

Store Document
Key Value
NoSQL
Datenbanken
Column
Graph

Übersicht Was lernen wir?



- › Motivation NoSQL
- › Theoretische Überlegungen (CAP, BASE, ACID)
- › Kategorisierung und Beispiele
- › Hands-on



HOW TO WRITE A CV



Leverage the NoSQL boom

Motivation

- › **Facebook** hat 180k Server (Ende 2012)
- › **Google** hat 450k Server (2006), über 1 Million?, 3 Milliarden Suchanfragen pro Tag
- › **Microsoft**: hat 100k - 500k Server (seit Azure)



Motivation

- › **Trend 1:** increasing data sizes (big data)
- › **Trend 2:** more connectedness (“web 2.0”)
- › **Trend 3:** more individualization (fewer structure)
- › and multimedia data

Big data

- › “huge amount of data produced by different devices and applications”
- › **Black Box Data**
 - Data captured by helicopter, airplanes, and jets, etc.
- › **Social Media Data**
- › **Stock Exchange Data**
- › **Power Grid Data** : The power grid data holds information consumed by a particular node with respect to a base station.
- › **Transport Data**
- › **Search Engine**

Big data: 3 types

- › **Structured data** : Relational data.
- › **Semi Structured data** : XML data.
- › **Unstructured data** : Word, PDF, Text, Media Logs.

NoSQL Einführung

- NoSQL = „not only SQL“
- verteilte und horizontale Skalierbarkeit, gleichrangige Knoten
- kostengünstige Rechnersysteme zur Datenspeicherung
- kein relationales Datenmodell (kein SQL)
- schemafrei / schwache Schemarestriktionen
- keine Transaktionen (nach gewisser Zeit konsistenter Zustand)

- verteilte Hardware -> hohe Ausfallsicherheit
- für spezifische Problemstellungen!

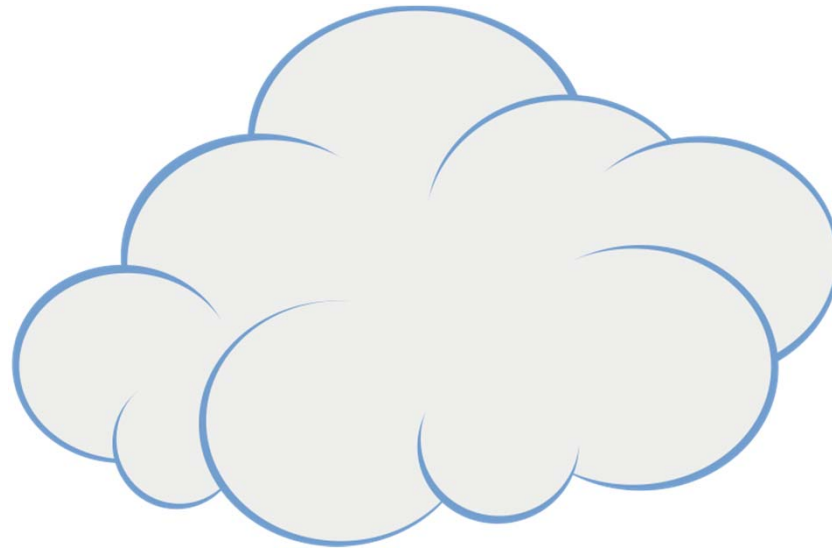
Vertikale Skalierbarkeit

- › Vertikale Skalierung (RAM,CPU,Storage)
- › Server auf mehr Leistungsfähigkeit trimmen



Horizontale Skalierbarkeit

- › Mehr (billige) Prozessoren
- › Einfügen von Nodes -> Verteilte Systeme



Verfügbarkeit

Klasse	Verfügbarkeit	Downtime/Jahr
2	99%	3 Tage 15 Stunden
3	99,9%	8 Stunden 45 Minuten
4	99,99%	52 Minuten
5	99,999%	5 Minuten

Anforderungen

- › Sicherheit (ACID)
- › Verfügbarkeit
- › unbegrenztes Wachstum

Trugschlüsse bei verteilten Systemen

- › the network is reliable
- › latency is zero
- › bandwidth is infinite
- › the network is secure
- › topology doesn't change
- › there is one administrator
- › transport cost is zero
- › the network is homogeneous

