Data Mining

Einführung

Johannes Zschache
Wintersemester 2019

Abteilung Datenbanken, Universität Leipzig
http://dbs.uni-leipzig.de

Inhaltsverzeichnis

Literatur: Kapitel 1 + 2 aus „Mining of Massive Datasets“ http://www.mmds.org/

Data Mining

- Annahme: Daten enthalten wertvolle Informationen (Wissen)
- Zur Gewinnung des Wissens müssen die Daten gespeichert, aufbereitet und **analysiert** werden

- **Data Mining (DM)** = Extraktion von umsetzbaren Informationen aus (meist sehr großen) Datensätzen

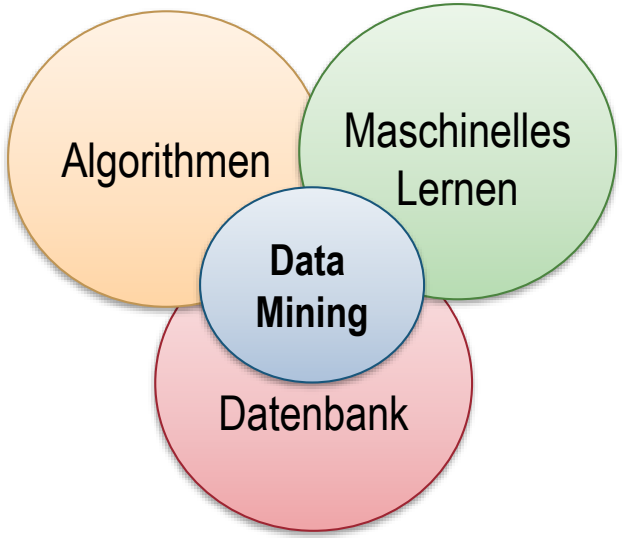
- Ergebnis eines DM-Verfahrens: **Modell** (auch: Muster)
  - **Deskriptiv**: für Menschen verständliche Zusammenfassung
  - **Prädiktiv**: Vorhersage unbekannter Werte aus bekannten Werten
  - Statistisch: Annahme einer Wahrscheinlichkeitsverteilung und Schätzen der Parameter

- Data Mining $\stackrel{?}{=}$ Maschinelles Lernen $\stackrel{?}{=}$ Data Science $\stackrel{?}{=}$ Big Data
  - Unterschiedliche Verwendung
  - Abgrenzung schwierig

Hintergrund

- „Welt der Datenbanken“: DM bezieht sich auf die Anwendung effizienter **Algorithmen** zur Erkennung von Mustern in großen Datenmengen
- „Welt der Statistik“: DM bedeutet *auch* **Inferenz** eines Modells
- Unterscheidung zwischen Algorithmus und Inferenz, z.B.
  - Berechnung eines Mittelwerts ist Algorithmus: $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$
  - Inferenz ermöglicht eine Bewertung des Ergebnisses: z.B. über $\widehat{Var}(\bar{x})$

- **Schwerpunkt dieser Vorlesung:**
  skalierbare Algorithmen
  - Parallelisierung oft notwendig bei Datenmengen/Berechnungen, die nicht in den Hauptspeicher eines Rechners passen
  - Verteilte Verarbeitung über z.B. **MapReduce** oder **Datenströme**



Abgrenzung zu anderen Vorlesungen

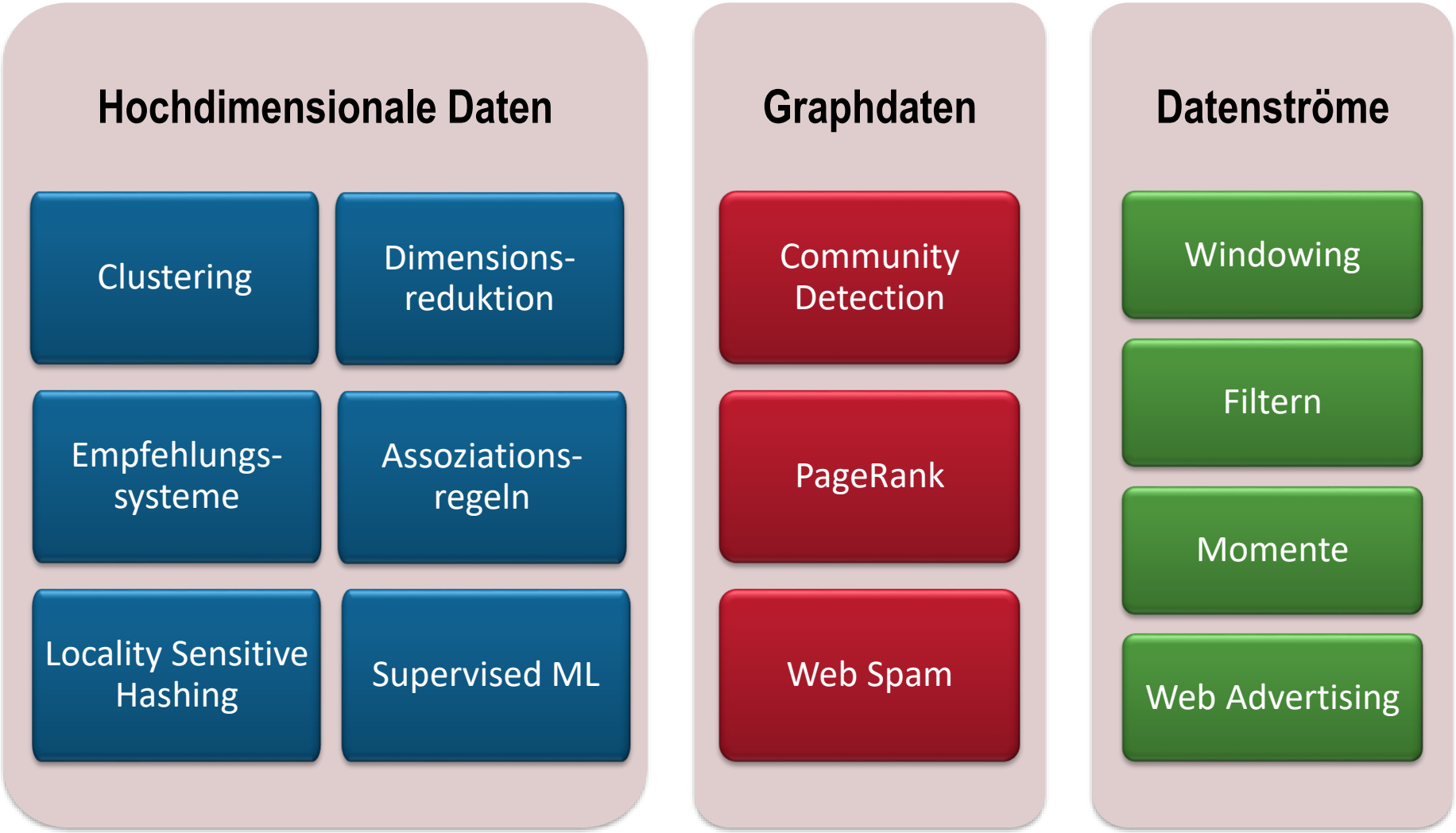
- **Schwerpunkt dieser Vorlesung: skalierbare Algorithmen**
- Data Mining ist letztes Kapitel der Vorlesung „*Data Warehousing*“
- *Cloud Data Management*: Ausführliche Behandlung von Technologien für verteilte Datenhaltung und -verarbeitung
- **Keine Datenbanken** (DBS 1+2, *Mehrrechner-Datenbanken, NoSQL*)
- **Keine Inferenz** (*Statistisches Lernen*)
- Betonung auf **Algorithmen**: Fortsetzung der Vorlesungen zu „*Algorithmen und Datenstrukturen*“
- Fokus auf das **Gebiet der Datenanalyse**

Lernziele

- **Vorlesung:** Kenntnis skalierbarer **Algorithmen** zur **Analyse von Daten**
  - Nachvollziehen der **Funktionsweise** der Algorithmen
  - **Anwendung** der Algorithmen an kleinen Beispieldaten
  - Beurteilung der **Anwendbarkeit** von Algorithmen (Komplexität, Engpässe)
  - **Vergleich** verschiedener Algorithmen

- **Praktikum**: Umsetzung der Algorithmen in Java/Spark
  - Anwendung bereits implementierter Algorithmen: K-Means, SVD, Vorhersage
  - Eigene Implementierung: Collaborative Filtering, A-Priori, LSH, PageRank, Datenströme

Übersicht


Hochdimensionale Daten
Clustering
Dimensions-reduktion
Empfehlungs-systeme
Assoziations-regeln
Locality Sensitive Hashing
Supervised ML
Graphdaten
Community Detection
PageRank
Web Spam
Datenströme
Windowing
Filtern
Momente
Web Advertising

Organisation

- Vorlesungstermine
  - Donnerstags, 9:15-10:45 Uhr, HS 6
  - Freitags, 9:15-10:45 Uhr, HS 5
  - 17.10.2019 – **6.12.2019** (31.10. entfällt)

