ScaDS DRESDEN LEIPZIG

UNIVERSITÄT LEIPZIG

AUS TRADITION
GRENZEN ÜBERSCHREITEN

Cloud Data Management

Kapitel 4: MapReduce

Dr. Eric Peukert

Wintersemester 2017

Universität Leipzig, Institut für Informatik
http://dbs.uni-leipzig.de

Inhaltsverzeichnis

Parallele Programmierung

- Effiziente Verarbeitung großer Datenmengen erfordert verteilte Berechnung auf mehreren Knoten
  - Divide-and-Conquer: Aufteilung in kleine(re) Sub-Tasks, unabhängige Ausführung und Kombination der Ergebnisse
- Probleme
  - Zerlegung des Problems in parallel ausführbare Teilprobleme (Tasks)
  - Zuordnung von Tasks zu Knoten/Prozessen (Workers)
  - Bereitstellen der notwendigen Daten pro Worker (Datenzugriff bei Shared Memory)
  - Synchronisation verschiedener Worker (u.a. Deadlock-Vermeidung)
  - Behandlung von Hardware-Ausfällen

Parallele Programmierung(2)

- Programmier-Frameworks: OpenMP (Open Multi Processing), MPI
  - Logische Abstraktion der Parallelisierung (u.a. durch Funktionen, Compilerdirektiven)
  - Fokus auf CPU-intensive Anwendungen
  - Datenbereitstellung muss durch Programmierer realisiert werden
- Message Passing Model
  - Inter Process Communication (blockierend/nicht-blockierend)
  - Sende und Empfangsoperationen zwschen Prozessen, point to point communications
  - Beispiel:

```c
/* C Example */
#include <stdio.h>
#include <mpi.h>


int main (argc, argv)
     int argc;
     char *argv[];
{
  int rank, size;

  MPI_Init (&argc, &argv);      /* starts MPI */
  MPI_Comm_rank (MPI_COMM_WORLD, &rank);        /* get current process id */
  MPI_Comm_size (MPI_COMM_WORLD, &size);        /* get number of processes */
  printf( "Hello world from process %d of %d\n", rank, size );
  MPI_Finalize();
  return 0;
}
```

MapReduce

- Framework zur automatischen Parallelisierung von Auswertungen auf großen Datenmengen [MapReduce]
- Nutzung v.a. zur Verarbeitung riesiger Mengen teilstrukturierter Daten in einem verteilten Dateisystem
  - Konstruktion Suchmaschinenindex
  - Clusterung von News-Artikeln
  - Spam-Erkennung …
- Ausnutzung vorhandener Datenpartitionierung (GFS, Bigtable)
- Verwenden zweier Funktionen Map und Reduce
  - Map: Verarbeitung von Key-Value-Paaren, Generierung und dynamische Partitionierung von Zwischenergebnissen (abgeleitete Key-Value-Paare)
  - Reduce: Mischen aller Zwischenergebnisse mit demselben Key; Datenreduktion; Generierung von Key-Value Paaren als Endergebnis