CAP Theorem *



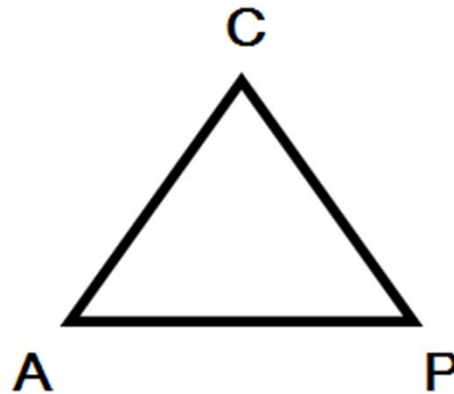
- › Laut dem CAP-Theorem kann ein verteiltes System zwei der folgenden Eigenschaften gleichzeitig erfüllen, jedoch nicht alle drei.
- › **Konsistenz (C)**
 - Alle Knoten sehen zur selben Zeit dieselben Daten.
- › **Verfügbarkeit (A)**
 - Alle Anfragen an das System werden stets beantwortet.
- › **Partitionstoleranz oder Ausfalltoleranz (P)**
 - Das System arbeitet auch bei Verlust von einzelnen Netzknoten weiter.

* Prof. Eric A. Brewer , ACM16-Symposium „Principles of Distributed Computing“, 2000


Teilung eines Netzwerkes

- › z.B. in zwei Hälften, beide getrennt erreichbar
- › Lösung A: Eine Hälfte abschalten — Konsistenz erhalten
- › Lösung B: Konsistenz aufgeben — Verfügbarkeit erhalten

CAP Theorem



- › AP – Domain Name Service
- › CA – relationale Datenbanken
- › CP - Bankautomaten



»in larger distributed-scale systems, network partitions are a given; therefore, **consistency** and **availability** cannot be achieved at the same time«

Werner Vogels, Amazon.com

ACID

› **Atomarität (Abgeschlossenheit)**

- Sequenz von Daten-Operationen entweder ganz oder gar nicht ausgeführt
- Konsistenz heißt, dass eine Sequenz von Daten-Operationen nach Beendigung einen konsistenten Datenzustand hinterlässt

› **Konsistenz**

- Konsistenter Datenzustand -> Sequenz von Operationen -> konsistenter Datenzustand

› **Isolation (Abgrenzung)**

- Keine Beeinflussung nebenläufiger Daten-Operationen

› **Dauerhaftigkeit**

- Dauerhafte Speicherung nach (erfolgreichem) Abschluss

BASE

- › Basically Available
 - most data is available most of the time
- › Soft state
 - the DB provides a relaxed view of data in terms of consistency
- › Eventually consistent
 - data is eventually copied to all applicable nodes, but there is no requirement for all nodes to have identical copies of any given data all the time.
 - Letztendlich nach einer möglichst kurzen Zeitspanne haben alle Teile eines verteiltes Systems wieder die gleiche Sicht auf die Daten.