- Anmeldung: AlmaWeb

- Webseite mit Folien: https://dbs.uni-leipzig.de/stud/2019ws/dm

- **Klausur: 13.12.2019, 9:30 Uhr, HS 9**

- Inhalt: „Mining of Massive Datasets“ von J. Leskovec, A. Rajaraman und J. Ullman, Stanford University
- Buchkapitel, Originalfolien und Videos: http://www.mmds.org/

Lehrveranstaltungen WS 2019

- Lehrveranstaltungen: https://dbs.uni-leipzig.de/stud
  - Datenbanksysteme 1 (DBS1)
  - Vorlesung: Mehrrechner-Datenbanksysteme
  - Vorlesung: **Data Mining**
  - Praktikum: Data Warehousing und **Data Mining**
  - Seminar: Trends in Machine Learning and Data Analytics
  - Bachelor-/Masterseminar (Vortrag über laufende Bachelor-/Masterarbeit)

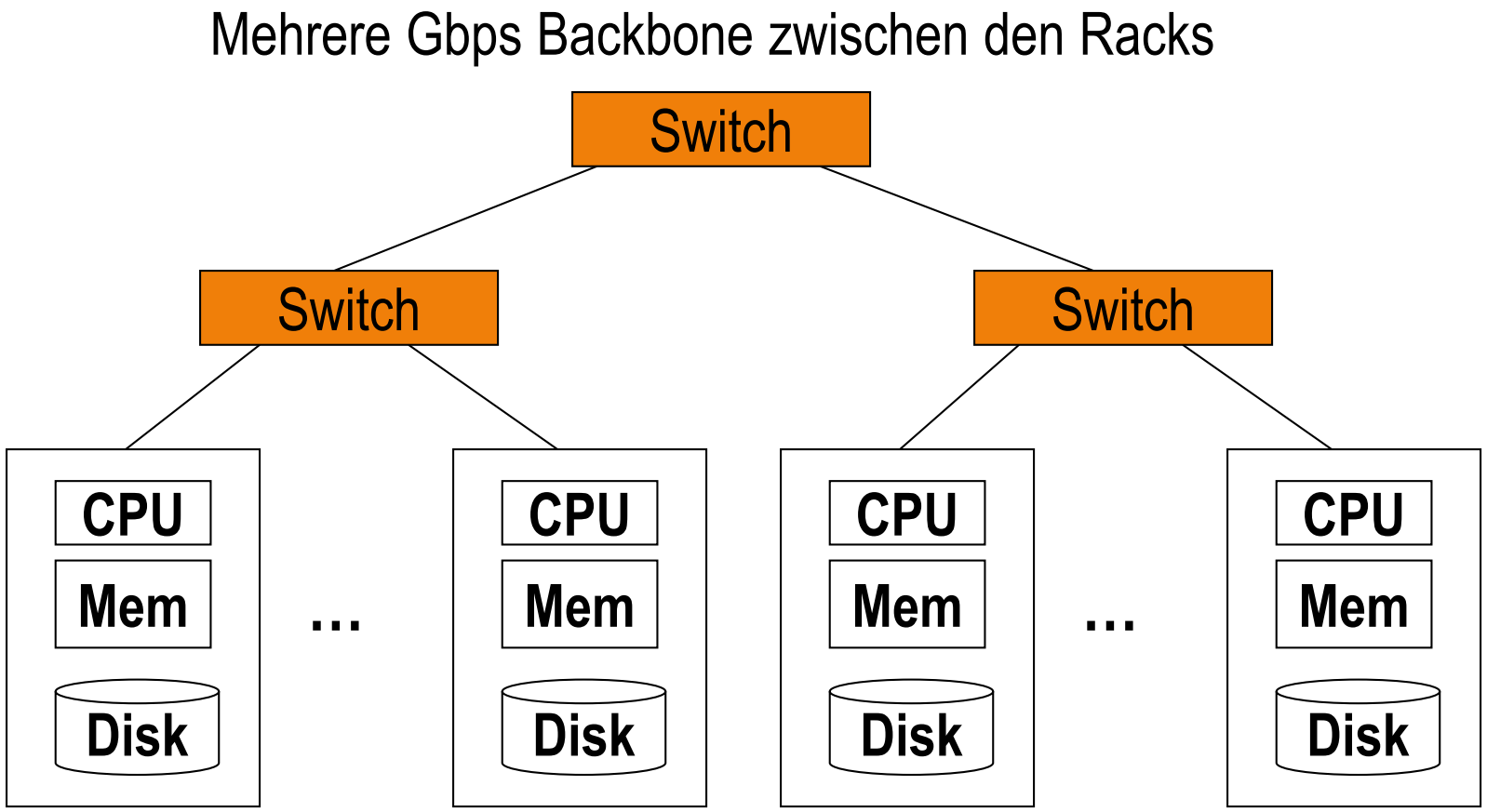
- Verwendung der Data-Mining-Vorlesung:
  - Bachelor-Modul „Realisierung von Informationssystemen“ (5LP, zwei Vorlesungen)
  - Master-Modul „Moderne Datenbanktechnologien”
    - (kleines) Kernmodul (5LP, zwei Vorlesungen)
    - (großes) Vertiefungsmodul (10LP, zwei Vorlesungen + Praktikum/Seminar)

Inhaltsverzeichnis

Literatur: Kapitel 1 + 2 aus „Mining of Massive Datasets“ http://www.mmds.org/

Rechencluster

- Single-Node-Architektur ist begrenzt bzgl. Speicher und Rechenzeit
- Cluster-Architektur:



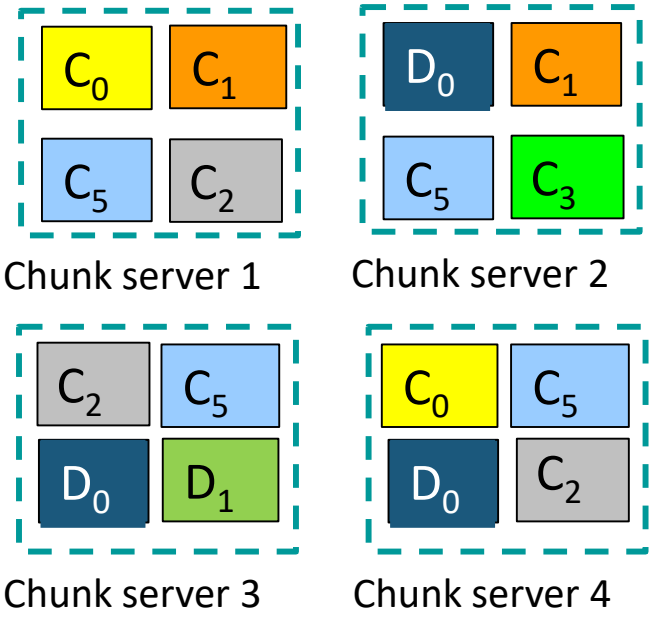
Large-Scale Computing mit Hadoop

- Herausforderungen:
  - Das Kopieren von Daten über ein Netzwerk braucht Zeit
  - Ausfall einzelner Rechner
    - Ein Rechner kann 3 Jahre (1000 Tage) ohne Unterbrechung laufen
    - Bei 1.000 Rechnern fällt 1 Rechner pro Tag aus
    - Bei 1M Rechnern versagen täglich 1000 Rechner

- Idee:
  - Bringe die Berechnungen zu den Daten
  - Repliziertes Speichern der Daten für erhöhte Zuverlässigkeit

- Hadoop adressiert diese Herausforderungen
  - Verteiltes Dateisystem: HDFS
  - Programmiermodell: MapReduce

Verteiltes Dateisystem



- Stellt **globalen Namespace** bereit
- Typisches Nutzung:
  - Riesige Dateien (100 GB bis einige TB)
  - Daten werden selten direkt aktualisiert
  - Lesen und Hinzufügen sind üblich
- **Chunk-Server**
  - Eine Datei wird in zusammenhängende Teile (Chunks, 16-64 MB) aufgeteilt
  - Jeder Chunk wird repliziert (2-3x) auf verschiedenen Rechnern/Racks gespeichert
- **Master/Name-Server**
  - Speichert Metadaten darüber, wo Dateien gespeichert sind
  - Sollte auch repliziert werden
- **Client-Bibliothek für den Dateizugriff**
  - Kommunikation mit Master um Chunk-Server zu finden
  - Stellt eine direkte Verbindung zu Chunk-Servern her, um auf Daten zuzugreifen

MapReduce

- MapReduce ist ein Programmiermodell für:
  - Einfache parallele Verarbeitung
  - Unsichtbare Verwaltung von Hard- und Softwarefehlern
- Implementierungen: Google, Hadoop, Spark, Flink
- 3 Schritte
  1. **Map**:
     - Ein Mapper pro Chunk
     - Anwendung *einer benutzerdefinierten* Funktion auf Chunk
     - Die Ausgabe der Map-Funktion ist eine Menge von Schlüssel-Wert-Paaren
  2. Gruppieren nach Schlüssel: Das System sortiert alle Schlüssel-Wert-Paare nach Schlüssel und gibt Schlüssel-Wertelisten-Paare aus
  3. **Reduce**:
     - Ein Reducer pro Schlüssel
     - Anwendung *einer benutzerdefinierten* Funktion auf die Werteliste

