Zusammenfassung Soziotechnische Informationssysteme

Six degrees of separation [small world]:

Experiment von Milgram: Jeder Mensch ist über 6 Kanten (first-name basis), also Freunde/Bekannte mit jedem anderen verbunden. [hier: Erreiche einen Politiker in Boston nur über first-Name Basis]

Strong ties:

Starke überlappende Interessen, viel Kommunikation (gut Freunde/Familie)
Sind sozial sehr wichtig, für soziale Netze allerdings unwichtiger, da diese Bekanntschaften auch ohne Netze gepflegt werden.

Weak ties:

Grobe Bekanntschaften/ehemals Bekannte (z.B. alte Schulkammeraden etc.) zu denen man ohne soziale Netze fast keinen oder überhaupt keinen Kontakt mehr hätte.
Sehr wichtig zur Ausbildung sozialer Netze. Weak ties bieten Anhaltspunkt für Werbung/Kontaktknüpfung, die eine Person ohne „Empfehlung" von sozialen Netzen nicht erreichen würden.
Weak ties bilden Brücken zwischen den Clustern/Cliquen der strong ties.

Absent Ties:

Keine oder unwesentliche Beziehungen, können vernachlässigt werden.

Triadic closure (nach Simmel, Clustering Triaden):

Wenn A B kennt und B C, dann kennt A höchstwahrscheinlich auch C. Eine sehr häufige Struktur in sozialen Graphen

Clustering Koeffizient:

Geschlossene Triaden: Drei Knoten, die mit drei Kanten verbunden sind
Offene Triaden: Drei Knoten, die mit nur zwei Kanten verbunden sind
Clustering Koeffizient: $\frac{\#Geschlossene}{\#Geschlossene+\#Offene}$

Preferential Attachment:

„Reiche werden reicher" -> Ein Knoten mit vielen Kanten hat eine hohe Chance weitere Kanten auszubilden

Barabasi-Albert Graph (PA & 80/20 Regel):

Skalenfrei: Kantengrade der Knoten entspricht Potenzgesetz
Gegeben: Initiale Knotenmenge >1 Kantengrad mind. 1
Generierung:

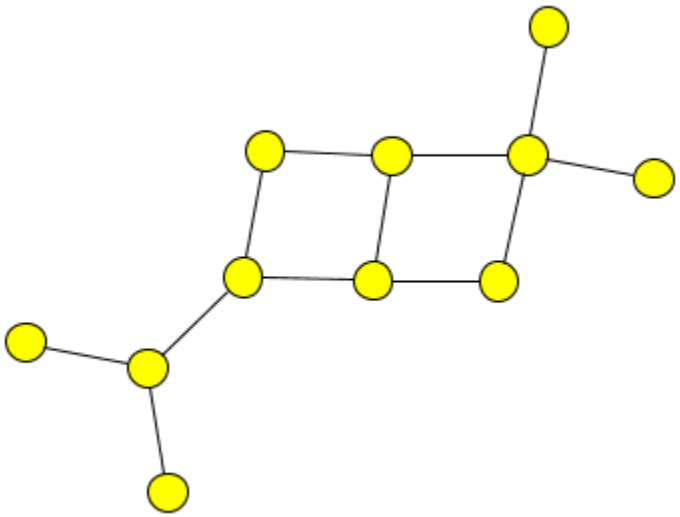
1. Füge einen neuen Knoten hinzu
2. Berechne für jeden Knoten die Chance eine neue Kante zu bekommen:
   $p_j = \frac{k_j}{\sum_i k_i}$ mit k=Kantengrad
   Knoten mit vielen Kanten haben also eine höhere Chance eine neue Kante zu bekommen
3. Ordne jedem Knoten einen Bereich zwischen 0 und 1 zu (abhängig von dessen p_j)

4. Würfle eine Zufallszahl zwischen 0 und 1 und erstelle eine Kante zwischen dem neuen und dem der Zufallszahl zugeordneten Knoten

GraphML:

Beschreibung von Graphen in XML
Beispiel:

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
     http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">
  <graph id="G" edgedefault="undirected">
    <node id="n0"/>
    <node id="n1"/>
    <node id="n2"/>
    <node id="n3"/>
    <node id="n4"/>
    <node id="n5"/>
    <node id="n6"/>
    <node id="n7"/>
    <node id="n8"/>
    <node id="n9"/>
    <node id="n10"/>
    <edge source="n0" target="n2"/>
    <edge source="n1" target="n2"/>
    <edge source="n2" target="n3"/>
    <edge source="n3" target="n5"/>
    <edge source="n3" target="n4"/>
    <edge source="n4" target="n6"/>
    <edge source="n6" target="n5"/>
    <edge source="n5" target="n7"/>
    <edge source="n6" target="n8"/>
    <edge source="n8" target="n7"/>
    <edge source="n8" target="n9"/>
    <edge source="n8" target="n10"/>
  </graph>
</graphml>
```



Problem von großen Graphen: In Memory nicht abbildbar (z.B. 10^8, 10^9)

SOAP:

XML basierte API (Simple Object Access Protocol)

Häufig verwendet, nutzt XSD / XSLT / XML Schema, daher leicht verifizierbar. Mit viel Overhead, daher häufig abgelöst von REST.

REST:

JSON basierte API (Representational state transfer)
Verwendet http Operationen

- GET für Anfragen
- POST für Erstellen
- PUT für Ändern
- DELETE für Löschen

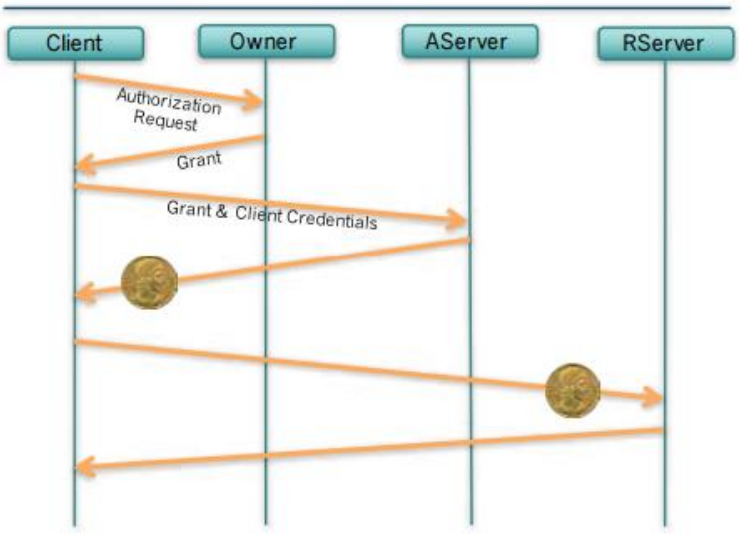
Jedes Objekt/State hat eine eigene URL

Unterschied zu anderen XML RPC Servics (z.B. mit SOAP): Leichter zu implementieren, verbraucht weniger Ressourcen, meist einzelne unabhängige REST Mini Services statt riesigem SOAP Blob mit IF Schleife zur Zuordnung der Anfragen – aber: nicht so einfache Constrains oder Checks möglich wie z.B. die eingebauten und vorgegebenen durch XSD bei SOAP.

Sicherheit:

Authentifizierung: Wer bin ich? (z.B. mit OpenID lösen)
Autorisierung: Darf ich das? (z.B. mit OAuth machen)
Password-Antipattern: In er Datenbank sein Passwort abspeichern, damit jemand damit im Namen des Passwort Inhabers handeln kann – mit vollen Rechten, nicht als Vertreter. Sehr sicherheitskritisch.
OAuth:
Authorisationsprotokoll (darf ich das?)
Erlaubt Zugriff einer Ressource auf Daten/Funktionen eines anderen Service. Dabei wird ein externer Authorisationsservice genutzt, der ein Token für diese Ressource ausstellt.
Cross-Authentifizierung (Passwort antipattern)

Man kann meist (siehe z.B. FB) feingranular Rechte vergeben, im Gegensatz zur „Übergabe des Passwortes“.



Owner: Besitzer im digitalen Gut (Nutzer)

RServer: Stellt digitales Gut bereit (Anwendungsprovider)
Client: Will auf diese Infos zugreifen (z.B. Nutzer Namen bei FB)