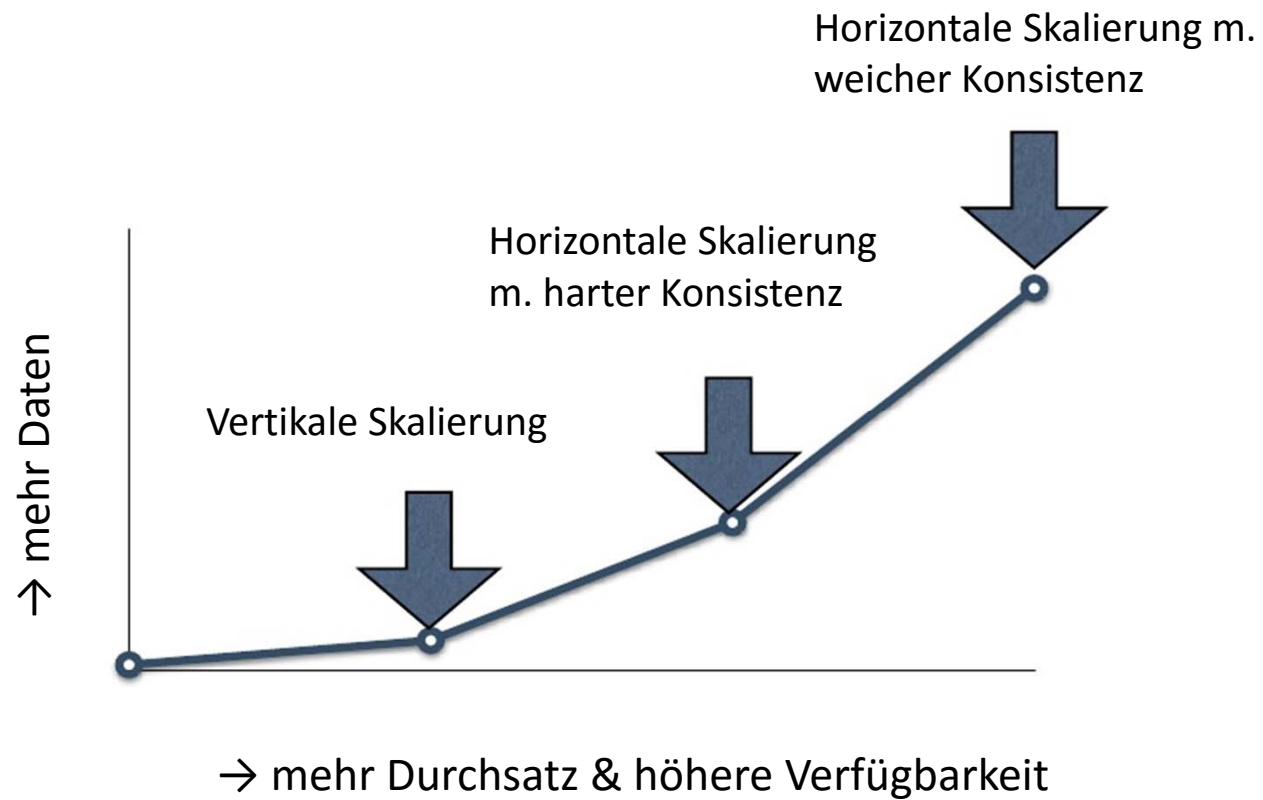
BASE vs. ACID

- › weiche Konsistenz
- › Verfügbarkeit
- › best effort
- › NÄhrungen akzeptabel
- › einfache Entwicklung
- › schneller
- › harte Konsistenz
- › Isolation
- › commit;
- › Verfügbarkeit?
- › komplexe Entwicklung(Schema)
- › sicherer

NoSQL Eigenschaften (revisted)

- › Selten ACID
- › Eingeschränkte Transaktionen
- › Kein JOIN
- › Kein SQL
- › Schemafrei
- › Skaliert horizontal
- › Replikation

Vorteile



Kategorisierung



› Document Store

- MongoDB, CouchDB

› Key Value

- Apache Hadoop, Riak, Redis, Membase, Amazons Dynamo

› Graph

› Wide Column Store

Kategorisierung

■ Key-value



■ Graph database



■ Document-oriented



■ Column family



Document Store



> Idee

- zusammengehörige Daten strukturiert in einem Dokument speichern

> Vorteile

- Struktur der Daten kann unterschiedlich sein, keine Schemen (Programmierung)
- Keine Relationen zwischen Tabellen
- Dokumente können gleiche oder unterschiedliche Schlüssel mit beliebigen Werten besitzen
- Einfaches Hinzufügen neuer Felder

Document Store



› Beispiele

- Cassandra (Apache, Java, JSON Format)
- CouchDB (Apache, JSON over REST/HTTP, limited ACID)
- Informix (IBM, RDBMS with JSON)
- MongoDB (MongoDB, Inc, C++, GNU AGPL)
- RavenDB (Hibernate Rhinos LTD, C#, JavaScript)

Document Store

- › speichern von "Texten" beliebiger Länge mit unstrukturierten Informationen

```
"Vorname": "Wallace"
```

```
"Adresse": "62 West Wallaby Street"
```

```
"Interessen": [ "Käse", "Cracker", "Mond" ]
```

- › Die gespeicherten Dokumente müssen nicht die gleichen Felder enthalten!
- › Abfragen (Views) = Javascript-Funktionen

Key/Value Store

> Idee

- Konzept der Konfigurationsdateien
- Zuordnung eines Werts zu einem Schlüssel
- z.B.: ip=172.16.59.252

> Vorteile

- Schneller Zugriff, hohe Datenmengen
- Wert: String, Integer, Liste, Menge usw.
- Relationen ► Programmierung
- Verteilung auf mehrere kleinere Server