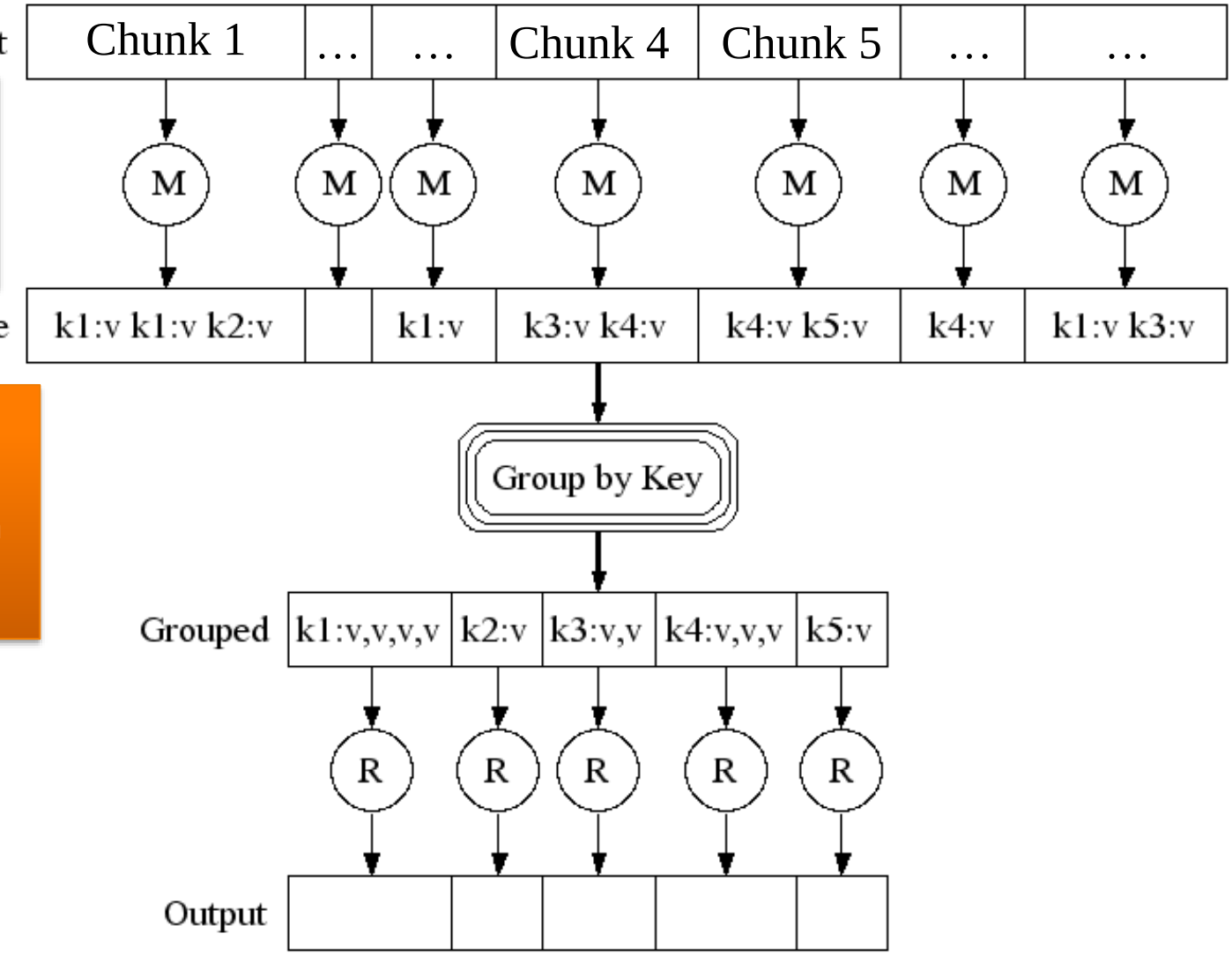
Schreiben einer Map-Funktion und einer Reduce-Funktion