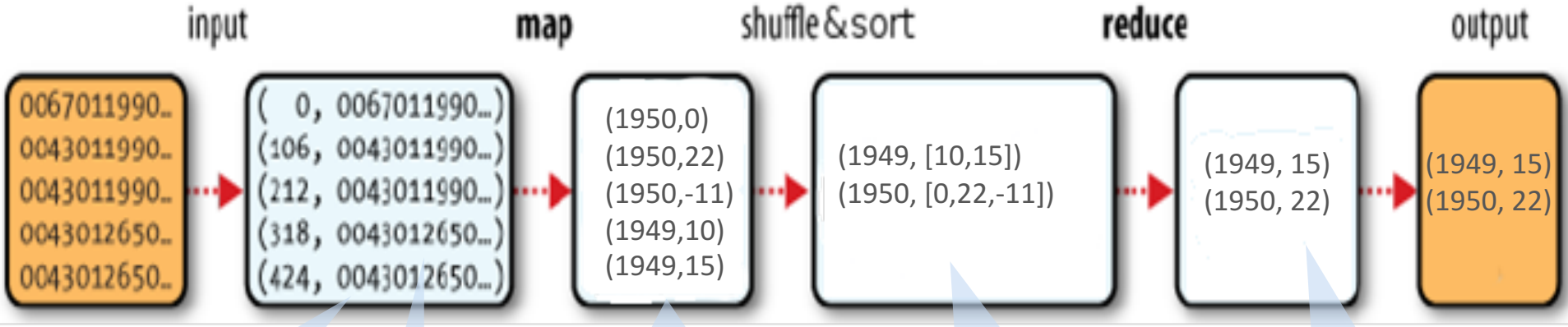
Hintergrund: Funktionale Programmierung

- Funktion **map**: (Funktion, Liste) → Liste
  - Wendet eine übergebene Funktion *f*: x → y auf jedes Element der Liste an
  - Bsp: map (x*x, [1,2,3,4,5])

- Funktion **fold**: (Funktion, Startwert, Liste) → Wert
  - Wendet eine übergebene Funktion g : x ,y→ z sukzessive auf die Elemente der Liste an und erzeugt einen (kombinierten) Wert
  - Bsp: fold (x+y, 0, [1, 4 , 9 , 16 , 25])

MapReduce: Datenfluss-Beispiel

- Bestimmen der höchsten aufgezeichneten Temperatur pro Jahr in einem großen ASCII-File mit Wetterdaten
  - Zeile: 0043011990999991**1950**051512004...9999999N9+**0022**1+99999999999...



k_1 vom Typ Long (File-Offset)

v_1 vom Typ String (Zeile selbst)

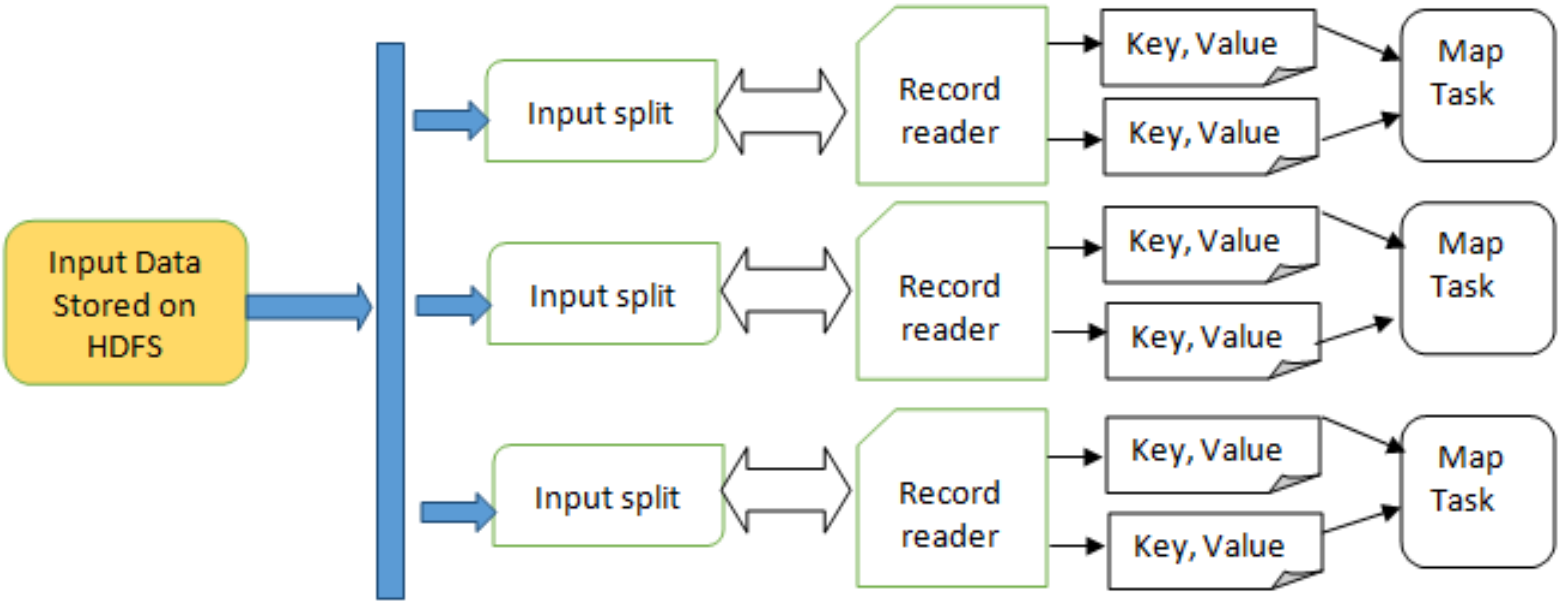
pro (k_1,v_1)-Paar wird ein (k_2,v_2)-Paar erzeugt (1-elementige Liste)