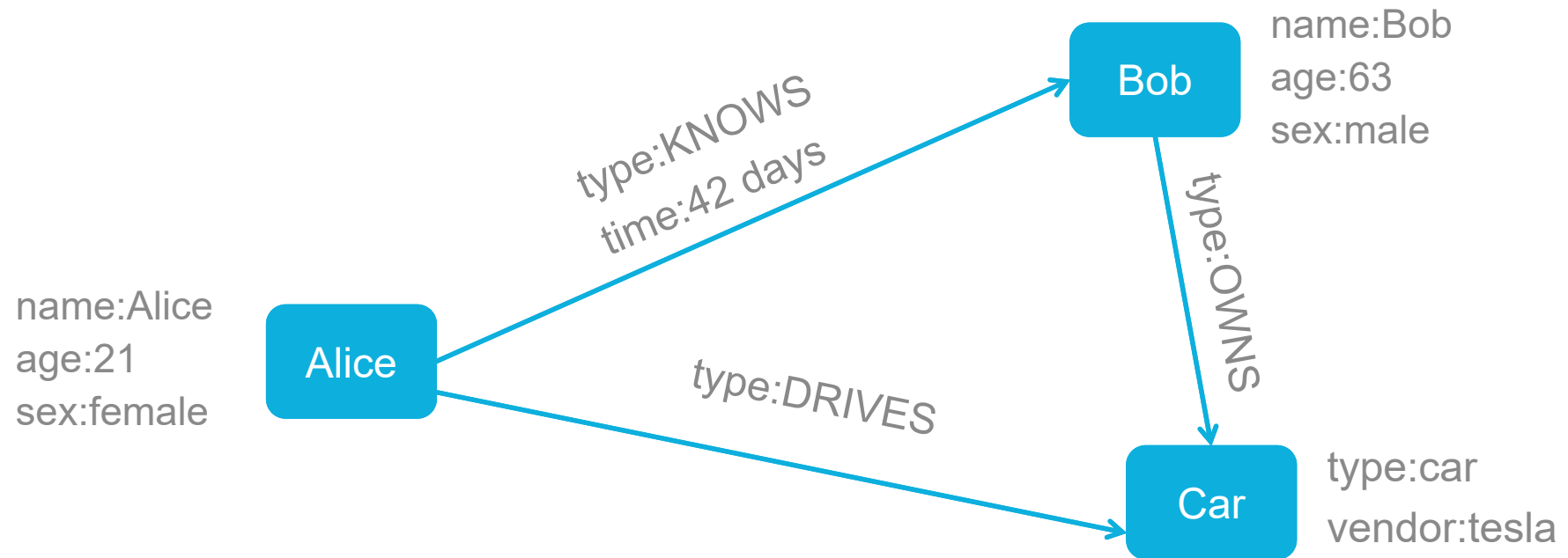
Key/Value Store

- › In-Memory-Variante/On-Disk-Versionen
- › Embedded-Datenbanken (Unix) dbm, gdbm und Berkley DB
- › Redis
 - Strings, Listen, Sets
- › Cassandra
 - spaltenorientiert (spezielle Listen, z.B. Verkaufszahlen) und key-value Ansatz
 - Apache Projekt aus Facebook entstanden
 - Spaltenfamilien, ständig ändernde Datenmengen
- › Riak, Membase

Graph

- › Netz aus miteinander mit Hilfe von Kanten verbundenen Knoten
- › Kanten mit Bezeichnern erweitert -> Netz mit Bedeutung zwischen den Verbindungen (Semantic Web)

Graph



Graph - Skalierung

- › herkömmlichen Datenbanken -> keine Suchen über Relationen (Relationstypen)
- › Daten auf mehrere Server
 - möglicherweise Knoten in zwei Subgraphen aufspalten
- › Problem: Pareto-Regel gilt auch hier
 - 20% der Knoten besitzen 80% Relevanz
 - Welche Abfragen werden an Graphen gestellt?
 - Welche Verbindungen von besonderer Bedeutung?

