Healthcare Data Analytics

Big Data Processing & Spark

Dr. Michael Strobel

04.07.2022

Inhalt

Letzte Vorlesung

- Datenbanksysteme
- Data Warehousing
  - OLAP vs OLTP
  - Sterne und Schneeflocken
- Intro zu Big Data

Diese Woche

- UNIX Philosophie
- Big Data Processing
- Spark
- Stream Processing
- GraphDB

Online Services / Batch Processing / Stream Processing

- Daten Anwendungen können unterschiedliche Anforderungen haben, letzte Woche haben wir OLTP und OLAP Systeme gesehen
- Datenverarbeitungssysteme lassen sich in drei Kategorien aufteilen
  - Online Services
  - Batch Processing
  - Stream Processing
- Die verschiedenen Anforderungen resultieren in sehr unterschiedlichen Algorithmen und Implementierungen

Online Services

- Das System wartet auf eingehende Request und startet daraufhin Berechnungen
- Die Bearbeitung erfolgt so schnell wie möglich und mit geringer Zeitverzögerung
- Die Datenmengen sind verhältnismäßig gering
- Wichtigste Performance Metriken: response time und availability
- Beispiele: REST Api und OLTP Systeme

Batch Processing

- Batch Processing verarbeitet in der Regel **große Datenmengen**
- Anwendungsgebiet sind Datenverarbeitung: Input Daten werden zu Output Daten verarbeitet und abgelegt
- Daten sind zu Begin des Batch Processing bekannt und fixiert
- Die Laufzeit ist lang: Minuten bis Tage
- Der User wartet in der Regel nicht auf das Ergebnis, normalerweise laufen die Anfragen auf periodischer Basis (täglich, wöchentlich, . . .)
- Wichtigste Performance Metrik: Datendurchsatz

Stream Processing

- Mittelweg aus Online Services und Batch Processing
- Anwendungsgebiet sind Datenverarbeitung: Input Daten werden zu Output Daten verarbeitet und abgelegt
- Stream Processing reagiert auf auf Events (z.B. Eingang von Daten)
- Verarbeitung findet zeitnah statt
- Datengröße variiert und ist zum Start des Stream Processing unbekannt
- Die Laufzeit ist geringer als beim Batch Processing

UNIX Philosophie

Die UNIX Philosophie nach McIlroy, dem Erfinder u.a. der UNIX Pipeline vom Jahr 1964

- Schreibe Computerprogramme so, dass sie nur eine Aufgabe erledigen und diese gut machen.
- Schreibe Programme so, dass sie zusammenarbeiten.
- Schreibe Programme so, dass sie Textströme verarbeiten, denn das ist eine universelle Schnittstelle.

HL7 V2 Beispiel, Körpergröße gerundet auf eine Stelle nach dem Komma

```
OBX|1|NM|^Body Height||1.80|m^Meter^ISO+|||||F
OBX|1|NM|^Body Height||1.50|m^Meter^ISO+|||||M
OBX|1|NM|^Body Height||1.70|m^Meter^ISO+|||||F
OBX|1|NM|^Body Height||1.80|m^Meter^ISO+|||||M
OBX|1|NM|^Body Height||1.90|m^Meter^ISO+|||||F
OBX|1|NM|^Body Height||1.70|m^Meter^ISO+|||||F
OBX|1|NM|^Body Height||1.40|m^Meter^ISO+|||||D
OBX|1|NM|^Body Height||2.00|m^Meter^ISO+|||||F
OBX|1|NM|^Body Height||1.70|m^Meter^ISO+|||||F
OBX|1|NM|^Body Height||1.50|m^Meter^ISO+|||||M
```

UNIX Philosophie, Beispiel cont'd

Wie berechnet man mit UNIX tools die drei Pantient:innengruppen mit der größten Körpergröße?

```
cat hl7.txt |
awk -F\| '{print $6}' |
sort |
uniq -c |
sort -r -n |
head -n 3

> 3 1.70
> 2 1.80
> 2 1.50
```