AServer: Leitet das Access Token an den Client weiter, nachdem der Owner sich selbst authentifiziert hat und das Token bestätigt hat / Zugriffsrechte. Danach kann der Client auf die freigebenden Daten zugreifen.

Access Token ist dabei meist nur eine Zeichenkette ohne innere Struktur und legt die Zugriffsdetails fest – Art des Zugriffs und die zeitliche Begrenzung.

Mit einem Refresh Token kann der Client selbstständig eine Verlängerung des Access Tokens ohne Owner Grant anfordern.

Hadoop:

Name Node: Verwaltet Data Nodes, bearbeitet eingehende Anfragen, legt den Replikationsgrad fest
Data Node: Speichern die Daten, führen MapReduce aus
Job Tracker: Sammelt Ausgaben der Data Nodes und aggregiert sie
HFS (Hadoop Distributed File System) kennt den Ort von Daten -> es wird die nächste Node gewählt, ideal auf demselben Rechner

MapReduce:

Soll möglichst stark horizontal skalierbar sein
Funktioniert sehr gut bei semistrukturierten Daten (Key/Value)
Map Phase: Auf jeder Node werden Paare herausgefiltert, die einer Suchanfrage entsprechen
Reduce Phase: Auf einer oder wenigen Nodes werden diese gefundenen Paare vereint/verrechnet

HBase:

Basiert auf Hadoop
Erlaubt im Gegensatz zu Hadoop schnellen Zugriff auf einzelne Records
Matrix Schlüssel x Daten (Eine große Tabelle „Bigtable“)

SQL/NOSQL:

SQL: Tabellen, Transaktionen. Verteilung nur über durch Kopieren von Tabellen erreichbar.
Üblicherweise weniger als 10 Nodes, da JOINS schlecht horizontal Skalieren
NOSQL: Semistrukturierte Daten, leicht horizontal skalierbar
NOSQL besitzt kein Schema (Key-Value,Document,Column-Family,Graph)

Key-Value: Bekannt, ein Schlüssel und ein dazugehöriger Wert z.B.: Cassandra (Hadoop), MongoDB, Redis
Document: Teilstrukturiert, JSON, Markup etc. : CouchDB (JSON), MongoDB (JSON), OrientDB, SimpleDB
Column-Family: Große Tabelle: BigTable Accumolo, HBase etc.
Graph: Speichern Graphen als Datenmodell, Graph Query Languages: Neo4j, OrientDB, FlockDB (Scala)

Polyglot Persistence:

ACID (in Teilen) und SQL werden „miteinander vermischt“ für eine neue Form Namens NewSQL. (Wechselseitige Synergieeffekte zwischen traditionellen Datenbanken und No-SQL Systemen)

BASE vs ACID:

ACID:

- Atomicity: Unteilbarkeit von Transaktionen, entweder es wird alles ausgeführt oder nichts
- Consistency: Datenkonsistenz bleibt auch bei fehlgeschlagenen Transaktionen erhalten
- Isolation: Gleichzeitige Abfragen beeinflussen sich nicht gegenseitig (lock)
- Durability: Nach Abschluss einer Transaktion sind die Daten dauerhaft gespeichert, nach Systemausfall reproduzierbar

BASE:

- Basically Available: Sollte immer erreichbar sein, Ausfälle sind aber eingeplant