Sortieren der (k_2,v_2)-Paare und gruppieren nach k_2

Zusammenfassung (Maximum) Aller v_2 zu einem k_2 (1-elementige Liste)

Adaptiert von: Hadoop The Definitive Guide, 2009, Tom White, O'Reilly

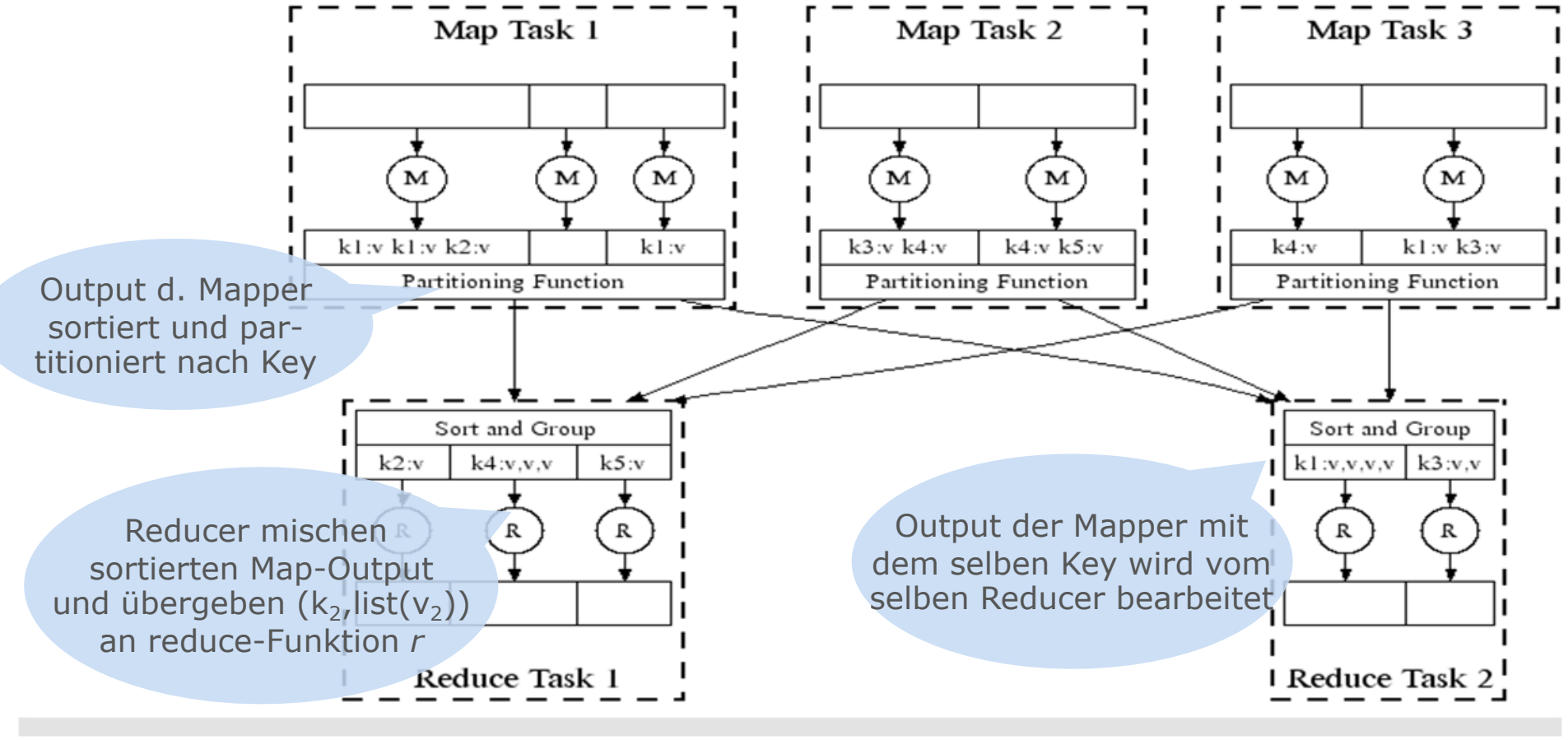
Map Reduce – Map Tasks



http://hadooptutorials.co.in/tutorials/hadoop/understanding-map-reduce-programming.html