- cat: liest die Datei in die UNIX Pipeline ein
- awk extrahiert die Größe (-F gibt das Zeichen an bei dem der Split erfolgen soll)
- sort sortiert lexikographisch aufsteigend, sort -n -r sortiert numerisch absteigend
- uniq -c, zählt die Anzahl gleicher und aufeinander folgender Zeilen
- head -n 3 gibt die 3 größten Instanzen aus

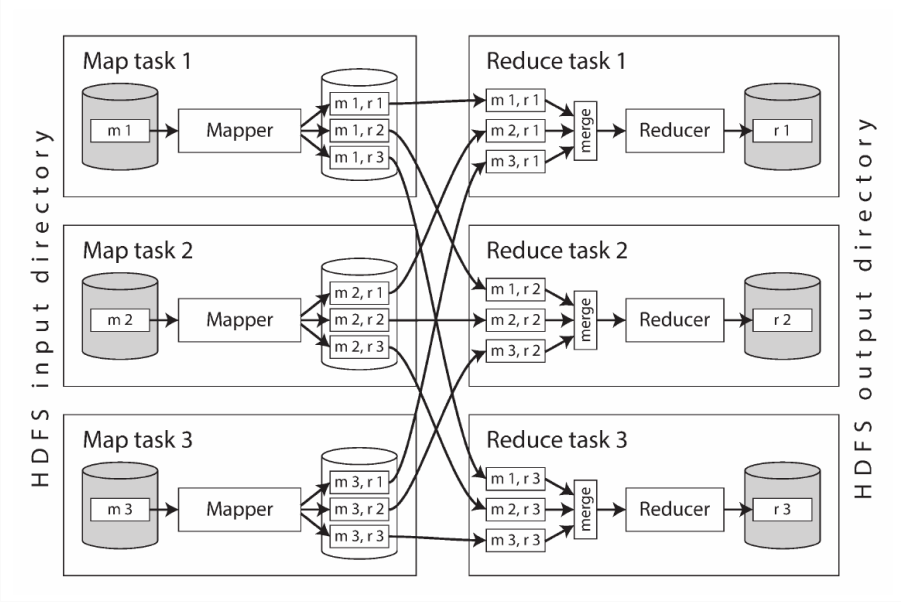
MapReduce

- MapReduce generalisiert die Idee UNIX Pipelines auf große Datenmengen
- Die Datenmengen werden ein Cluster von Rechnern verteilt und parallel abgearbeitet
- Die Idee von MapReduce wurde von Google popularisiert um den Google Suchindex zu erstellen
- Mit MapReduce können Operationen auf mehreren Petabyte Daten verarbeitet werden

MapReduce, cont'd

- MapReduce besteht aus zwei Phasen
  - Map: die Map Funktion wird auf jeden Eintrag (Record) der Daten angewendet, dabei wird ein *Key/Value* Paar generiert
  - Reduce: die Ergebnisse von der Map Phase werden nach Keys sortiert und dann zu einem Endergebnis weiter verarbeitet
- Map und Reduce sind von der funktionalen Programmierung inspiriert und haben keine Seiteneffekte
- Im Unix Beispiel:
  - Map: `cat hl7.txt | awk -F| '{print $6}' | sort | uniq -c`
  - Reduce: `sort -r -n | head -n 3`

MapReduce, Visualisierung



Kleppmann, M. (2017). Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems.

