Ausgewählte Kapitel sozialer Webtechnologien - Einführung

Seminaristischer Unterricht

Prof. Dr.-Ing. Hendrik Gärtner

Gliederung

- Übersicht über die Veranstaltung
  - Programmiertechniken - Scala
  - Klassische Relationale Datenbanken/NoSQL/Analyseframeworks
  - Datenanalyse
- Organisatorisches

Termine

Seminaristischer Unterricht:

Mittwoch von 12:15 – 13:45 Uhr

Raum WHC 639

Übungen:

Mittwoch von 14:00 – 15:30 Uhr alle ungeraden Wochen

Raum WHC 639L

Veranstaltung hat 5 Credit Points – das entspricht einem Workload von 150 Stunden. Viel Eigenarbeit erforderlich.

Findet eine parallele Veranstaltung statt? Können die Übungstermine wechseln?

Kontaktdaten

Prof. Hendrik Gärtner
E-Mail-Adresse: gaertner@HTW-Berlin.de
Tel: 0049 30 5019 3594

Sprechstunde

donnerstags 14:00 – 15:00
und nach Vereinbarung
Raum WH C 608

Literatur

Programmierung:

- *Odersky, M.; Spoon, L.; Venners, B.:* Programming in Scala- A comprehensive step-by-step guide, Artima Press.

Datenbanken/Frameworks:

- *Karau,H.; Konwinski, A.; Wendell, P.; Zaharia,M.: Learning Spark: Lightning-Fast Big Data Analysis, O'Reilley, 2015.*
- *Ryza, S.; Laserson, U.; Owen, S.; Wills, J.: Advanced Analytics with Spark: Patterns for Learning from Data at Scale, O'Reilley, 2015.*
- *Parsian,M.: Data Algorithms: Recipes for Scaling Up with Hadoop and Spark, O'Reilley, 2015.*
- *Sadalage, P.; Fowler, M.*: NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence, Addison Wesley, 2012.

Mehr unter:
http://people.f4.htw-berlin.de/lehrende/gaertner/veranstaltungen/ausgewaehltekapitelswt/literatur.html

Tipp

- http://www.coursera.org
- http://www.edx.com

Kurse zu den Themen:

  - Functional Programming Principles with Scala, Martin Odersky
  - Machine Learning, Andrew Ng
  - Spark, Data Analysis

- Alle Kusunterlagen befinden sich unter:

  plus.htw-berlin.de
  Kurs: Big Data WS2051/2016
  Token: BigData2016#

Welche Probleme gibt es bei der Verarbeitung von Daten geben?

Datenaufbereitung 1/2

- Inputformat: Text oder Binary?

Beispiel Logdatei eines Web-Servers:

199.72.81.55 - - [01/Jul/1995:00:00:01 -0400] "GET /history/apollo/ HTTP/1.0" 200 6245
unicomp6.unicomp.net - - [01/Jul/1995:00:00:06 -0400] "GET /shuttle/countdown/ HTTP/1.0" 200 3985
199.120.110.21 - - [01/Jul/1995:00:00:09 -0400] "GET /shuttle/missions/sts-73/mission-sts-73.html HTTP/1.0" 200 4085
burger.letters.com - - [01/Jul/1995:00:00:11 -0400] "GET /shuttle/countdown/liftoff.html HTTP/1.0" 304 0

Beispiel Twitter-Datenstream:

{"created_at":"Mon Sep 22 05:46:38 +0000 2014","id":513927330952380416,"id_str":"513927330952380416","text":"RT @Michael5SOS: @pewdiepie first things first im a burger","source":"\u003ca href=\"http:\/\/twitter.com\/download\/iphone\" rel=\"nofollow\"\u003eTwitter for iPhone\u003c\/a\u003e","truncated":false,"in_reply_to_status_id":null,"in_reply_to_status_id_str":null,"in_reply_to_user_id":null,"in_reply_to_user_id_str":null,"in_reply_to_screen_name":null,"user":{"id":745744075,"id_str":"745744075","name":"LisaVDV","screen_name":"VdvLisa","location":"","url":null,"description":null,"protected":false,"verified":false,"followers_count":17,"friends_count":20,"listed_count":0,"favourites_count":153,"statuses_count":56,"created_at":"Wed Aug 08 18:53:34 +0000 2012","utc_offset":null,...