MapReduce: Diagramm

MAP:
Liest die Eingabe und erzeugt eine Menge von Schlüssel-Wert-Paaren

Gruppierung:
Sammle alle Paare mit dem selben Schlüssel

Reduce:
Nimm alle Werte eines Schlüssels und berechne Ausgabe

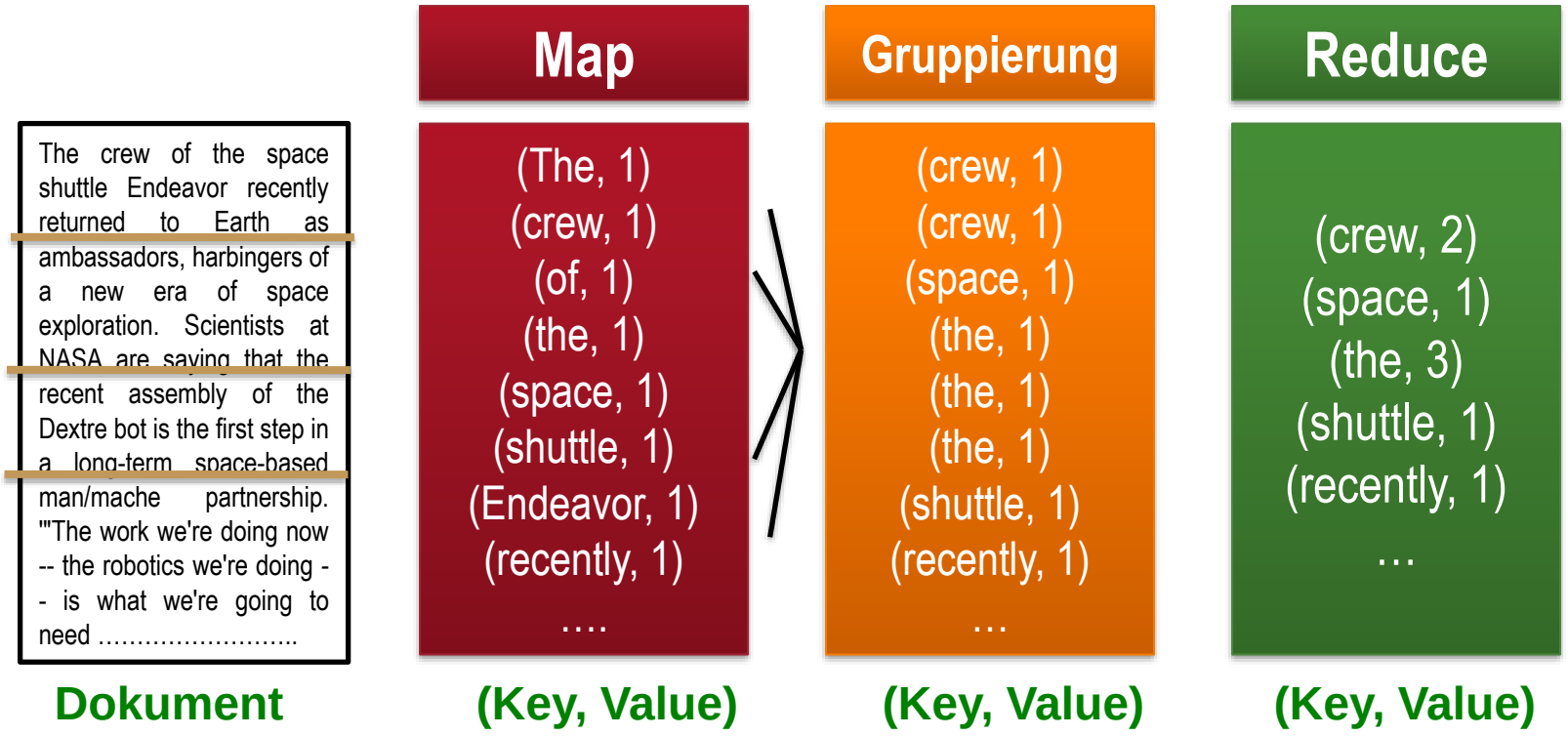


Beispiel: Häufigkeiten von Wörtern

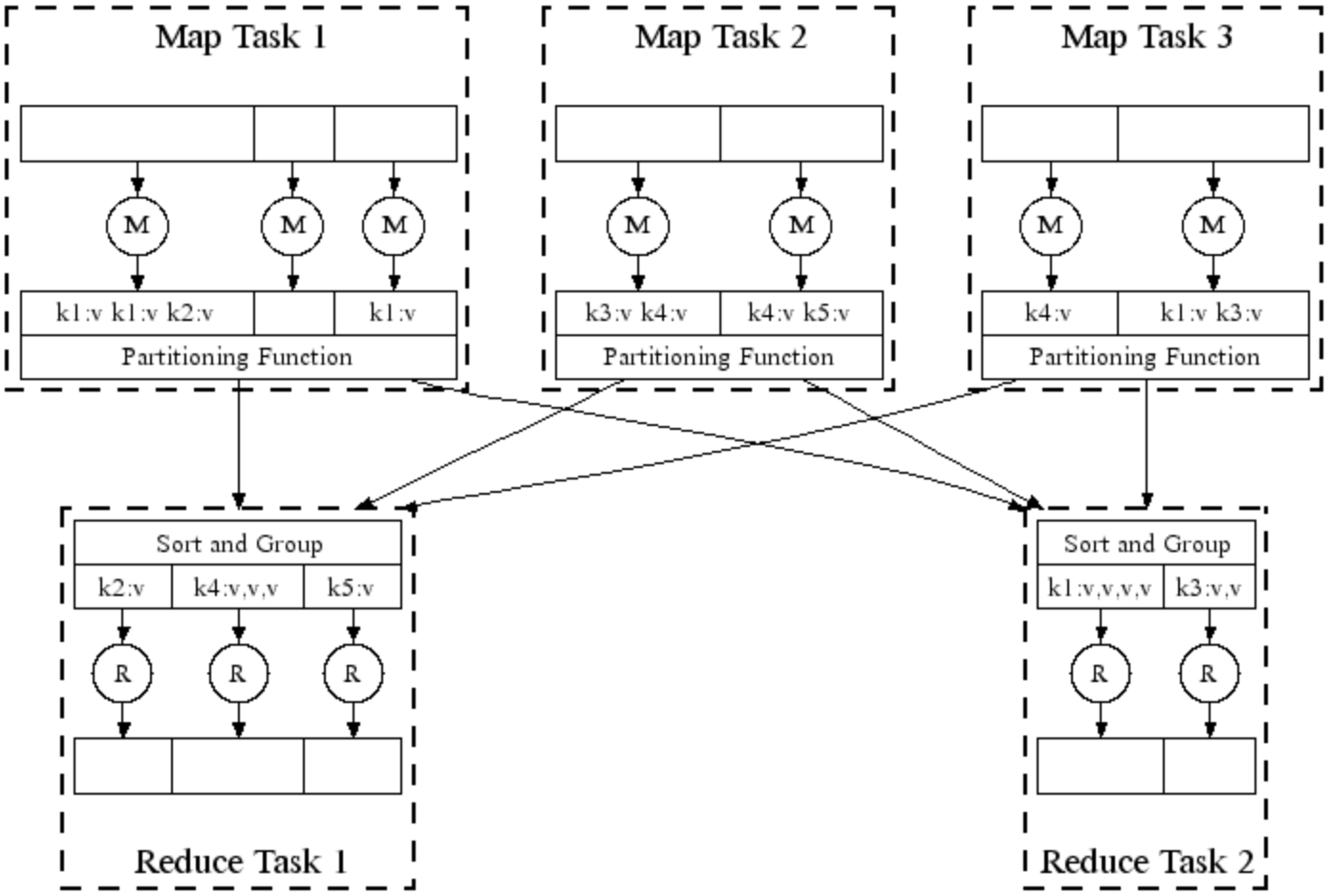
```
map(input):
// input: text of the document
   for each word w in input:
       emit(w, 1)
```

```
reduce(key, values):
// key: a word; value: an iterator over counts
       result = 0
       for each count v in values:
             result += v
       emit(key, result)
```

Beispiel: Häufigkeiten von Wörtern



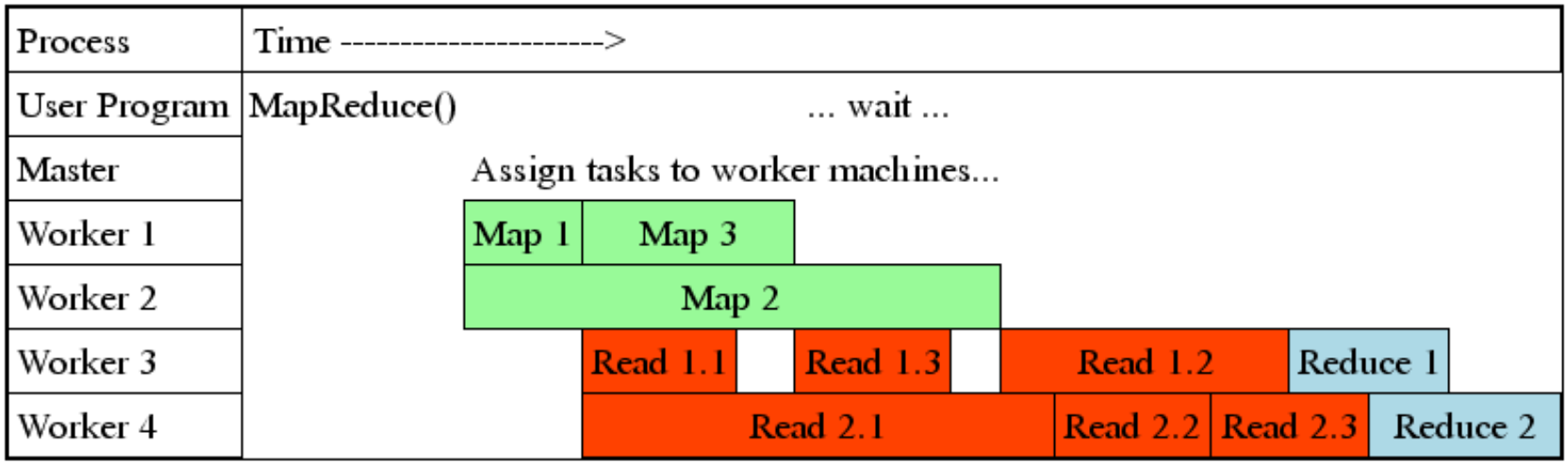
MapReduce: Verteilte Ausführung



Alle Mapper und Reducer werden gruppiert in Map-/Reduce-Tasks verteilt auf mehreren Rechnern ausgeführt

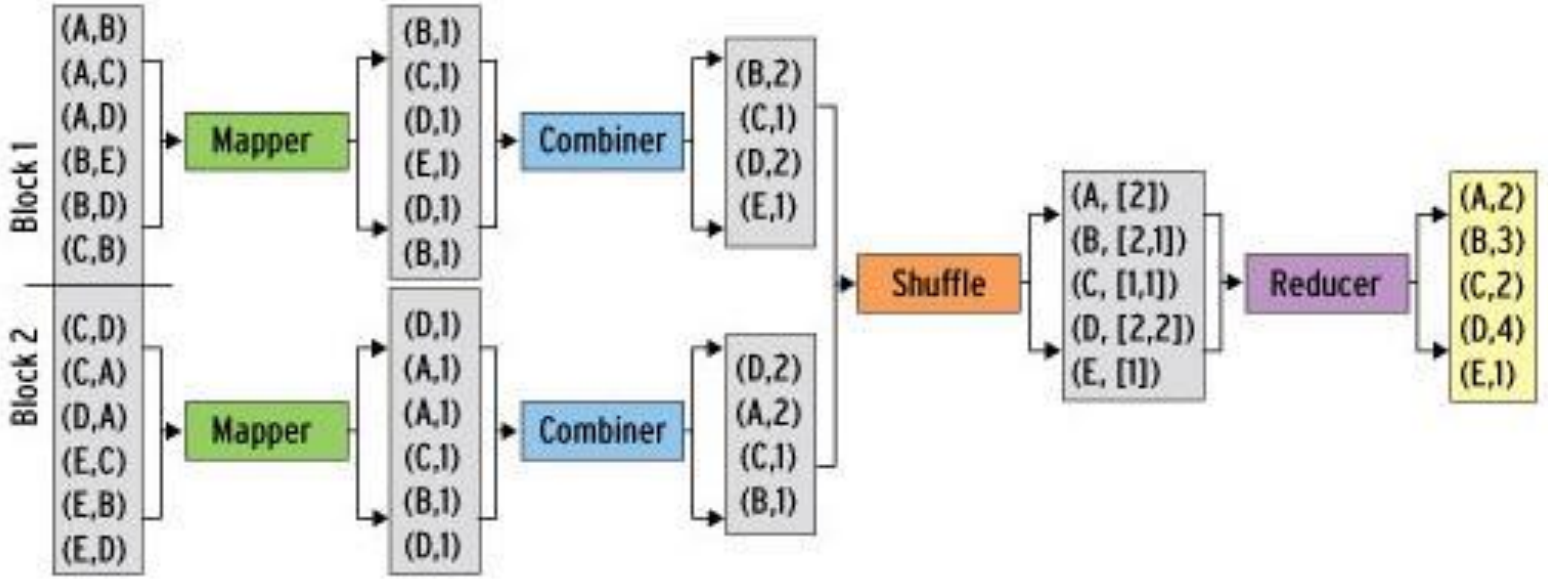
MapReduce: System

- MapReduce-Implementierung kümmert sich um
  - Planung der Programmausführung über mehrere Rechner hinweg: Mapper sollten „in der Nähe“ des physischen Speicherorts der Daten ausgeführt werden
  - Verwaltung der Kommunikation zwischen den Rechnern: Zwischenergebnisse der Mapper werden im lokalen Dateisystem (auf Festplatte) der ausführenden Rechner gespeichert → Informationen zu Speicherort an Reducer gesendet
  - Durchführung des Gruppierungsschritts (Engpass in der Praxis)
  - Behandlung von Serverausfällen



MapReduce: Combiners

- Ein Map-Task erzeugt oft viele Paare des gleichen Schlüssels
- Weniger Kommunikation zwischen Rechnern, wenn Werte im Map-Task durch eine Combiner-Funktion vorab aggregiert werden
  - Combiner-Funktion ist normalerweise identisch zur Reduce-Funktion
  - Funktioniert nur, wenn die Reduce-Funktion kommutativ und assoziativ ist
- Beispiel: Häufigkeiten von Wörtern