Beispiel: wie viel Patientinnen wurden jeweils in einem Monat behandelt?

```
db.observations.mapReduce(
  function map() {
    const year = this.observationTimestamp.getFullYear();
    const month = this.observationTimestamp.getMonth() + 1;
    emit(year + "-" + month, this.numPatients);
  },
  function reduce(key, values) {
    return Array.sum(values);
  },
  {
    query: { gender: "female" },
    out: "monthlyPatientReport",
  }
);
```

MapReduce: Datenbeispiel

Gegeben seien folgende Input Daten für MapReduce

```
[
  {
    observationTimestamp: Date.parse("Mon, 25 Dec 1995 12:34:56 GMT"),
    gender: "female",
    names: ["Patient A", "Patient B", "Patient C"],
    numPatients: 3,
  },
  {
    observationTimestamp: Date.parse("Tue, 12 Dec 1995 16:17:18 GMT"),
    gender: "female",
    names: ["Patient B", "Patient D", "Patient E", "Patient F"],
    numPatients: 4,
  },
];
```

Dieser Code würde: `emit("1995-12", 3)` und `emit("1995-12", 4)` ausführen und zu `reduce("1995-12", [3, 4])` mit Ergebnis 7 führen.

Hadoop

- Apache Hadoop ist eine mögliche und sehr bekannte Implementierung des MapReduce Modells
- Hadoop verteilt Berechnungen von großen Datenmengen auf Rechner im Cluster
- Die Skalierung reicht von einem Rechner bis hin zu tausenden Maschinen
- Dabei ist Hadoop fehlertolerant und ersetzt defekte oder unerreichbare Knoten automatisch
- Ein wichtiger Bestandteil davon ist das Hadoop distributed file system (HDFS)
  - Verteilt Daten über Knoten im Cluster
  - Design für konventionelle Hardware und Netzwerke
  - Stellt ein großes Dateisystem bereit auf die zugegriffen werden kann und nutzt den Speicher aller Machinen
  - Stellt Redundanz sicher, falls es zu Ausfällen kommt

Beyond MapReduce

Obwohl MapReduce ein großer Schritt im Bereich Big Data war hat es auch einige Nachteile

- Zwischenergebnisse der Berechnungen sind oft von Interesse, sind nur über Umwege verarbeitbar
- Komplexe Berechnungen, wie z.B. Recommender Systeme, können 50 bis 100 MapReduce Schritte beinhalten
- Erst wenn alle Ergebnisse beim Reducer vorliegen kann dieser starten, daher kann es zu Wartezeit kommen
- Reduce benötigt eventuell Daten von einem anderen Knoten und muss dann über das Netzwerk kopiert werden

Um diese Nachteile zu beheben nutzt man heute sog. *Dataflow engines* wie Pig, Hive oder Spark.

Spark

- Spark ist ein Framework, dass sich auf Big Data Analytics spezialisiert
- Spark läuft wie Hadoop verteilt auf einem Cluster von Rechnern
- Spark verarbeitet sowohl Batch als auch Stream Processing Aufgaben
- Kann schnell SQL Queries ausführen und steht damit in Konkurrenz zu Data Warehouses
- Ist auf Data Science Tassks ausgelegt und hat (teilweise) Kompatibilität zu Pandas
- Ist auch für Machine Learning Tasks geeignet

Spark, Beispiel

```
from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("KMeansExample").getOrCreate()
dataset = spark.read.format("libsvm").load("data/mllib/sample_kmeans_data.txt")

# Trains a k-means model.
kmeans = KMeans().setK(2)
model = kmeans.fit(dataset)
# Make predictions
predictions = model.transform(dataset)
# Evaluate clustering by computing Silhouette score
evaluator = ClusteringEvaluator()
silhouette = evaluator.evaluate(predictions)

spark.stop()
```

Future Work

Sachen die Sie vielleicht spannend finden könnten

- Recurrent Neural Networks:
  - wie modelliert man Zeitabläufe mit und wiederkehrende Ereignisse mit ML?
  - wie funktioniert moderne Spracherkennung?
- Reinforced Learning: wie funktioniert Alpha Go und co?
- Stream Processing von Daten
- Deployment von Machine Learning Modellen
- ...

Referenzen

- Kleppmann, M. (2017). Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems. " O'Reilly Media, Inc.".