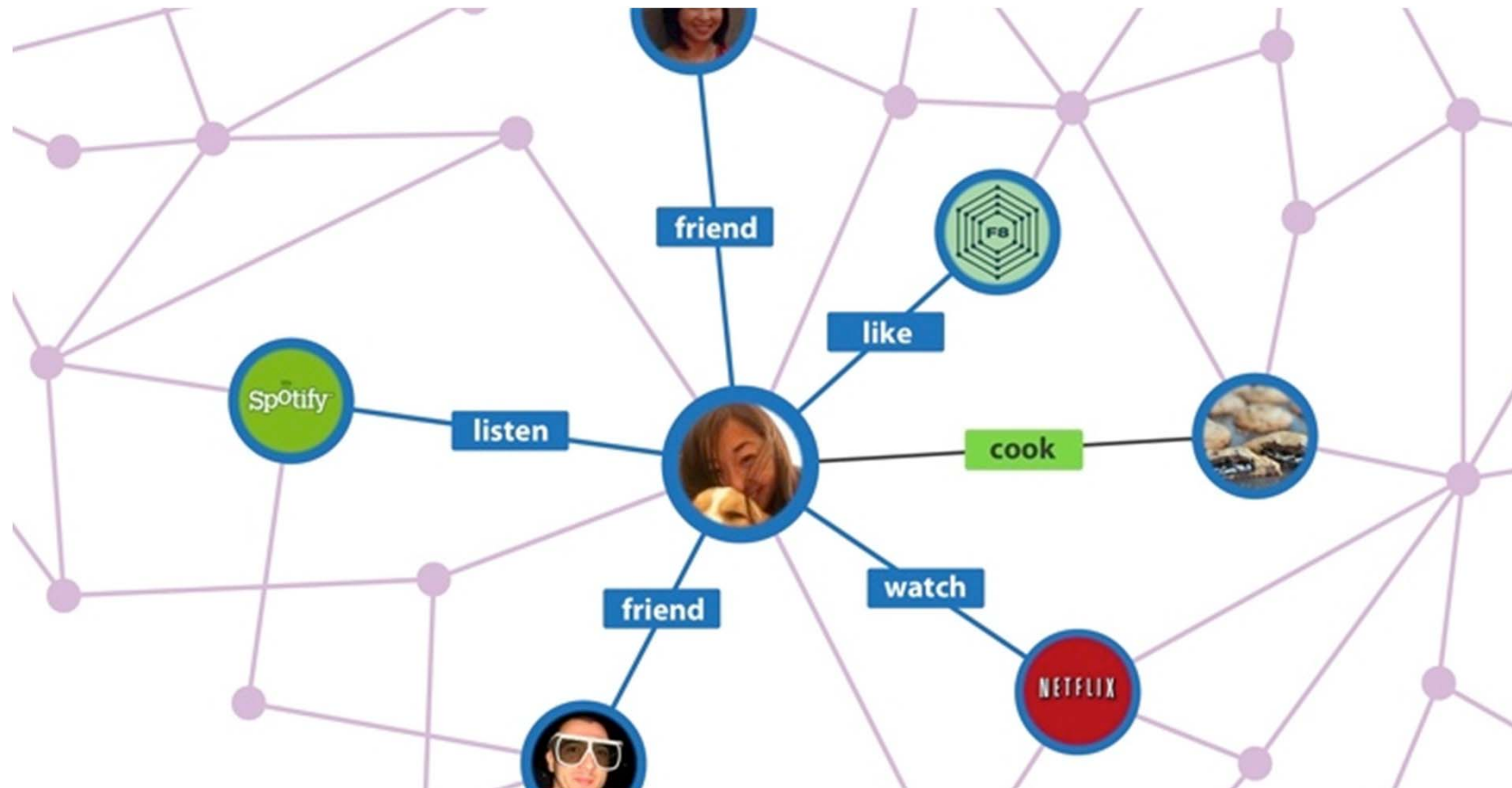
Graph - Einsatz

- › komplex vernetzte Daten (entspricht vielen Join-Operationen)
- › GIS-Anwendungen (Geographic Information System),
 - Navigation zwischen zwei Orten
- › Soziale Netze
 - › Wer kennt wen?
 - › Über welche Wege stehen zwei Entitäten (Knoten) in Beziehung