Inhaltsverzeichnis

- **Einführung**
  - **Data Mining**
  - **Übersicht zur Vorlesung**
  - **Organisation**
- **Datenverarbeitung mit MapReduce**
  - **Typische Algorithmen**
  - **Kommunikationskosten & Komplexität**
- **Übungen**

Literatur: Kapitel 1 + 2 aus „Mining of Massive Datasets“ http://www.mmds.org/

Algorithmen mit MapReduce

- MapReduce ist ineffizient für Probleme, die wahlfreien Zugriff (random access) auf Daten erfordern, z.B. OLTP
- MapReduce ist ideal für
  - Probleme, die einen sequentiellen Datenzugriff erfordern
  - Große Datenmengen mit wenig Aktualisierungen
  - Große Batch-Jobs (nicht interaktiv)
  - Analytische Anfragen auf großen Datenmengen
- Beispiele:
  - Operationen der Relationalen Algebra:
    - Selektion
    - Projektion
    - Gruppierung & Aggregation
    - Natural Join
  - Matrix-Vektor-Produkt
  - Matrixmultiplikation

MapReduce: Relationale Algebra

- **Selektion**: Einträge der Relation R mit Eigenschaft F ($\sigma_F(R)$)
  - Eingabedaten: Elemente (Zeilen) e der Relation R
  - Map: Prüfe auf Eigenschaft F und erzeuge Paar (e,e), falls F für e erfüllt ist

- **Projektion**: Auswahl der Attribute A von Relation R ($\pi_A(R)$)
  - Eingabedaten: Elemente (Zeilen) e der Relation R
  - Map: Erzeuge e‘ durch Auswahl der Attribute A von e und gib Paar (e‘, e‘) aus
  - Falls Duplikate eliminiert werden sollen (distinct): Reducer für Schlüsselwert e‘ erhält Paare der Form (e‘, [e‘, e‘, …, e‘]) aus GroupByKey-Schritt und gibt (e‘, e‘) aus

- **Gruppierung** über Attribut A und *Aggregation* durch Ausführung der Funktion $\theta()$ auf dem Attribut B ($\gamma_{A,\theta(B)}(R)$):
  - Eingabedaten: Elemente (Zeilen) aus Attributen a,b,c… der Relation R
  - Map: Erzeuge Paare (a,b)
  - Reducer für Schlüsselwert a erhält Paare der Form (a, $[\mathrm{b}_1, \mathrm{b}_2, \ldots, \mathrm{b}_n]$) aus GroupByKey-Schritt und berechnet (a, $\theta(\mathrm{b}_1, \mathrm{b}_2, \ldots, \mathrm{b}_n)$)

MapReduce: Join

$R(A,B) \bowtie S(B,C)$

A	B
a_1	b_1
a_2	b_1
a_3	b_2
a_4	b_3

R

$\bowtie$

B	C
b_2	c_1
b_2	c_2
b_3	c_3

S

$=$

A	B	C
a_3	b_2	c_1
a_3	b_2	c_2
a_4	b_3	c_3

- Map: Alle Elemente R(a,b) auf (b,(a,"R")) und S(b,c) auf (b,(c,"S"))
- Reducer für Schlüsselwert b:
  - Liste von Paaren [(a_1,"R"), (a_2,"R"), …, (c_1,"S"), (c_2,"S"), (c_2,"S"), (c_3,"S") …]
  - Aufteilung in Paare der Form (a_i,"R") und Paare der Form (c_j,"S")
  - Erzeuge alle Kombinationen der Paare (a_i,"R") mit allen Paaren (c_j,"S")
  - Ergebnis: Liste von Tupeln (a_i,b,c_j)

MapReduce: Matrix-Vektor-Produkt

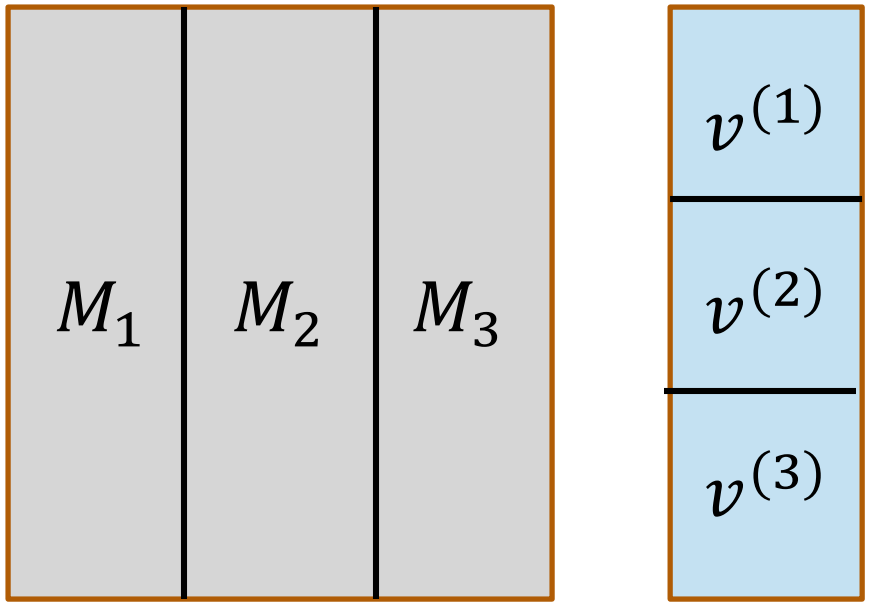
- $n\ x\ n$ - Matrix $\boldsymbol{M} = \left(m_{ij}\right)_{i,j=1,\ldots,n}$ und Vektor $\boldsymbol{v} = \left(v_j\right)_{j=1,\ldots,n}$
- Produkt $\boldsymbol{x} = \boldsymbol{M} \cdot \boldsymbol{v}$ mit $\boldsymbol{x} = (x_i)_{i=1,\ldots,n}$ und

$$x_i = \sum_{j=1}^{n} m_{ij} v_j$$

- Annahme: Vektor $\boldsymbol{v}$ liegt im Hauptspeicher jedes Map-Rechners
- Eingabe: Elemente m_{ij} der Matrix $\boldsymbol{M}$ in der Form (i, j, m_{ij})
- Map: Elemente (i, j, m_{ij}) auf $(i, m_{ij} v_j)$
- Reducer für Schlüsselwert i bekommt Liste $[m_{i1} v_1, m_{i2} v_2, \ldots, m_{in} v_n]$ und berechnet $\left(i, \sum_j m_{ij} v_j\right)$ also den Eintrag x_i von $\boldsymbol{x}$

MapReduce: Matrix-Vektor-Produkt

- Falls Vektor $\boldsymbol{v}$ nicht in den Hauptspeicher eines Map-Rechners passt
- Aufteilung der Matrix $\boldsymbol{M}$ in Streifen, z.B:



- Jeder Map-Task ist nur für einen Streifen M_k zuständig und benötigt auch nur den dazugehörigen Teil $v^{(k)}$ von $\boldsymbol{v}$
- Berechnung sind analog, nur das ein Map-Task nur die Elemente (i, j, m_{ij}) des ihm zugeordneten Streifens erhält

MapReduce: Matrix-Produkt

- $m\ x\ n$ - Matrix $\boldsymbol{M} = \left(m_{ij}\right)_{i,j}$ und $n\ x\ p$ - Matrix $\boldsymbol{N} = \left(n_{jk}\right)_{i,j}$
- Produkt $\boldsymbol{P} = \boldsymbol{M} \cdot \boldsymbol{N}$ mit $\boldsymbol{P} = (p_{ik})_{i,k}$ und

$$p_{ik} = \sum_{j=1}^{n} m_{ij} n_{jk}$$

- Map:
  - Elemente (i, j, m_{ij}) auf $((i, k), ("M", j, m_{ij}))$ für alle $k = 1, \ldots, p$
  - Elemente (j, k, n_{jk}) auf $((i, k), ("N", j, n_{jk}))$ für alle $i = 1, \ldots, m$
- Reducer für Schlüsselwert (i, k) bekommt Liste mit allen Elementen, die für die Berechnung von p_{ik} notwendig sind

Inhaltsverzeichnis

- **Einführung**
  - **Data Mining**
  - **Übersicht zur Vorlesung**
  - **Organisation**
- **Datenverarbeitung mit MapReduce**
  - **Typische Algorithmen**
  - **Kommunikationskosten & Komplexität**
- **Übungen**

Literatur: Kapitel 1 + 2 aus „Mining of Massive Datasets“ http://www.mmds.org/

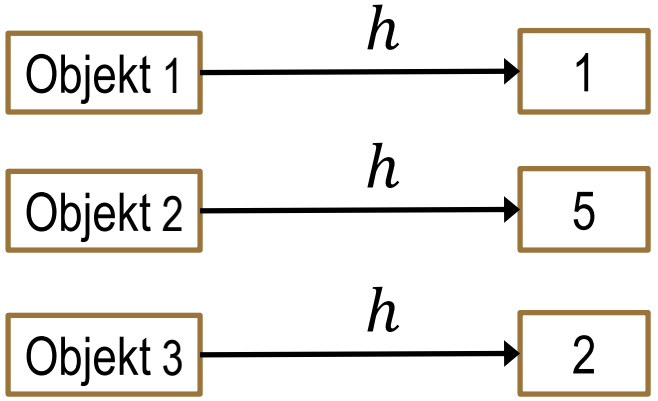
Kommunikationskosten

- Engpass in MapReduce ist Kommunikation zwischen den Tasks
- Berechnungszeit der Tasks „weniger wichtig“, da
  - Oft linear in der Größe der Eingabe
  - Berechnungen im Hauptspeicher sind WESENTLICH SCHNELLER als Kommunikation mit Festplatte bzw. über Netzwerk