Datenaufbereitung 2/2

- In welchem Format sind die Daten? Ist die Spezifikation des Formats eindeutig?
- Wie werden die Daten eingelesen und wie sieht die Objektstruktur aus, in der sie gehalten werden?
- Enthalten die Daten Formatfehler und wenn ja, wie soll damit umgegangen werden?
- Wie soll mit fehlenden Werten innerhalb der Daten umgegangen werden? Sollen sie ignoriert werden?
- Welche Methoden bietet die verwendete Programmiersprache, um die Daten einzulesen?

Datenabfragen/-analysen

- Durchsuchen nach bestimmten Datensätzen:
  - Welche Tweets hat ein bestimmter User abgesetzt?
  - Welche Tweets beinhalten einen speziellen Hashtag?
  - Von welchen IP-Adressen wurde eine bestimmte Seite aufgerufen?
  - Welche Seiten wurden von einer IP-Adresse aufgerufen?
- Aggregation von Daten
  - Welcher User hat am meisten Tweets abgesetzt?
  - Welche Seite wurde am häufigsten aufgerufen?
  - Auf welcher Seite wird durchschnittlich am längsten verweilt?
- Maschinelles Lernen

Maschinelles Lernen – Simples Anwendungsbeispiel

- Vorhersage von Gewinnen mit Foodtrucks

Datenmenge: Tabelle Einwohner der Stadt und Gewinn

Einwohner	Profit
6.1101	17.592
5.5277	9.1302
8.5186	13.662
7.0032	11.854
5.8598	6.8233
8.3829	11.886
7.4764	4.3483
8.5781	12
6.4862	6.5987
…	…

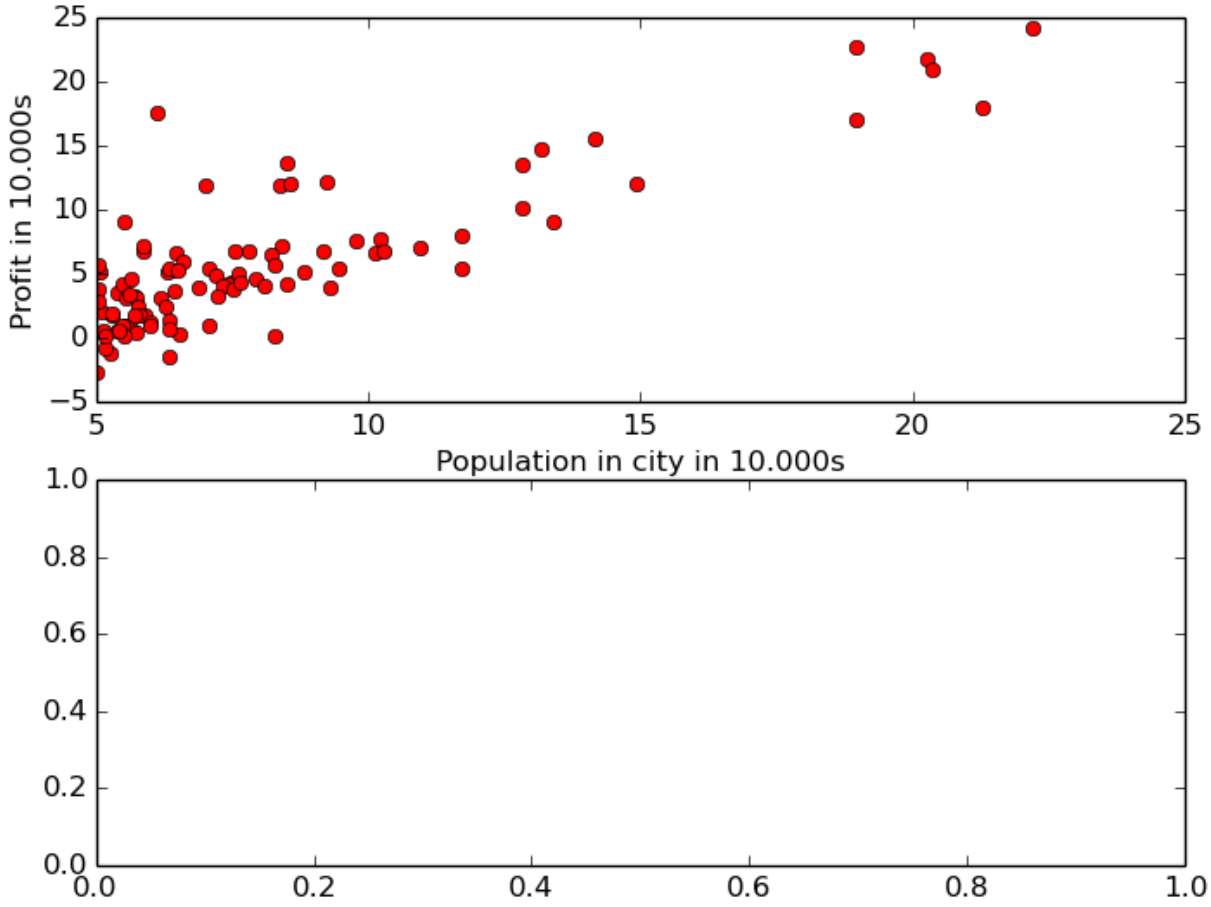
Gibt es einen Zusammenhang zwischen Größe der Stadt und dem Gewinn? Ist es sinnvoll die Foodtrucks in kleine oder große Städte fahren zu lassen?