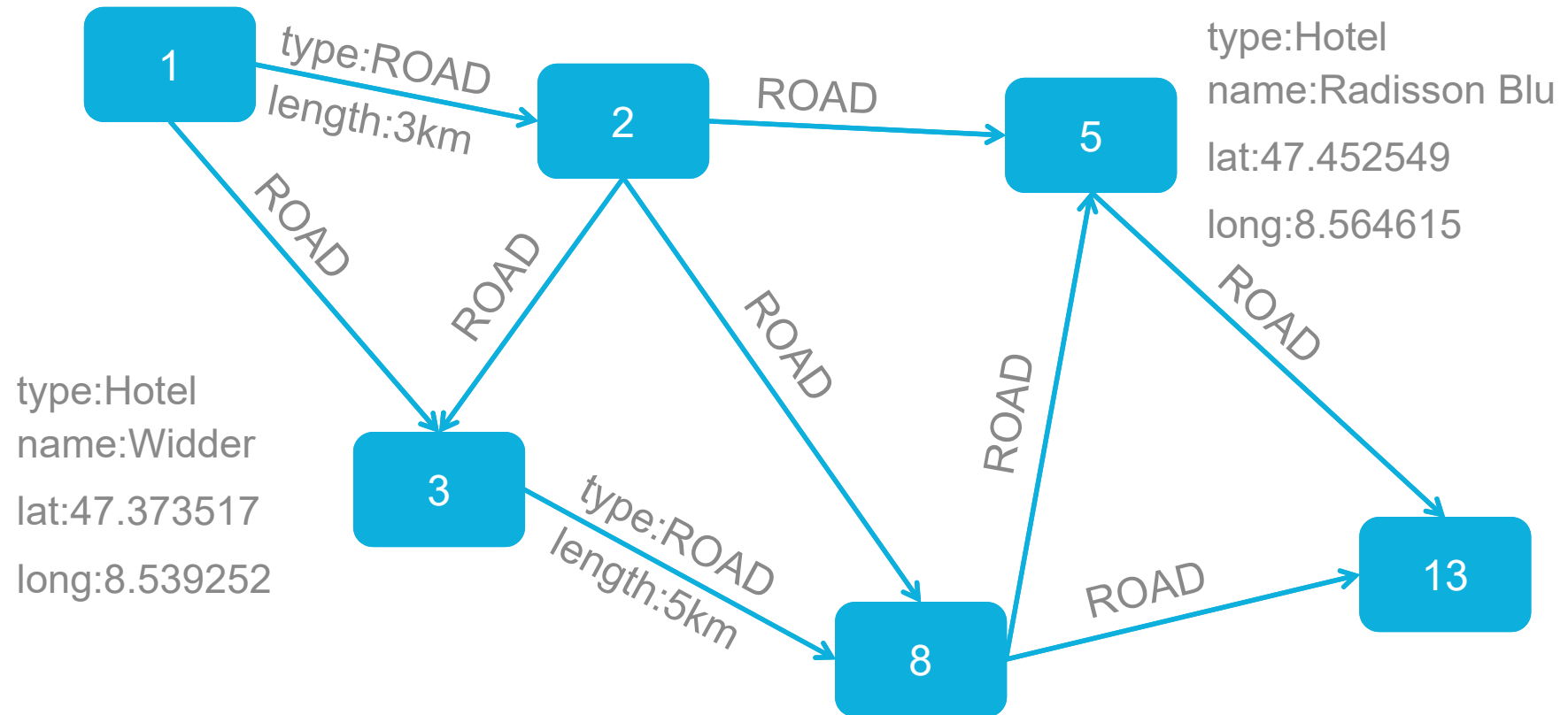
Graph - Einsatz

- › komplex vernetzte Daten (entspricht vielen Join-Operationen)
- › Beispiele: Neo4J, OrientDB