- **Replikationsrate r:** durchschnittliche Anzahl der Schlüssel-Wert-Paare, die pro Eingabe durch Mapper erzeugt werden

- **Beispiel: Matrix-Produkt**
  - $r = m \cdot \frac{|N|}{|N|+|M|} + p \cdot \frac{|M|}{|N|+|M|}$
  - Reduktion über Verwendung von **Hashfunktionen**

Einschub: Hashfunktion



- „Bucket“-Anzahl $B \in \mathbb{N}$
- Hashfunktion h: Objekt $\rightarrow \{0, \ldots, B-1\}$
- *Randomisierend*:
  - Gleichmäßige Aufteilung der Objekte über die Buckets
  - B ist kleiner/größer als die Menge der eindeutigen Objekte
- Beispiele
  - N modulo p, wobei p eine Primzahl und N eine natürliche Zahl
  - Zeichenketten: Summe über deren ASCII/Unicode-Repräsentation
  - Tupel: Summe über Hash-Werte der Elemente

Alternative Berechnung Matrix-Produkt

- Hashfunktionen:
  - $h: \{1, \ldots, m\} \rightarrow \{1, \ldots, b\}$ mit $b < m$
  - $g: \{1, \ldots, p\} \rightarrow \{1, \ldots, c\}$ mit c $< p$
- Ein Reducer für jedes Paar (v, w) mit $\mathrm{v} \in \{1, \ldots, b\}$ und $\mathrm{w} \in \{1, \ldots, c\}$
- Map:
  - Elemente (i, j, m_{ij}) auf $((h(i), k), ("M", i, j, m_{ij}))$ für alle $k = 1, \ldots, c$
  - Elemente (j, k, n_{jk}) auf $((i, g(k)), ("N", j, k, n_{jk}))$ für alle $i = 1, \ldots, b$
- Reducer für Schlüsselwert (v, w) bekommt Liste mit allen Elementen, die für die Berechnung aller p_{ik} mit $h(i) = v$ und $\mathrm{g}(k) = w$ nötig sind
- ***Reduzierung der Kommunikationskosten auf***

$$\boldsymbol{r = b \cdot \frac{|N|}{|N| + |M|} + c \cdot \frac{|M|}{|N| + |M|}}$$

- **Aber**: Erhöhung der Rechenzeit der Reducer

Komplexität von MapReduce

- Verhältnis zwischen Kommunikationskosten und Rechenzeit (Reducer)
- Rechenzeit wird minimiert durch
  - Parallelisierung (über mehrere Kerne/CPUs)
  - Berechnung im Hauptspeicher
- Dazu sollte Eingabe der Reducer nicht zu groß sein (z.B. 2 - 32GB passen in Hauptspeicher eines Rechners)

- **Reducer-Size q**: Obere Schranke der Anzahl der Werte für einen Schlüssel aus dem Map-Schritt
- Oft: ***Inverse Beziehung*** zwischen q und Replikationsrate r

- Beispiel: Matrix-Produkt
  - Ohne Hashfunktionen: $q = 2n$
  - Mit Hashfunktionen: $q = n \cdot \left(\frac{m}{b} + \frac{p}{c}\right)$

Beispiel: Similarity Join

- Berechnung der Ähnlichkeiten zwischen 1 Million Bilder (jeweils 1 MB)
- **Naiver** Ansatz:
  - Map: jedes Bild P_i auf die Schlüssel $\{i, j\}$ mit $j \in \{1, \ldots, 10^6\}, j \neq i$
  - Reduce: zwei Bilder P_i und P_j pro Reducer, d.h. $q = 2$
  - Da $r = 999.999$, ca. 1 Exabyte Kommunikation, 2,5 Jahre bei 100 Gbps Ethernet

- **Besser**: Gruppierung der Bilder in g Gruppen
  - Map: jedes Bild P_i auf die Schlüssel $\{u, v\}$ mit $v \in \{1, \ldots, g\}, u \neq v$ und u ist die Gruppe von $P_i \rightarrow r = g - 1$
  - Reducer des Schlüssels $\{u, v\}$ erhällt alle Bilder der Gruppen u und v
  - $q = 2\frac{10^6}{g} \rightarrow$ dem Reducer genügt ein 2GB-Hauptspeicher, falls $\mathrm{g} = 1000$
  - Kommunikation benötigt „nur noch“ 22 Stunden

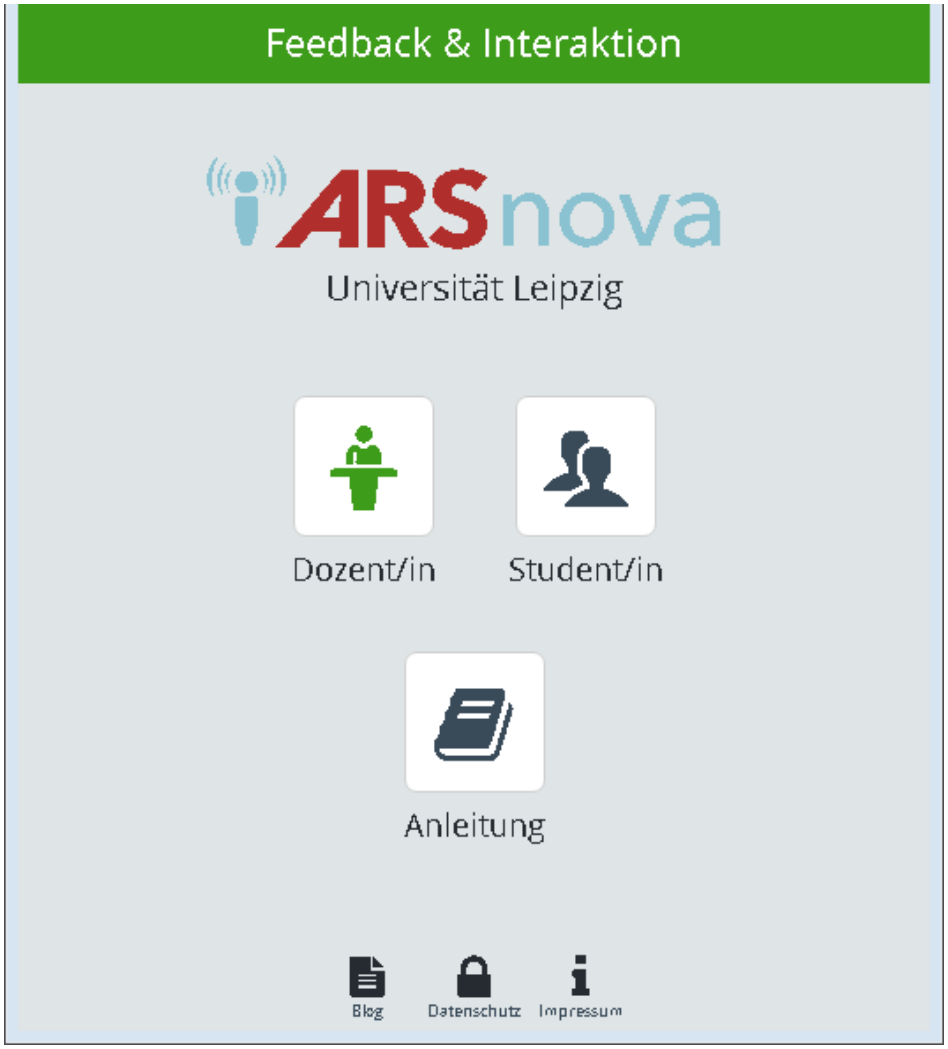
- Effizienteres Verfahren über *Locality-sensitive Hashing*

