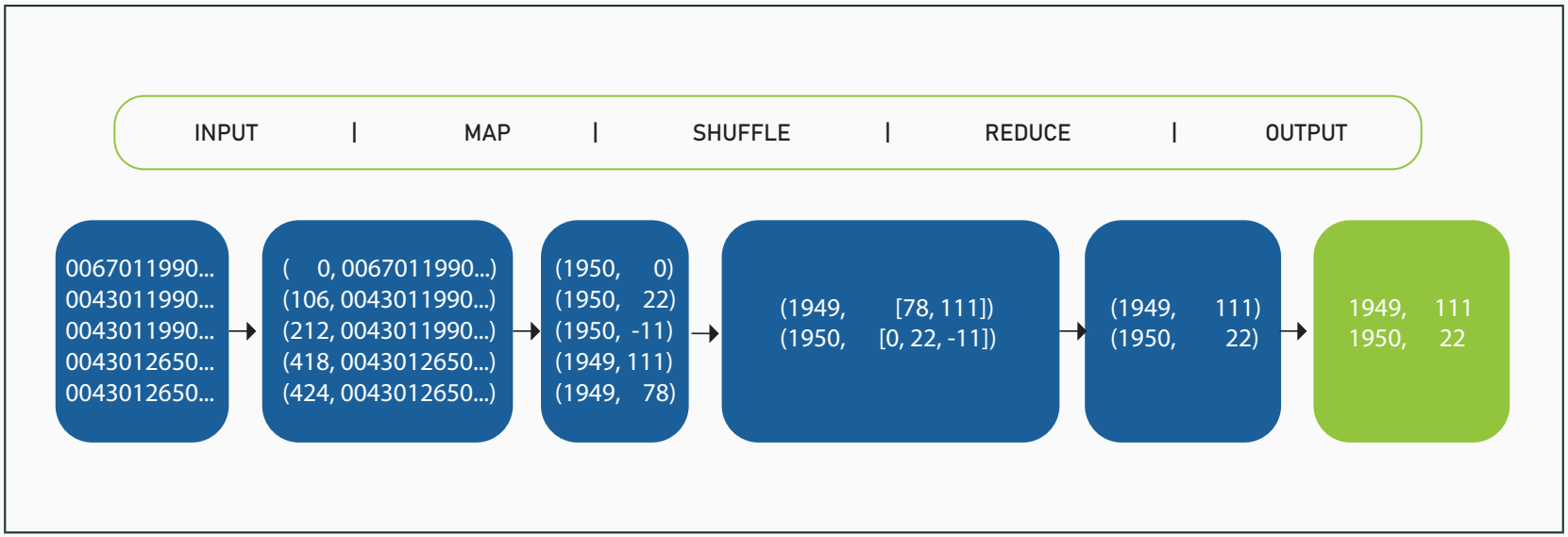

Figure 4: Reduce[4]

[4] Original Bild von https://dbs.uni-leipzig.de/file/seminar_0910_findling_König.pdf

Abgrenzung zum Paradigma der funktionalen Programmierung

- Funktionale Programmierung: Map/Reduce (bzw. Fold)
- Map wendet Transformierung auf alle Elemente einer Liste an
- Reduce fasst Elemente einer Liste zu einem Element zusammmen
- Ähnlich zu Transformierung von Schlüssel/Wert-Paaren
- Unterschied liegt in Reduce: In der parallelen Programmierung werden nur jeweils einzelne Teile der Daten zusammengefasst

Beispielhaftes Problem

Aufgabe

- Erstellung eines Indexes über Menge von Dokumenten zur Implementierung einer Suchfunktion
- Ab einer gewissen Menge an Dokumenten sind Rechen-/Speicheraufwand zu groß für ein einzelnes System
- Lösung: Einsatz von MapReduce

Lösung mit MapReduce

- Eingabedaten müssen als Schlüssel/Wert-Paare vorliegen, also als Tupel $< k, v >$, wobei k ein Verweis auf ein Dokument und v der Inhalt des Dokuments ist.
- In der Shuffle-Phase werden nun alle Ergebnistupel $< k', v' >$ nach Schlüsseln gruppiert. Alle Tupel $< k', v' >$ mit dem gleichen Schlüssel k' werden zu einem Tupel $< k', l >$ zusammengefasst, wobei $l = \{v_1, v_2, ..., v_i\}$ eine Liste aus allen Verweisen auf Dokumente mit dem Schlüssel k' ist.
- In der Reduce-Phase wird nun ein Tupel $< k', l >$ jeweils einem Knoten zugeordnet, der Duplikate aus der Liste entfernt und eine Liste aus Ergebnistupeln der Form $< k', l' >$ ist.