Supervised Learning

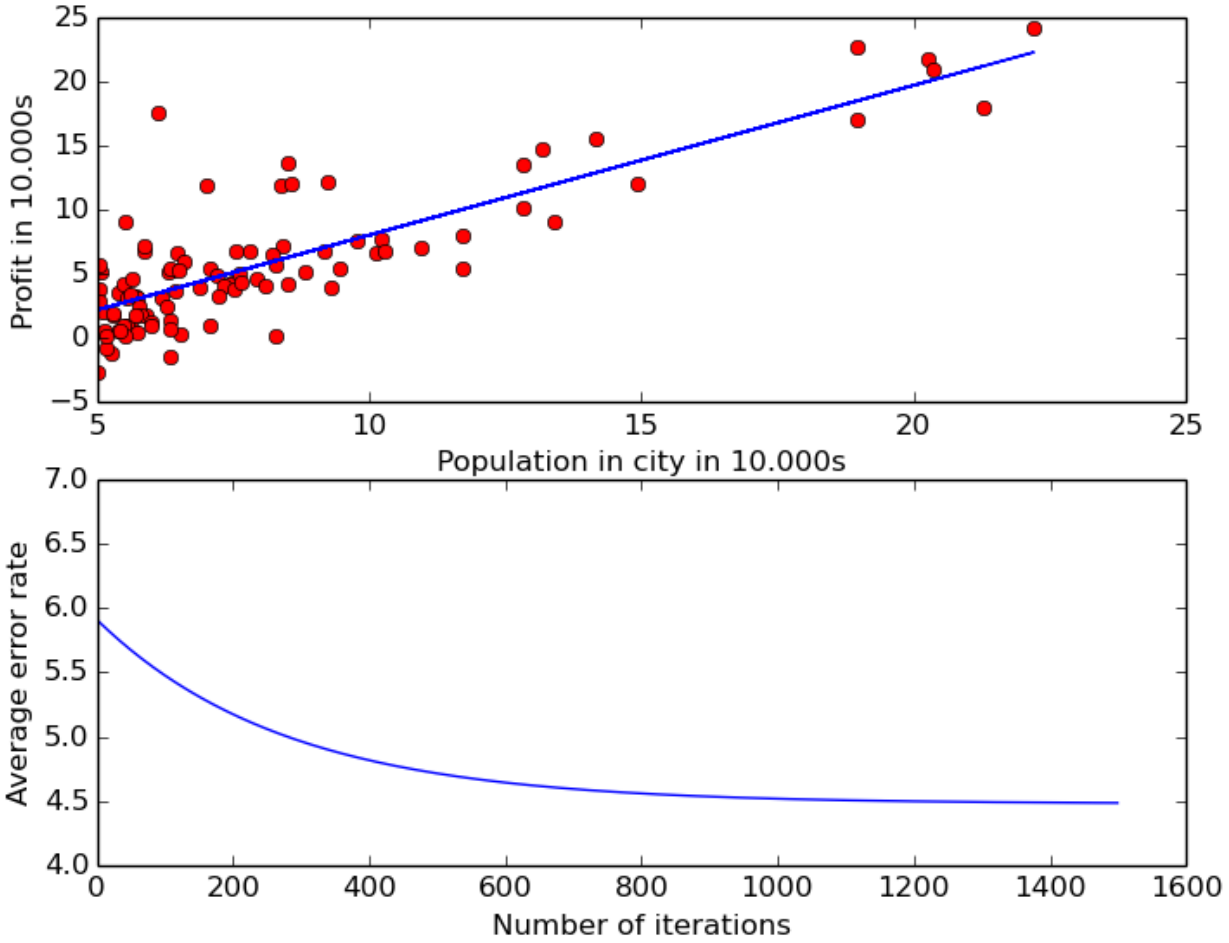
- Eine Menge von Gelabelten Trainingsdaten
  Merkmal: Stadtgröße
  Label: Profit
- Trainieren einer Funktion auf Basis der Trainingsdaten
- Funktion bildet ab, bei welcher Stadtgröße welcher Gewinn zu erwarten ist
- Verfahren: Lineare Regression