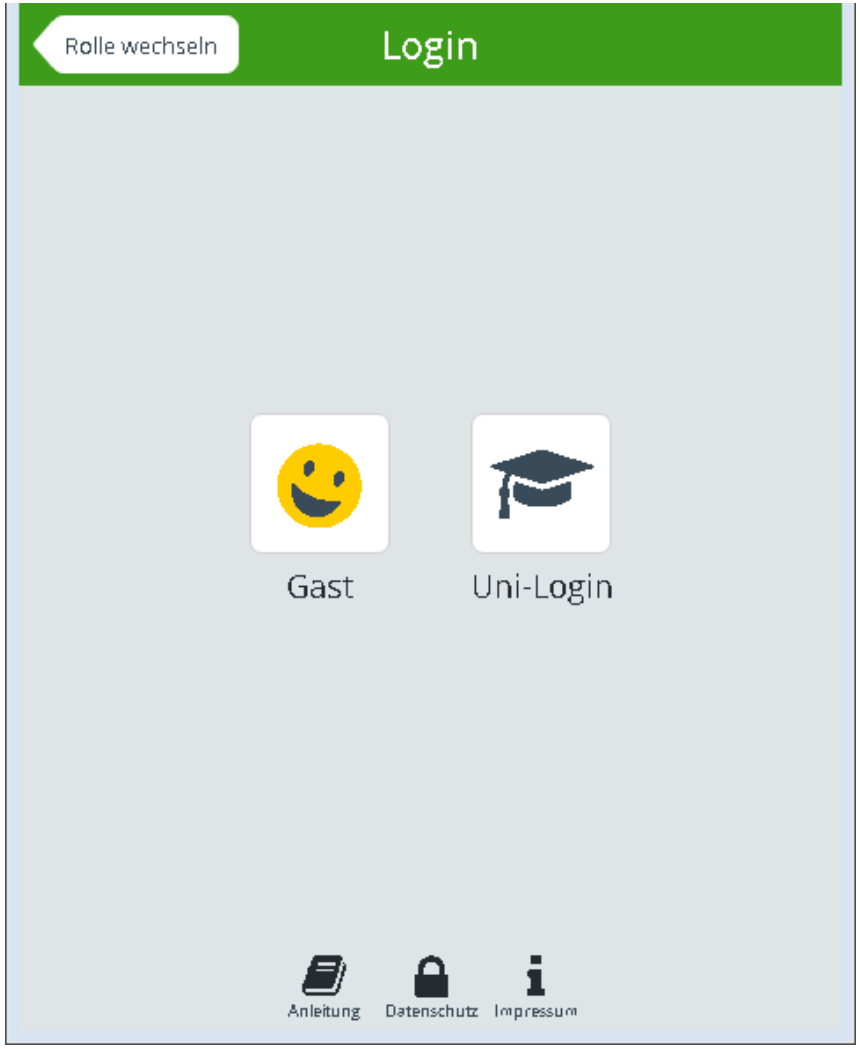
Inhaltsverzeichnis

- **Einführung**
  - **Data Mining**
  - **Übersicht zur Vorlesung**
  - **Organisation**
- **Datenverarbeitung mit MapReduce**
  - **Typische Algorithmen**
  - **Kommunikationskosten & Komplexität**
- **Übungen**

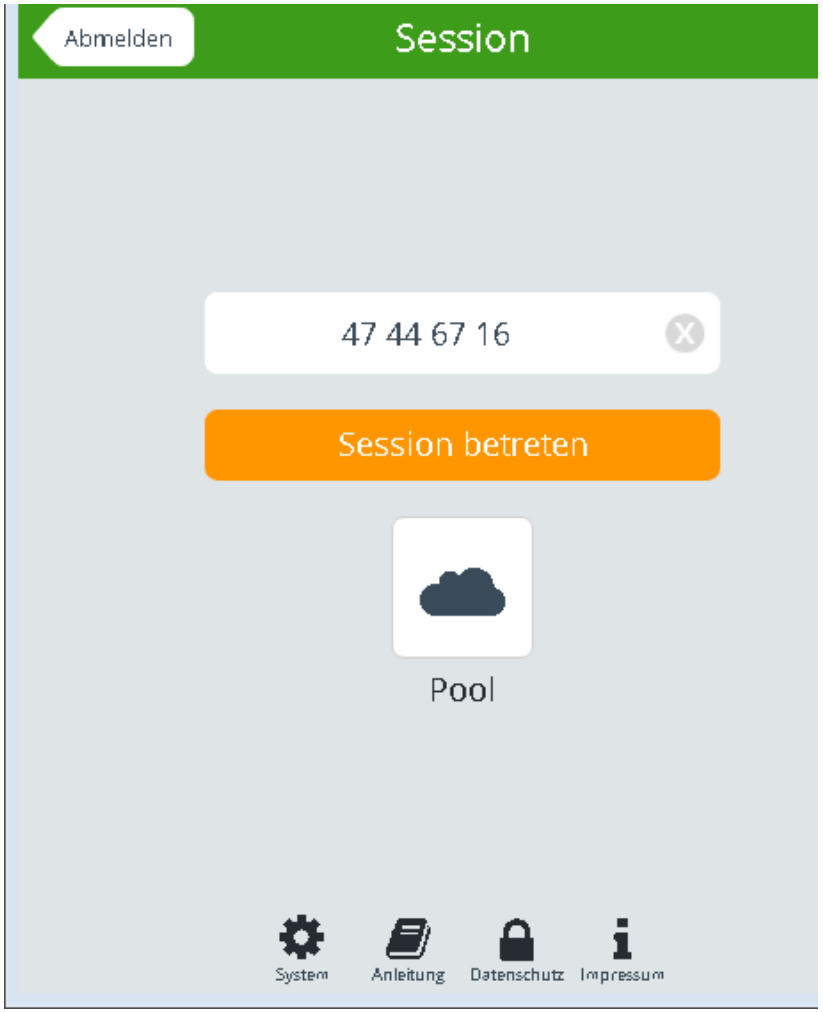
Literatur: Kapitel 1 + 2 aus „Mining of Massive Datasets“ http://www.mmds.org/

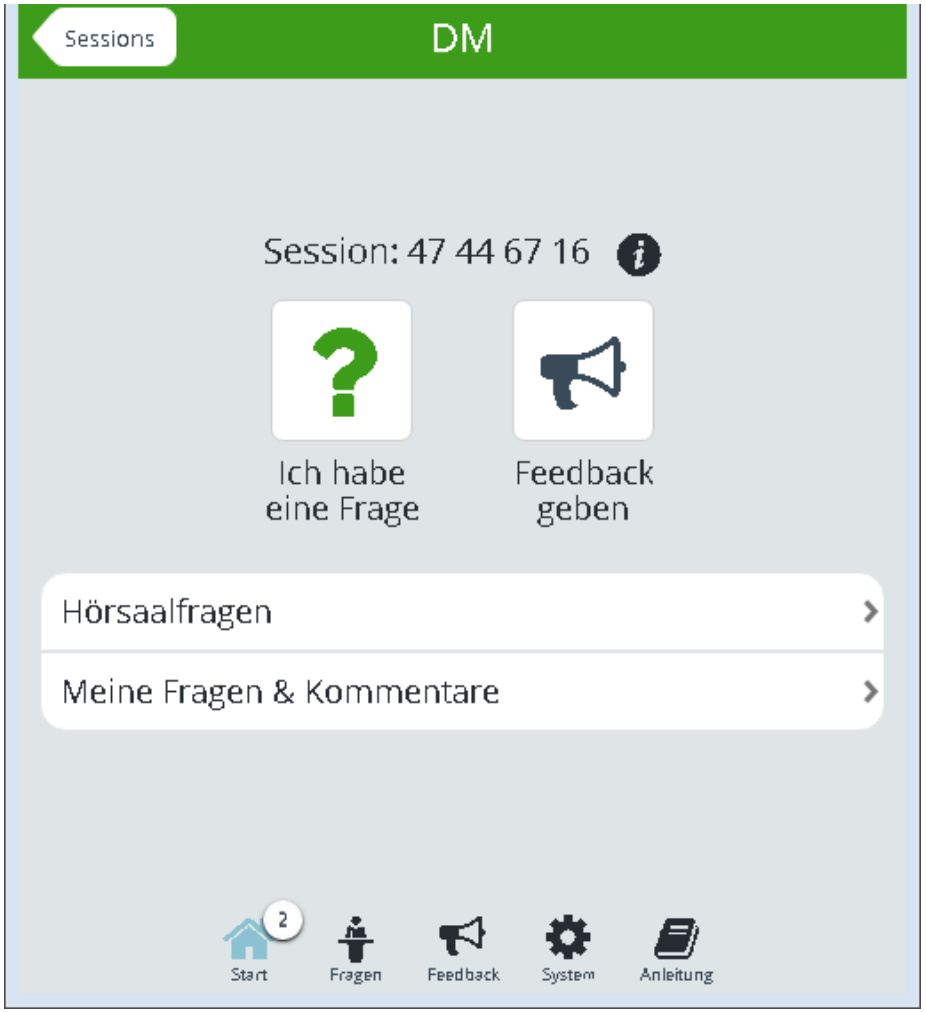
ARSnova: arsnova.rz.uni-leipzig.de 47 44 67 16