Soziale Netze

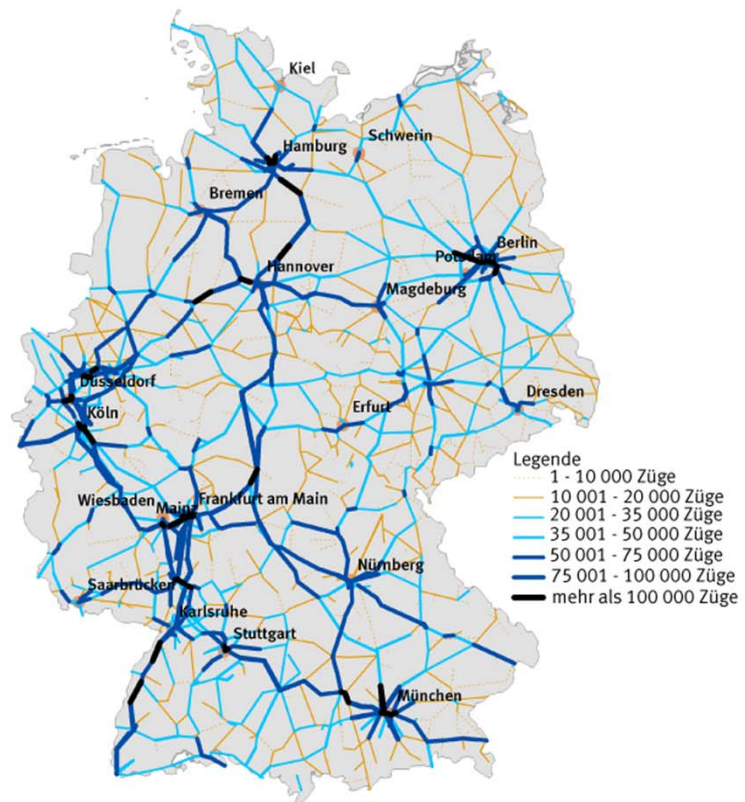


GIS (Geographic Information Systems)



Deutsches Schienennetz

Karte 1: Belastung des Schienennetzes der Eisenbahnen, insgesamt 2005



© Statistisches Bundesamt Deutschland 2007

Graphendatenbanken – Neo4J

- › Die verbreitetste Graphendatenbank der Welt
- › Robust: In Betrieb seit 2003
- › AGPLv3 Community Edition – OpenSource
- › Advanced/Enterprise – für kommerzielle Anwendungen

Graphendatenbanken – Neo4J

- › Objektorientiert, flexible Netzwerkstruktur
- › Support für ACID Transaktionen
- › Horizontal skalierbar
- › Java API
 - Anbindung mittels Java, Groovy, Scala, JRuby
- › Runtime
 - Standalone Server
 - Embedded Database

Wide Column Store (spaltenorientiert)

- › Speicherung von Daten mehrerer Einträge in Spalten anstatt in Zeilen
- › Spalten mit ähnlichen oder verwandten Inhalten, -> „Column Family“ (analog Tabelle). Es gibt keine logische Struktur in der Column Family.
- › Millionen von Spalten -> Wide Columns
- › Beispiele:
 - Cassandra, Apache Hbase, Amazon SimpleDB

Wide Column Store (spaltenorientiert)

Personalnr	Nachname	Vorname	Gehalt
1	Schmidt	Josef	40000
2	Müller	Maria	50000
3	Meier	Julia	44000

```
1, Schmidt, Josef, 40000; 2, Müller, Maria, 50000; 3, Meier, Julia, 44000;
```

versus

```
1, 2, 3; Schmidt, Müller, Meier; Josef, Maria, Julia; 40000, 50000, 44000;
```

Wide Column Store (spaltenorientiert)

- › Wann ist die spaltenorientierte Speicherung ein Vorteil?

Vorteil? Ja/Nein

- › `SELECT SUM(Gehalt) FROM tabelle;`
- › `UPDATE tabelle SET Gehalt = Gehalt * 1.03;`
- › `SELECT * FROM tabelle WHERE Personalnr = 1;`
- › `INSERT INTO tabelle (Personalnr, Nachname, Vorname, Gehalt) VALUES (4, Maier, Karl-Heinz, 45000);`