- Unsupervised Learning: Ungelabelte Daten, z.B. finden von zusammenhängenden Strukturen

Graphische Interpretation des Verfahrens (1/2)



Graphische Interpretation des Verfahrens (2/2)



Beispiele für Supervised Learning

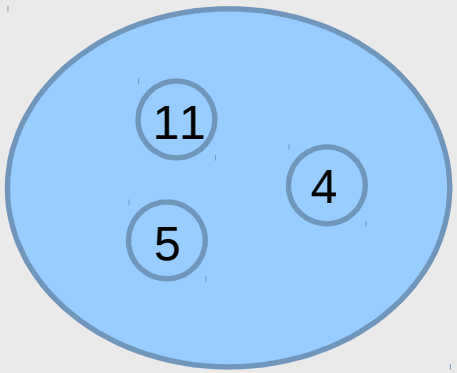
- E-Mail-Spam-Filter:
  Daten: E-Mails als Spam oder nicht Spam klassifiziert
  Merkmale: Vorkommen von Wörtern
  Verfahren: Logistische Regression
- Empfehlungssysteme:
  Daten: Datenbank mit Filmen und Bewertungen von Usern
  Merkmale: Filmeinschätzungen und Userbewertungen
  Verfahren: Collaborative Filtering
- Handschrifterkennung:
  Daten: Bilder von Zahlen mit Ihren realen Werten als Labels
  Merkmale: Pixel im Bild
  Verfahren: Neuronale Netze

Warum eignen sich Funktionale Konzepte für den Umgang mit großen Datenmengen?

Higher Order Functions

Sie lassen mit ihren Higher Order Function (werden häufig auch Lambda Expressions genannt) Operationen auf einem höheren Abstraktionsgrad zu!

Menge l




```
List<Integer> l=
    new ArrayList<Integer>();
l.add(4);
l.add(5);
l.add(11);
```

Aufgabe:

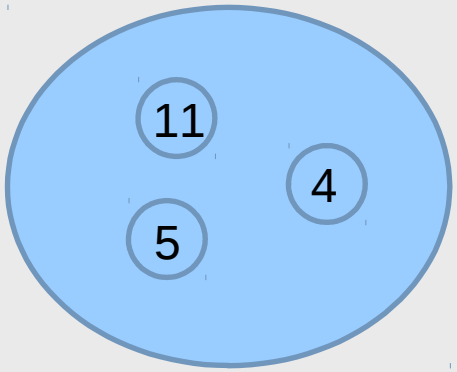
- Verdoppelung jedes Elements der Menge
- Ausgabe aller Elemente

```
for (int i=0; i<l.size();i++){
    l.set(i, l.get(i)*2);
}

for (Integer i : l){
    System.out.println(i);
}
```

Beispiel in Scala

Menge l




```
val l:List[Int]= List(5,6,11)
```

Aufgabe:

- Verdoppelung jedes Elements der Menge
- Ausgabe aller Elemente

```
l.map(X=>2*X).foreach(X=>println(X))
```

Erkärung:
(X=>2*X): Ist eine Funktion die ein Element X auf das Element 2*X abbildet. Analog dazu in der Mathematik: f(X)= 2*X
map: Wendet eine Funktion auf alle Elemente einer Menge an und gibt eine neue Menge mit den veränderten Elementen zurück
foreach: Wendet eine Funktion auf eine Menge von Elementen an, hat im Gegensatz zu map jedoch kein Rückgabewert.