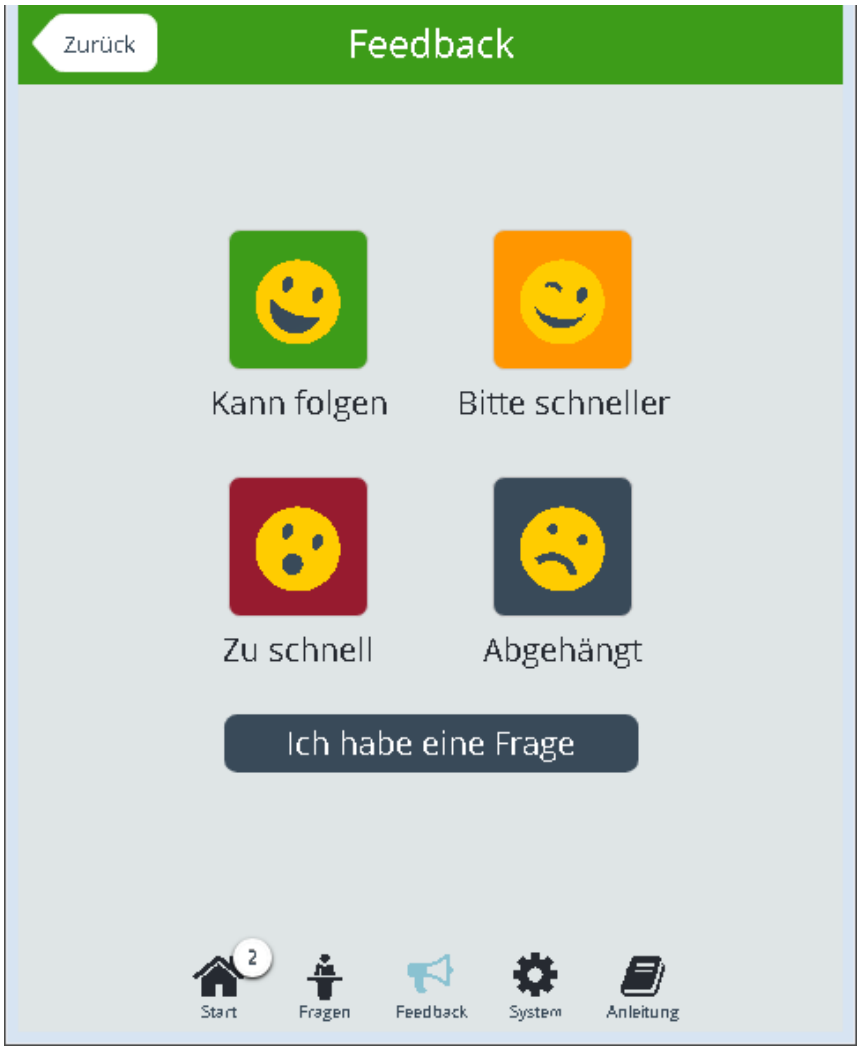


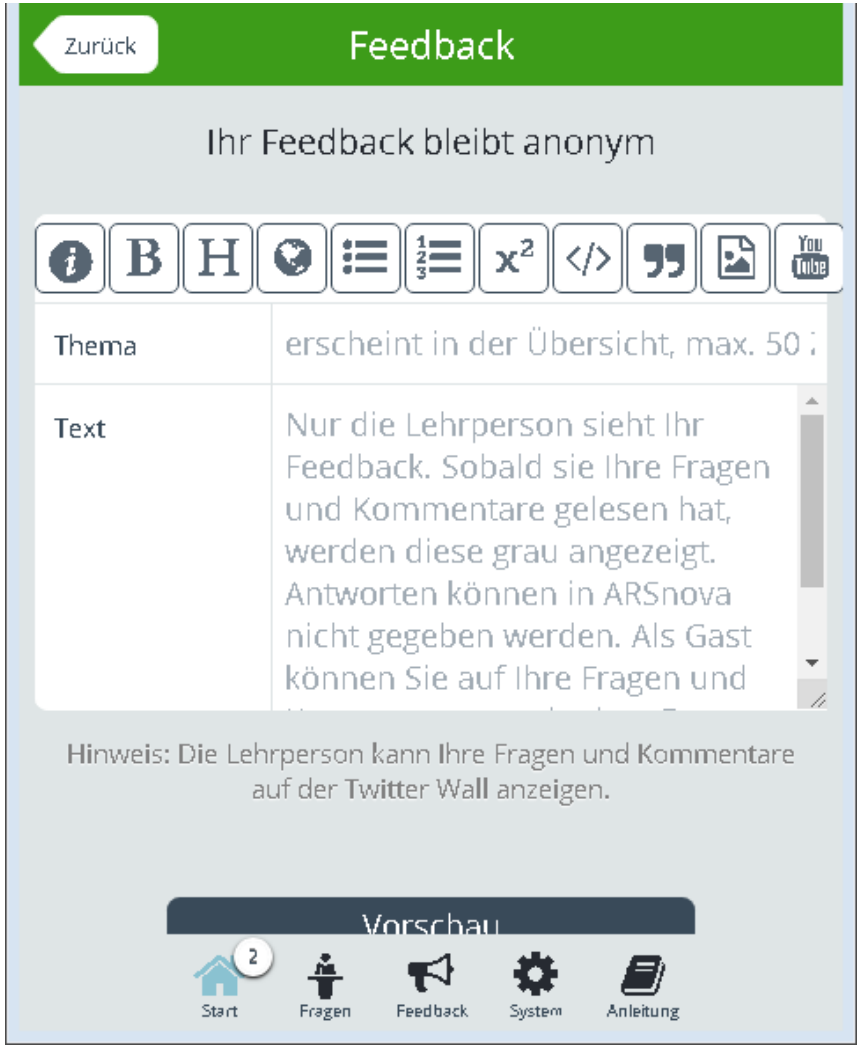
ARSnova: arsnova.rz.uni-leipzig.de 47 44 67 16





ARSnova: arsnova.rz.uni-leipzig.de 47 44 67 16





Übung 1: Lösung

Stellen Sie sich vor, Sie sollen den MapReduce-Algorithmus für die Häufigkeiten von Wörtern auf allen Dokumenten des WWW ausführen. Ihr System verwendet dafür etwa 100 Map-Tasks.

a) Erwarten Sie große Unterschiede in den Verarbeitungszeiten der Reducer, wenn keine Combiner verwendet werden? ***Reducer, die für Stoppwörter (z.B. und, der, …) zuständig sind, benötigen wesentlich länger als Reducer, die für seltene Wörter zuständig sind***

b) Welche der folgenden Aussagen trifft zu? Je kleiner die Anzahl der Reduce-Tasks, desto ***ähnlicher die Ausführungszeiten der Tasks. Begründung: häufige und seltene Wörter werden gleichmäßig über die Reducer-Tasks verteilt***

c) Erwarten Sie große Unterschiede in den Verarbeitungszeiten der Reducer, wenn Combiner verwendet werden?
   - ***Da nur 100 Map-Tasks, umfassen die Ergebnisse eines Combiner beinahe alle vorkommenden Wörtern***
   - ***Jeder Reducer bekommt dann ca 100 Werte pro Schlüssel (genauso viele Combiner wie Map-Tasks)***

Übung 2: Lösung

Formulieren Sie MapReduce-Algorithmen für folgende Anfragen auf eine sehr große Datei mit ganzen Zahlen

- Maximum
  - Map: Berechnung des Maximum m über Chunk und Abbildung auf (1, m)
  - 1 Reducer bildet Maximum über alle Maxima der Mapper
- Durchschnitt
  - Map: Abbildung der Summe s und Anzahl n der Elemente des Chunk auf (1, (s,n))
  - 1 Reducer berechnet Durchschnitt
- Eindeutige Elemente (distinct)
  - Map: Abbildung eines Wortes w auf (w,w)
  - Ein Reducer pro Wort
- Anzahl eindeutiger Elemente
  - Über 2 MapReduce-Prozeduren

Übung 3

Gegeben ist folgender MapReduce-Algorithmus für den 2-fachen Join über 3 Relationen: ***R(A,B) ⋈ S(B,C) ⋈ T(C,D)***

- 2 Hashfunktionen:
  - $h_B: Werte(B) \rightarrow \{1, \ldots, g\}$
  - $h_C: Werte(C) \rightarrow \{1, \ldots, g\}$
- **Map:**
  - Elemente (a,b) aus R(A,B) auf $((h_B(b), j)$, "R", $(a, b))$ für alle j $\in \{1, \ldots, g\}$
  - Elemente (c,d) aus T(C,D) auf $((i, h_C(c))$, "T", $(c, d))$ für alle i $\in \{1, \ldots, g\}$
  - Elemente (b,c) aus *S(B,C)* auf $\Big(\big(h_B(b), h_C(c)\big)$, "$S$", $(b, c)\Big)$
- Ein **Reducer** für jedes Paar (i, j) mit i $\in \{1, \ldots, g\}$ und j $\in \{1, \ldots, g\}$
  - Erhält Liste von Tupeln aus R, S und T mit $h_B(b) = i$ und $h_C(c) = j$
  - Reducer erzeugen Join aus diesen Tupeln
- **Berechnen Sie die Replikationsrate r und die Reducer-Size q!**

Übung 3: Lösung

- **Berechnen Sie die Replikationsrate r und die Reducer-Size q!**
- Sei b die Anzahl der eindeutigen Elemente des Attributs B und b_{max} das maximale Vorkommen eines Elementes aus B in R
- Sei c die Anzahl der eindeutigen Elemente des Attributs C und c_{max} das maximale Vorkommen eines Elementes aus C in T
- Sei $s_{\max}$ das maximale Vorkommen eine Kombination von Elementen (b,c) in S

$$r = g\frac{|R| + |T|}{|R| + |T| + |S|} + \frac{|S|}{|R| + |T| + |S|}$$

$$q = \frac{b}{g} \cdot b_{max} + \frac{c}{g} \cdot c_{max} + \frac{b}{g} \cdot \frac{c}{g} \cdot s_{max}$$