

Middleware – Cloud Computing – Übung

MapReduce: Übersicht & Ablauf

Wintersemester 2020/21

Michael Eischer, Laura Lawniczak, Tobias Distler

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Verteilte Systeme und Betriebssysteme)

www4.cs.fau.de



Lehrstuhl für Verteilte Systeme
und Betriebssysteme



FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

MapReduce

- Einführung und Grundlagen

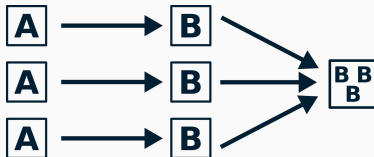
- Ablauf eines MapReduce-Jobs

- Aufgaben des Frameworks

MapReduce

Einführung und Grundlagen

- Programmiermodell zur Strukturierung von Programmen für **parallele, verteilte** Ausführung
- Map und Reduce ursprünglich Bausteine aus funktionalen Programmiersprachen (z. B. LISP)
 - **Map**: Abbildung eines Eingabeelements auf ein Ausgabeelement
 - **Reduce**: Zusammenfassung mehrerer gleichartiger Eingaben zu einer einzelnen Ausgabe
- Formulierung zu lösender Aufgaben in MapReduce
 - Aufteilen in (potentiell mehrere) Map- und Reduce-Schritte
 - Implementierung der Map- und Reduce-Methoden (Entwickler)
 - Parallelisierung und Verteilung (MapReduce-**Framework**)



- „*MapReduce: Simplified data processing on large clusters*” (OSDI’04)
- Ursprüngliche Implementierung von Google nicht öffentlich
- Zahlreiche Open-Source-Implementierungen (z. B. **Apache Hadoop**, Disco, MR4C)
 - Ermöglicht Verarbeitung riesiger Datenmengen
 - Vereinfachung der Anwendungsentwicklung

- Literatur



Jeffrey Dean and Sanjay Ghemawat

MapReduce: Simplified data processing on large clusters

Proceedings of the 6th Conference on Operating Systems Design and Implementation (OSDI '04), pages 137–150, 2004.

Hadoop-Framework (Komponenten)

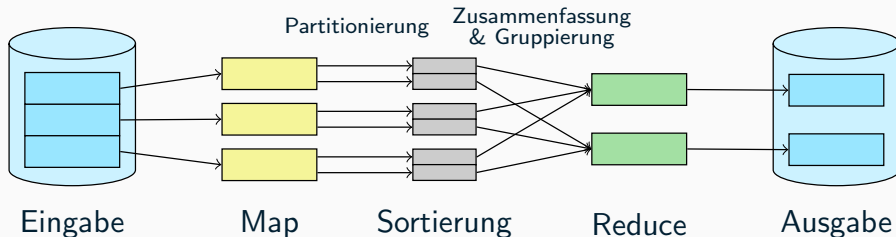


Quelle der Illustration: <https://blog.codecentric.de/2013/08/einfuehrung-in-hadoop-die-wichtigsten-komponenten-von-hadoop-teil-3-von-5/>

MapReduce

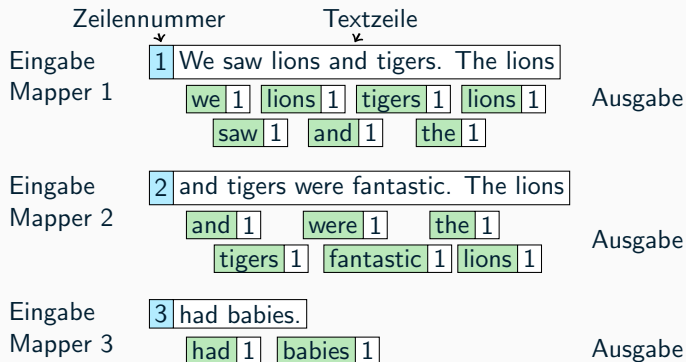
Ablauf eines MapReduce-Jobs

- Übersicht: Ablauf eines MapReduce-Durchlaufs



- Darstellung der Daten in Form von **Schlüssel-Wert-Paaren**

- Abbildung in der Map-Phase
 - Parallele Verarbeitung verschiedener Teilbereiche der Eingabedaten
 - Eingabedaten in Form von Schlüssel-Wert-Paaren
 - Abbildung auf **variable Anzahl** von **neuen** Schlüssel-Wert-Paaren
- Beispiel: Zählen von Wörtern



■ Schnittstelle **Mapper** in Apache Hadoop

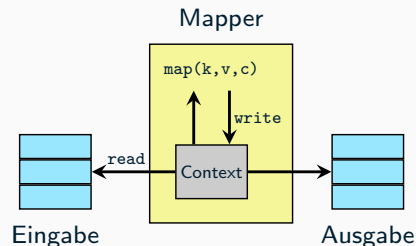
```
public class Mapper<KEYIN, VALUEIN, KEYOUT, VALUEOUT> {  
    void map(KEYIN key, VALUEIN value, Context context) {  
        context.write((KEYOUT) key, (VALUEOUT) value);  
    }  
}
```

■ Festlegen von Datentypen mittels „Generics“

■ Parameter:

- **key**: Schlüssel, z. B. Zeilennummer
- **value**: Wert, z. B. Inhalt der Zeile
- **context**: Ausführungskontext, enthält `write()`-Methode zur Ausgabe von Schlüssel-Wert-Paaren