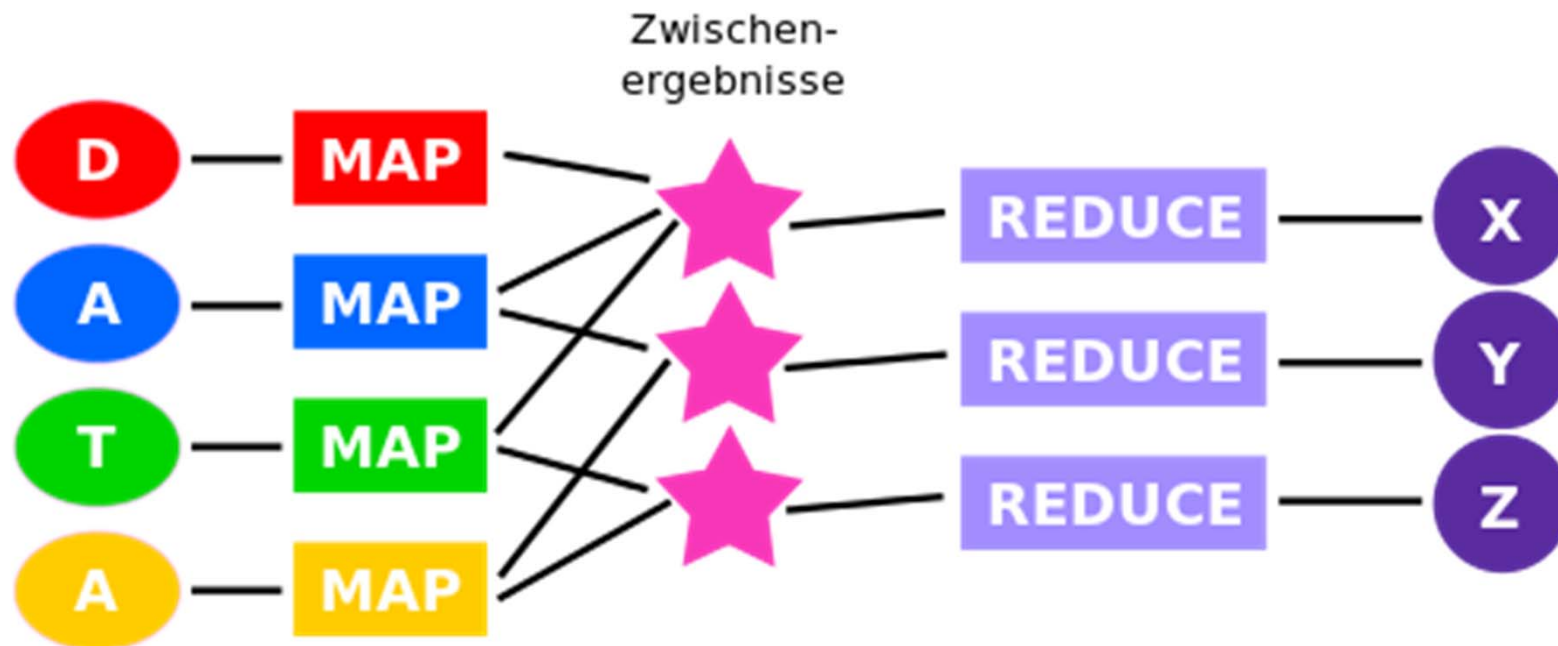
Map/Reduce

- › Programmiermodell von Google entwickelt
- › MapReduce dient zur verteilten und parallelen Verarbeitung großer Mengen strukturierter und unstrukturierter Daten
- › Divide-and-Conquer-Ansatz = verteilte Berechnungen über große Rechnercluster

Map/Reduce

„Wir möchten alle Bücher in der Bibliothek zählen. Sie zählen Regal Nr. 1, ich Regal Nr. 2. **Dies entspricht dem Vorgang „map“ (erfassen).** Jetzt addieren wir unsere beiden Zählungen. **Dies entspricht dem Vorgang „reduce“ (reduzieren).**“

Map/Reduce



- › Entnommen aus http://de.wikipedia.org/wiki/MapReduce#Beispielhafte_Berechnung

MapReduce – Beispiel

„Das Lied von der Glocke“,
1799 von Friedrich Schiller



http://de.wikipedia.org/wiki/MapReduce#Beispiel:_Verteilte_H.C3.A4ufigkeitsanalyse_mit_MapReduce

Zu guter Letzt: Vorteile klassisches relationales DBMS

- › Reife des Systems
 - Seit drei Jahrzehnten am Markt
- › Kompatibilität:
 - standardisierte Schnittstellen, einheitliches relationales Datenmodell
 - Klare Trennung von der Anwendung
- › Konsistenzerfordernisse
 - Absichern der Datenkonsistenz (Transaktionssicherheit oder Fremdschlüsselbeziehungen) von der Datenbank selbst unterstützt
- › Mächtigkeit der Abfragen
 - komplexe Abfragen fassen verschiedene Tabellen zusammen
 - filtern, gruppieren und sortieren
 - Wenig Applikationscode

Quellen

- › <http://de.wikipedia.org/wiki/ACID>
- › <http://www.heise.de/open/artikel/NoSQL-im-Ueberblick-1012483.html>
- › <http://de.scribd.com/doc/30637338/ACID-vs-BASE-NoSQL-erklart>
- › <http://www.heise.de/developer/artikel/Neo4j-NoSQL-Datenbank-mit-graphentheoretischen-Grundlagen-1152559.html>
- › <http://docs.neo4j.org/chunked/stable/tutorials-java-embedded.html>
- › http://de.wikipedia.org/wiki/Spaltenorientierte_Datenbank
- › http://wikis.gm.fh-koeln.de/wiki_db/Datenbanken/SpaltenorientierteDatenbank
- › <https://news.ycombinator.com/item?id=2849163>