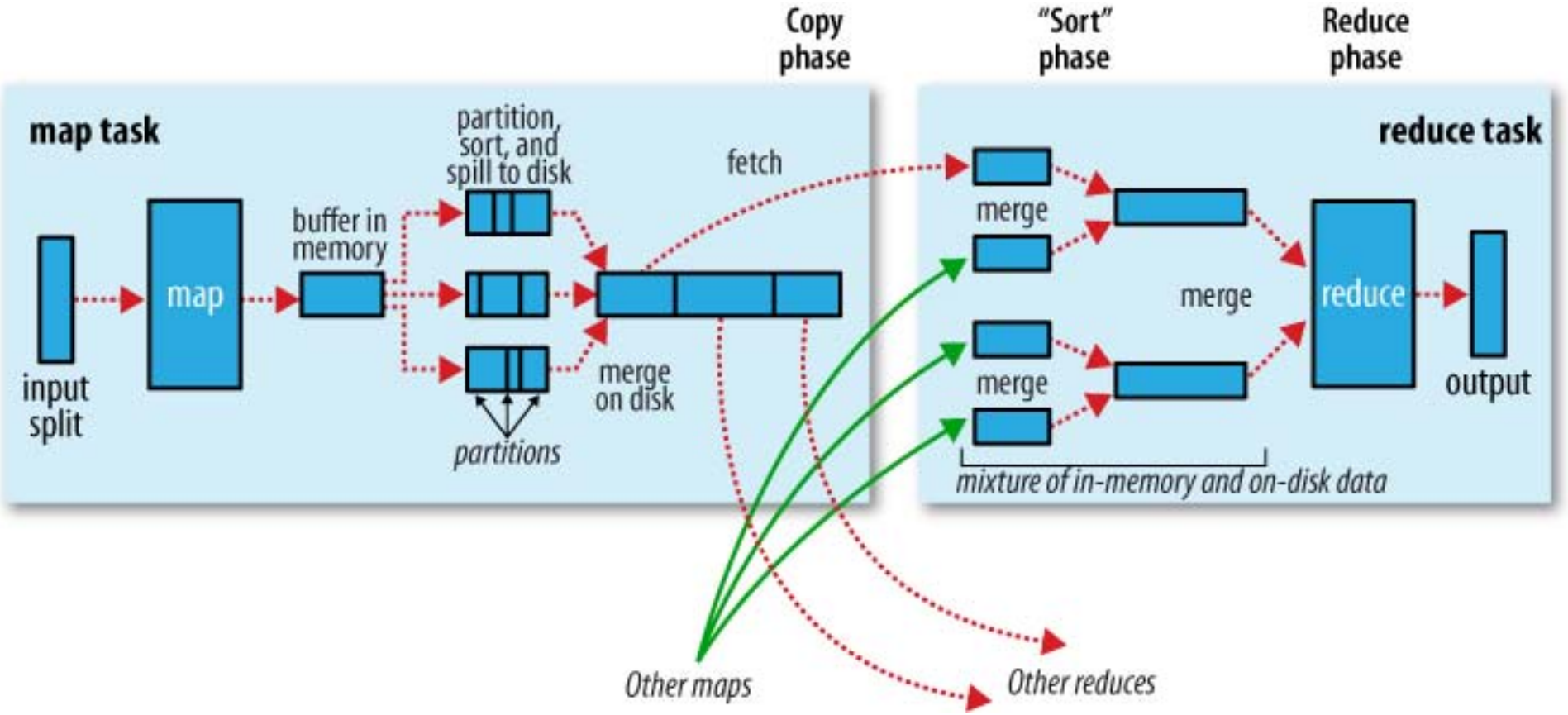
MapReduce: Ausführung

- Parallele Ausführung der Map- und Reduce-Funktionen
  - Lokale Speicherung des partitionierten Map-Outputs; Reducer holen sich ihren Input ab



Quelle: http://www.jiahenglu.net/course/advancedDataManagement/

Shuffle und Sort in MapReduce



https://haritbigdata.files.wordpress.com/2015/07/mapreduce.png

MapReduce: Programmdefinition

- Konfiguration der Map- und Reduce-Tasks
  - *m* = Anzahl der Map-Tasks und *r* = Anzahl der Reduce-Tasks
- Definition der Programm-Logik in Programmiersprache (z.B. Java)
  - ***map* : (key$_{in}$, value$_{in}$) → list (key$_{tmp}$, value$_{tmp}$)**
  - ***reduce* : (key$_{tmp}$, list (value$_{tmp}$)) → list (key$_{out}$, value$_{out}$)**
- Funktionen zur Steuerung des Datenflusses (Optional)
  - *comp*: Vergleichsfunktion zweier Keys
    - Default: Standard-Vergleichsfunktion bei einfachen Datentypen (z.B. String, Int)
  - *part*: Partitionierungsfunktion, d.h. Zuordnung von Keys zu Reduce-Tasks