- Jeder Mapper referenziert einen Context
- Context verwaltet die Ein- und Ausgabe des Mappers
 - Stellt Reader zum Lesen von Schlüssel-Wert-Paare aus Eingabe des Mappers bereit
 - Sammelt und bereitet Schlüssel-Wert-Paare für die Ausgabe des Mappers vor (z.B. Partitionierung)
- Datenfluss im Mapper:
 1. Reader liest nächstes Schlüssel-Wert-Paar aus Eingabe
 2. Schlüssel-Wert-Paar wird an map-Funktion übergeben
 3. map-Funktion übergibt Context das Ergebnis
 4. Context schreibt Ergebnis in die Ausgabe

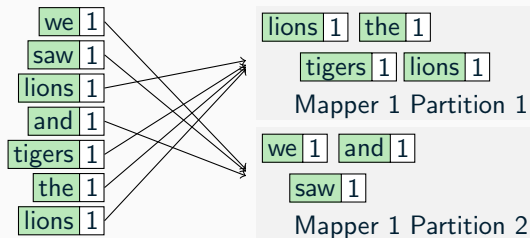


Context-Parameter in map

Der Parameter `context` in `map` zeigt auf das selbe Objekt, dass der Mapper intern verwendet. Diese Trennung erlaubt eine komfortablere Implementierung der `map`-Funktion.

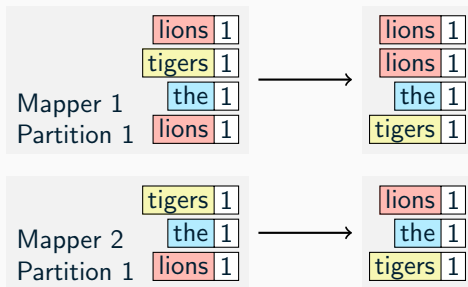
- Zuordnung der Mapper-Ausgabe zu späterem Reducer
 - Gleiche Schlüssel müssen zu gleichem Reducer
 - Eingaben der einzelnen Reducer sind unabhängig → parallelisierbar
- Schnittstelle **Partitioner** in Apache Hadoop

```
public class Partitioner<KEY, VALUE> {  
    int getPartition(KEY key, VALUE value, int numPartitions) {  
        return (key.hashCode() & Integer.MAX_VALUE) % numPartitions;  
    }  
}
```



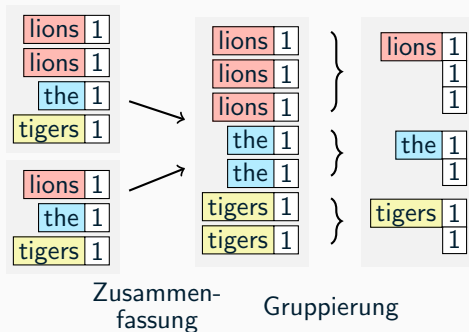
■ Sortieren der Partitionen nach **Schlüssel**

- Lokale Vorsortierung nach Verarbeitung der Daten durch Mapper
- Jede Partition wird einzeln sortiert



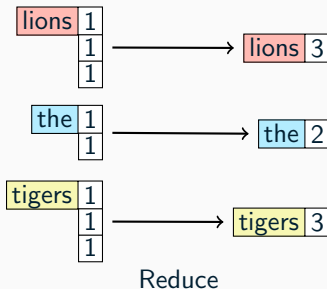
Sortierung

- Zusammenfassen und Gruppierung der Daten nach **Schlüssel**
 - Eingaben für Reducer befinden sich in (mehreren) Mapper-Ausgaben
 - **Zusammenfassung** der vorsortierten Partitionen zu einer vollständig sortierten Gesamtliste
 - **Gruppierung** aller Werte unter identischem Schlüssel
 - Statt Schlüssel-Wert-Paar nun Schlüssel und **Liste von Werten**



■ Zusammenführen von Daten in der Reduce-Phase

- Eingabe in Form von Schlüssel und allen zugehörigen Werten aus Mapper
- Parallele Verarbeitung verschiedener Teilbereiche von Schlüssel
- Abbildung auf **variable Anzahl** von **neuen Schlüssel-Wert-Paaren**



■ Schnittstelle **Reducer** in Apache Hadoop:

```
public class Reducer<KEYIN, VALUEIN, KEYOUT, VALUEOUT> {  
    void reduce(KEYIN key, Iterable<VALUEIN> values, Context context) {  
        for(VALUEIN value : values) {  
            context.write((KEYOUT) key, (VALUEOUT) value);  
        }  
    }  
}
```

■ Parameter:

- **key**: Schlüssel aus Sortierungsphase
- **values**: Liste von Werten, welche zu dem Schlüssel gruppiert wurden
- **context**: Ausführungskontext, enthält `write()`-Methode zur Ausgabe von Schlüssel-Wert-Paaren

MapReduce

Aufgaben des Frameworks

- Generelle **Steuerung** der MapReduce-Abläufe
 - Scheduling einzelner (Teil-)Aufgaben
 - Einhaltung der Reihenfolge bei Abhängigkeiten
 - Zwischenspeicherung der Daten
- Implementiert grundsätzliche **Algorithmen** (z. B. Sortierung)
- Bereitstellen von **Schnittstellen** zur Anpassung von
 - Dateneingabe (Deserialisierung)
 - Mapper
 - Partitionierung
 - Sortierung/Gruppierung
 - Reducer
 - Datenausgabe (Serialisierung)