Parallelisierung der Verarbeitung (1/2)

Beispiel: Erzeugen einer Liste mit 10 Mio Doubles

Aufgabe: Berechnen sie das Quadrat des Sinus des Logarithmus (Funktion egal – nur Workload gefragt)

Imperative Variante:

```
val li1 = Range.Double(1D,10000001D,1).toArray
val t1 = System.nanoTime
var res1=0.0
for (i <- 0 to 9999999) res1= res1 + math.pow(math.sin(math.log(li1(i))),2)
println("time: "+(System.nanoTime-t1)/1e6+"ms")
println(res1)
```

Ergebnis:

time: 1305.274058ms

2855319.2051415644

Parallelisierung der Verarbeitung (2/2)

Funktionale Variante:

```
val li2 = Range.Double(1D,10000001D,1).toArray.par

val t2 = System.nanoTime

val res2= li2 map (X=> math.pow(math.sin(math.log(X)),2))
        reduce ((X,Y)=> X+Y)

println("time: "+(System.nanoTime-t2)/1e6+"ms")

println(res2)
```

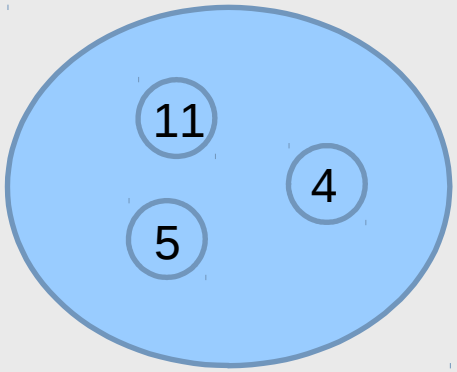
Ergebnis: ?

```
time: 234.545985ms    // 7x schneller

2855319.2051416417
```

Beispiel in Java 8

Menge l




```
List<Integer> l = Arrays.asList(5,6,11);
```

Aufgabe:

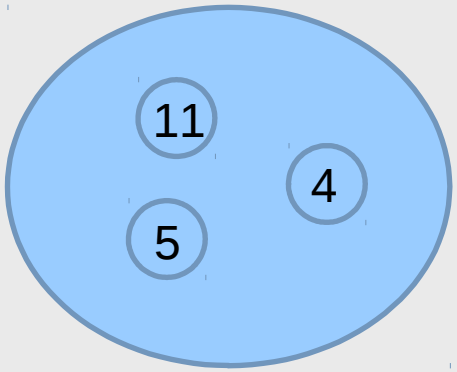
- Verdoppelung jedes Elements der Menge
- Ausgabe aller Elemente

```
l.stream().map(x->2*x).forEach(System.out::println);
```

Erkärung:
(X->2*X): Ist eine Funktion die ein Element X auf das Element 2*X abbildet. Analog dazu in der Mathematik: f(X)= 2*X
map: Wendet eine Funktion auf alle Elemente einer Menge an und gibt eine neue Menge mit den veränderten Elementen zurück
forEach: Wendet eine Funktion auf eine Menge von Elementen an, hat im Gegensatz zu map jedoch kein Rückgabewert.

Beispiel Reduce

Menge l



Aufgabe: Reduzieren Sie die Menge l auf einen Wert, in dem Sie alle Elemente aufaddieren:

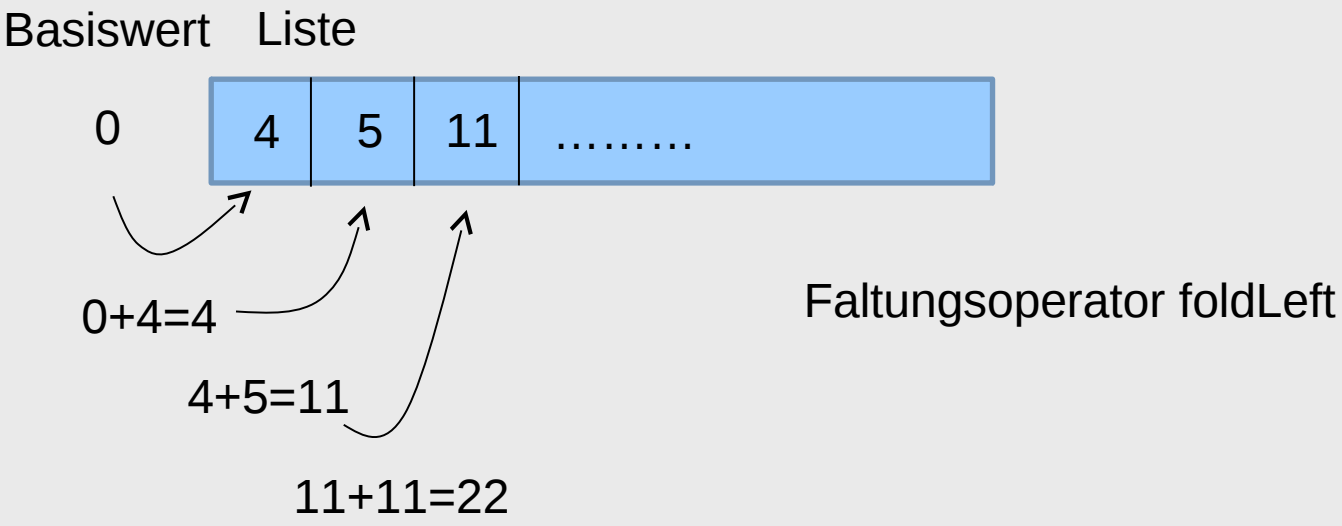
Java:

```
int result=0;
for (Integer i : l){
        result= result+i;
}
System.out.println(result);
```