    - Default: "Hashwert(Key) modulo *r*"
  - *group*: Gruppierung der Keys pro Aufruf von Reduce (Default: Key
  - Combiner-Funktion -> Map-seitiges Reduce
- Definition des MR-Programms unabhängig von Cloud-Umgebung
  - z.B. Anzahl Knoten, Anzahl Prozesse, ...
  - Kenntnis der Umgebung für Optimierung nützlich (z.B. *m*=Anzahl Prozesse

Inhaltsverzeichnis

Termhäufigkeit + MapReduce

- Bestimmung der Häufigkeit eines Terms in einer Dokumentenkollekt
- Beispiel:

Dokumente

d1:"A A B C"
d2:"B D D"
d3:"A B B E"

Termhäufigkeit

A → 3
B → 4
C → 1
D → 2
E → 1

- Parallelisierbar
  - Dokumente können unabhängig voneinander bearbeitet werden
  - Termhäufigkeit = Summe der Termhäufigkeit in Dokumenten
- Map
  - Eingabe: Dokumentenmenge
  - Ausgabe: Für jedes Term-Auftreten Paar (Term, 1)
- Shuffle + Sort
  - Sortierung und Gruppierung nach Key, d.h. Term
- Reduce
  - Zählen der Vorkommen (Term, 1) pro Term

Termhäufigkeit + MapReduce: Beispiel