Visualisierung

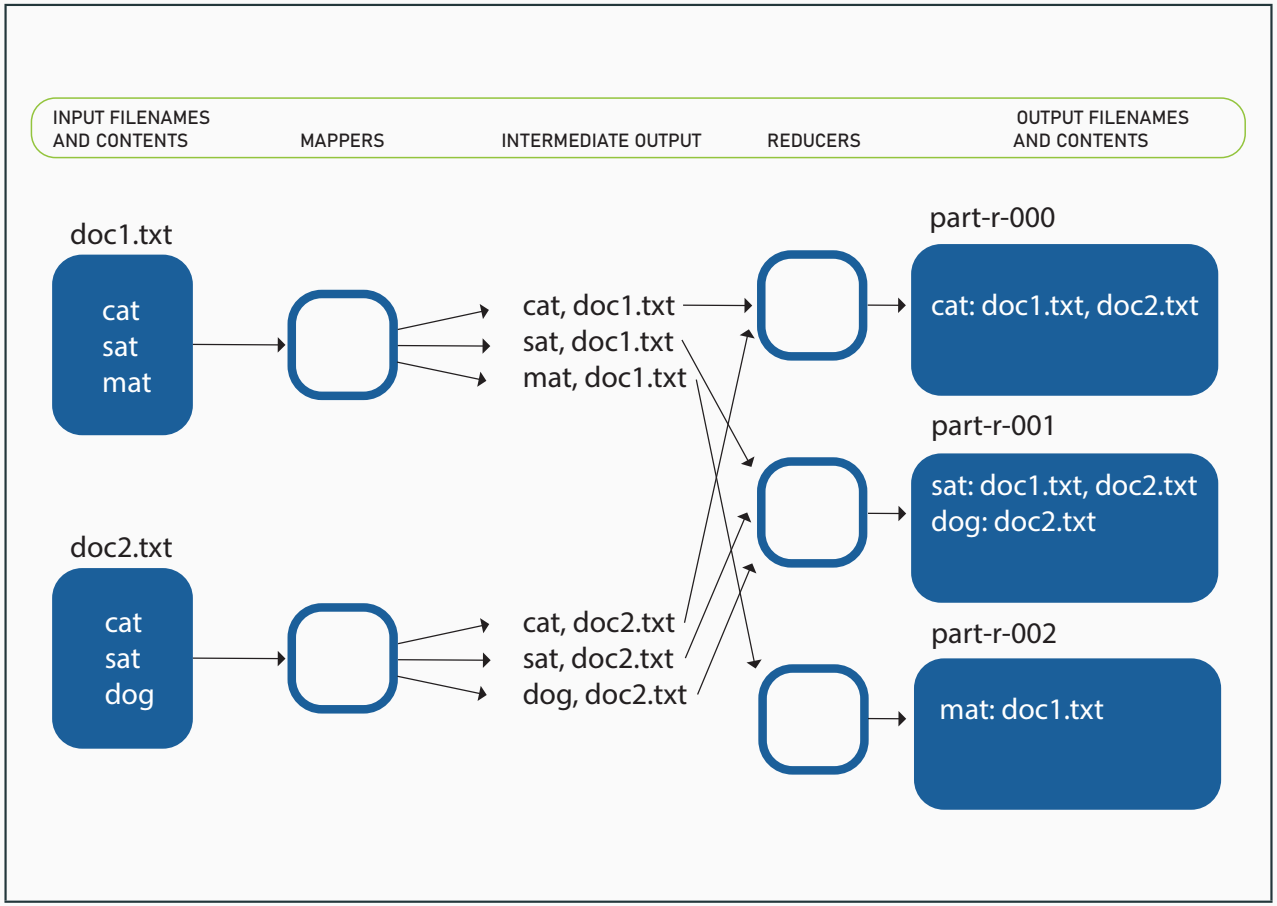

Figure 5: Indexerstellung in MapReduce[5]

[5] Original Bild von https://livebook.manning.com/book/hadoop-in-practice-second-edition/chapter-1/139

Vergleich mit anderen Lösungen

- Rechenaufwand, der ein einzelnes System überlasten würde, kann auf mehrere bei weitem nicht so leistungsfähige Systeme aufgeteilt werden.
- Um einen Index über alle Dokumente zu bauen, müsste ein einzelnes System alle Dokumente durchgehen und dabei den Index erstellen.
- Durch die Anwendung von MapReduce wird diese Arbeit auf viele Systeme aufgeteilt, wobei keines davon jemals alle Dokumente verarbeiten muss.

Fazit

Aktueller Stand und Entwicklungen

- MapReduce ist ein sehr gutes und weit verbreitetes Konzept für verteilte Datenverarbeitung
- Inzwischen gilt MapReduce aber wieder als veraltet
- Nachteil von MapReduce ist die lineare Datenverarbeitung unter Verwendung der Festplatte. RDD verarbeitet Datensätze unabhängig voneinander im Arbeitsspeicher.
- ”Nachfolger”: Apache Spark - Resilient Distributed Dataset (RDD)

Vielen Dank für die Aufmerksamkeit!

Quellen

MapReduce (Jeffrey Dean, Sanjay Ghemawat)
https://static.googleusercontent.com/media/research.google.com/de/archive/mapreduce-osdi04.pdf