Scala:

```
l.foldLeft(0)((X,Y)=>X+Y)
```

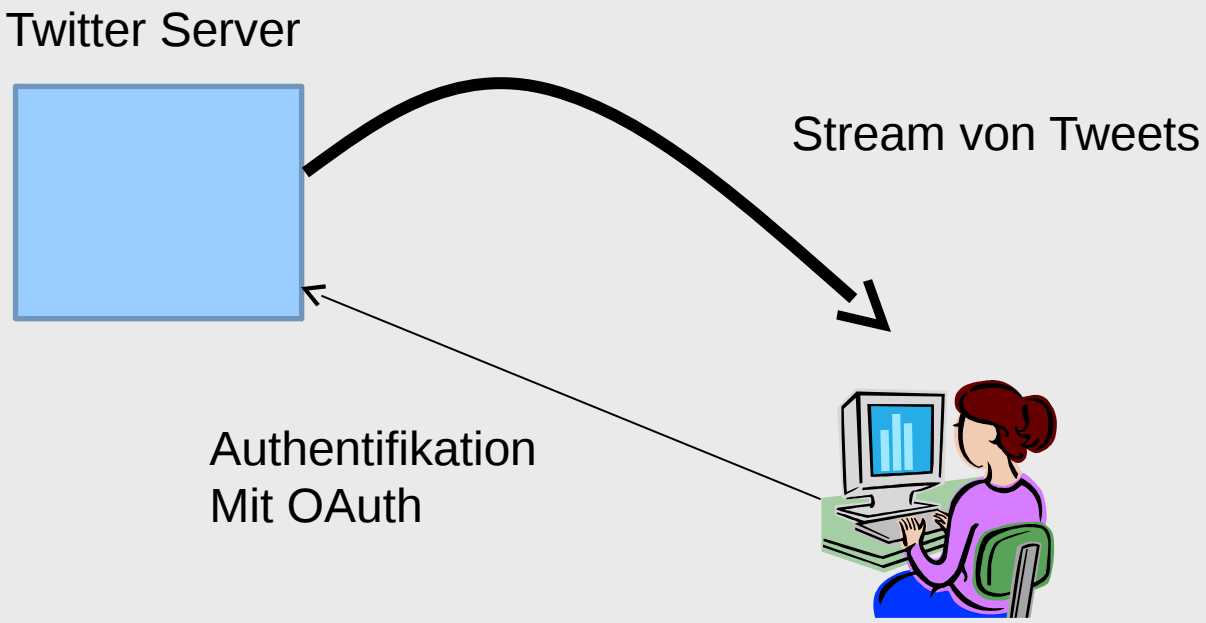
Funktionsweise



- Ursprung von MapReduce
- Programmierer definiert nur noch die Map- und die Reduce-Funktion
- Alles Weitere wird vom System bereitgestellt
- Map und Reduce sind unabhängig von der Größe der Liste

Beispiel Twitter

Twitter API

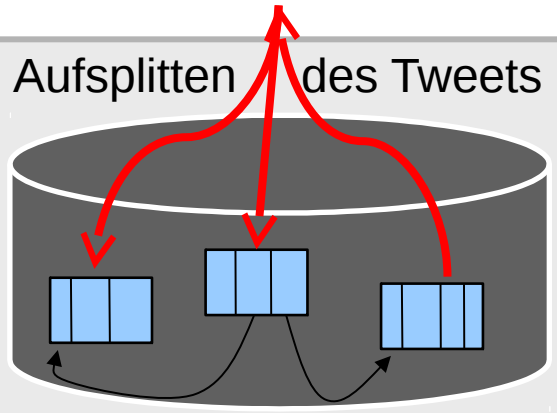


- Jeder Tweet besteht aus einem JSON-Dokument
- Im JSON-Dokument befindet sich der eigentliche Text
- Der Rest beschreibt den User, die Hashtags, etc.
- Datenvolumen: ca. 700MB

Wie soll man mit den vielen Daten umgehen?

Klassischer Ansatz


```
{
  "statuses": [
    {
      "coordinates": null,
      "favorited": false,
      "truncated": false,
      "created_at": "Mon Sep 24 03:35:21 +0000
2012",
      …
}
```

Klassische Relationale Datenbank
Bspw. PostgreSQL

- Aufsplitten des Tweets
- Speichern in mehreren Tabellen:
  - versendender User
  - versendete Nachricht
  - Assoziierte Hashtags,
  - …
- Tabellen in normalisierter Form verhindert Redundanzen

Aber:

- Jede Abfrage eines Tweets ist mit komplexen Join-Operationen verbunden

Fortgeschrittene Datenbanktechniken

- Indexierung von Tabellen (B-Bäume, Hash-Indexe)
- Anfrageoptimierung (Relationale Algebra?)
- Implementierung von Datenbankfunktionen (plpgsql)
- Einsatz von Triggern

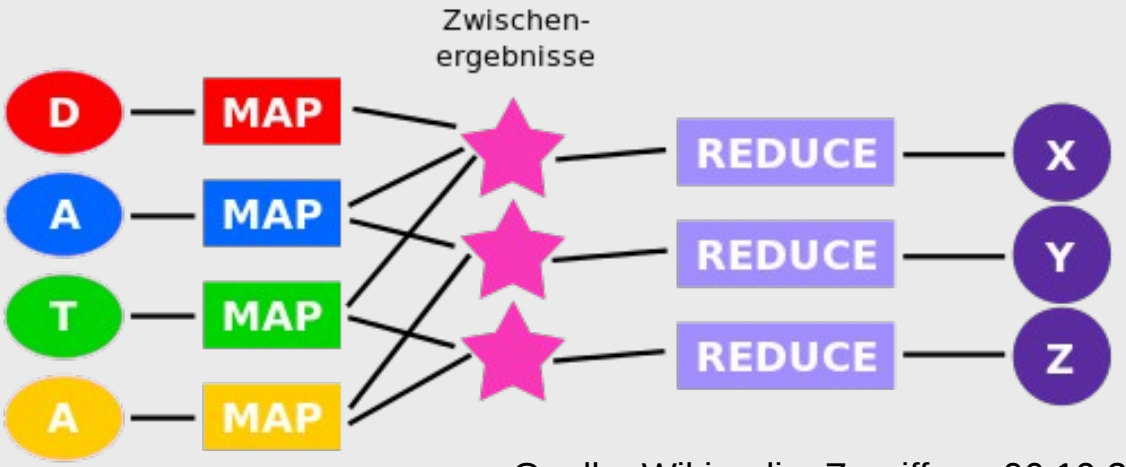
Problemstellung: Flaschenhals Festplatte

Wie gut skaliert der Ansatz?

MapReduce-Ansatz

- Einschränkung: Daten werden nur geschrieben und abgefragt – nicht geändert (z.B. Tweets oder Logfiles)
- Aufteilung der Files in Chunks (meist 64MB groß)
- Replikation der Chunks innerhalb eines verteilten Filesystems
- Bearbeitung der Teile mittels des MapReduce-Ansatzes

MapReduce-Ansatz



Quelle: Wikipedia, Zugriff am 06.10.2014

- Die Eingabedaten (D, A, T, A) werden auf eine Reihe von Map-Prozessen erteilt (bunte Rechtecke),
- Jede dieser Map-Instanzen legt Zwischenergebnisse ab
- Von jeder Map-Instanz fließen Daten in eventuell verschiedene Zwischenergebnisspeicher
- Sind alle Zwischenergebnisse berechnet, ist diese sogenannte Map-Phase beendet und die Reduce-Phase beginnt.
- Für jeden Satz an Zwischenergebnissen berechnet jeweils genau ein Reduce-Prozess

Infrastruktur

- Start mit HDFS und Hadoop (Verwendung mit Java)
- Spark
- Cloudera Distribution (als VM oder Komplettinstallation)
- Clusterinfrastruktur:
  - dumbo01.f4.htw-berlin.de
  - …
  - dumbo06.f4.htw-berlin.de

Mögliche Anwendungsgebiete

- Analyse der Twitterdaten
- Beispiele aus dem Information Retrieval
  - Suche
  - Ähnlichkeit von Dokumenten berechnen
  - Plagiatserkennung
  - PageRank (Relevanz von Webseiten auf Basis einer Linkanalyse)

Achtung: Es wird ein wenig Lineare Algebra (Rechnen mit Matrizen) und Wahrscheinlichkeitsrechnung benötigt!

Und wie können Daten jetzt geändert werden???

NoSQL-Datenbanken

- Grundlagen: ACID-Prinzip, Transaktionen, BASE, Eventual Consistency, Verteilungsprinzipien
- Verschiedene Datenmodelle
  - Dokumentenorientierte DB (z.B. CouchDB oder MongoDB)
  - Graphdatenbanken (z.B. Neo4J)
  - Key/Value-Datenbanken (z.B. Riak)
  - Spaltenorientierte Datenbanken (z.B. Cassandra, SimpleDB)
  - OO-DB, Verteilte ACID-DB, etc.
- Diskussion über den Einsatz der verschiedenen DBs
- Eventuell Vorstellung der einzelnen Vertreter im Rahmen einer Belegarbeit (würde dann mit in die Note einfließen)

Ziele der Veranstaltung

- Erweiterung der Programmierkenntnisse
  - Einführung in die Funktionale Programmierung (minimal – Extrakurs) und Kombination mit der Objektorientierung
  - Anwendung der Paradigmen mit Scala
- Umgang mit großen Datenmengen
  - Daten können im Hauptspeicher gehalten werden
    - Programmiertechniken und Mengenimplementierungen
    - Korrespondiert mit der Einführung in Scala
  - Daten passen nicht in den Hauptspeicher
    - Klassischer Ansatz: Relationale Datenbanken
    - MapReduce-Frameworks
    - NoSQL-Datenbanken
- Algorithmen für die Datenanalyse

Scheinkriterien - Allgemein

- Drei Belegarbeiten
  - Belegarbeiten sollen in Teams mit 1-2 Personen bearbeitet werden
  - Lösungen müssen in der Übung präsentiert werden
  - Belegarbeiten werden boolesche bewertet und dienen der Klausurzulassung

- 90-minütige Klausur am Ende des Semesters
  - Reine Klausur mit Aufgaben, die schriftlich zu lösen sind
  - Kein Praktischer Teil mit dem Rechner
  - Sowohl der im Seminaristischen Unterricht gelehrte Stoff als auch die praktischen Beispiele sind für das Bestehen der Klausur wichtig
  - Ist in der Regel eine Kofferklausur

Klausur

- Prüfungszeiträume
  - 01.02.2016 - 20.02.2016
  - 29.03.2016 - 09.04.2016
- Vorschlag Prüfungstermine
  - 02.02.2016 von 12:15-13:45 Uhr
  - 06.04.2015 von 15:45-17:15 Uhr
- 16 Veranstaltungen

Bitte überprüfen, ob es Terminkollisionen gibt!!!!

Unterrichtsmaterialen

- Vorlesungsfolien werden unter http://plus.f4.htw-berlin.de bereitgestellt (Einloggen mit FB4-Account)
- Zu vielen VL-Inhalten wird es ein Übungsblatt geben (Bearbeitung freiwillig)
- In den Übungen werden die Lösungen besprochen sowie die neuen Aufgabenstellungen
- Es werden Musterlösungen zu den Übungen bereitgestellt (nicht zu den Belegarbeiten)
- Funktionsfähigkeit der Belege muss in der Übung oder in der Sprechstunde demonstriert werden
- Für freiwilligen Übungen, kann ich folgende Seite empfehlen: http://projecteuler.net/

Vielen Dank für

Ihre Aufmerksamkeit

Prof. Dr.-Ing. Hendrik Gärtner