- Soft-State: Eine Node muss die Daten nach einem Ausfall nicht selbst wiederherstellen können, es wird davon ausgegangen, dass andere Nodes diese Daten zur erneuten Replikation zur Verfügung stellen
- Eventual Consistency: Es kann mehrere verschiedene Replikate geben, aber zu irgendeinem Zeitpunkt werden sie sich synchronisiert haben und konsistent sein

CAP:

Consistency: Falls es mehrere Kopien gibt, sind diese gleich
Availability: Alle Operationen liefern letztendlich ein Ergebnis
Partition Tolerance: Das System ist auch im Falle einer Partitionierung komplett einsatzbereit
Es können immer nur 2 dieser 3 Eigenschaften vorhanden sein.
CA: Vertikale Skalierung, Partitionierung heißt Ausfall – Klassische DBS
AP: Hauptsache verfügbar, hoher Replikationsgrad, Inkonsistenzen akzeptieren – DNS,NoSQL
CP: Hauptsache Konsistent, es wird auf alle Knoten gewartet – Geldautomaten

Vertikale Skalierbarkeit:

Ein Server, der aber sehr stark

Horizontale Skalierbarkeit:

Bedeutet extrem viel kleinere, schwächere Systeme über Netzwerk zusammenzuhängen und an einem Problem verteilt arbeiten zu lassen. Die Erweiterbarkeit ist hier meist sehr einfach.

Was bedeutet die Aussage „The Medium ist he Message“ von Marshall McLuhan im Zusammenhang mit Soziotechnischen Systemen

Medium dient der Übertragung von Informationen. Laut McLuhan ist ein Medium schon selbst sinntragend, beeinflusst mehr als die Nachricht selbst die übertragen werden soll. Im konkreten Fall ist die Übertragung einer gedruckten Nachricht mittels Schiff viel langsamer als per Satellit direkt auf einen ganzen Kontinent. Beschleunigte Nachrichtenübertragung führen zu einem „Global Village“ und FB und Co verkörpern dies perfekt da sie in sekundenbruchteilen Informationen auch viral übertragen können.

Warum sollt man auf einer Plattform zum kollaborativen Entwickeln (Github) anderen Entwicklern folgen?

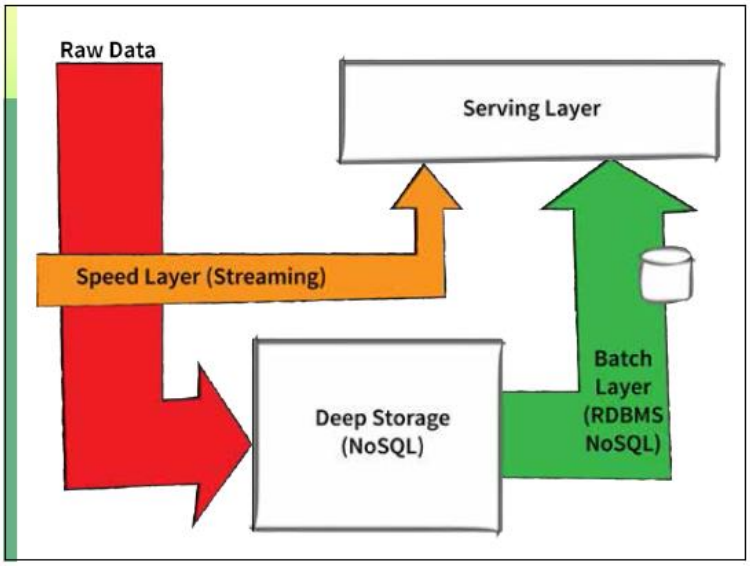
Die sogenannten „Rockstars“ setzen zum einen Trends und lassen frühzeitig Entwicklungsrichtungen sehen (was man zukünftig können sollte), zum anderen können sie sehr schnell Projekte empfehlen, die einen selbst interessieren könnten. Weiterhin kann man dadurch, dass man z.B. PRs für Projekte dieser Leute stellt schnell in den Fokus rücken und leicht sich bei einer Firma positionieren.

Stream Processing:

Die Verarbeitung von großen Datenmengen die über mehrere Server umgeleitet werden und in jedem Schritt reduziert werden, so dass diese händelbar werden. Ist eine horizontale Spielart, sehr attraktiv, besonders für Log Dateien und ähnliches zu verwenden. (z.B. Am Anfang Sources, dann Processing, dann Sink)

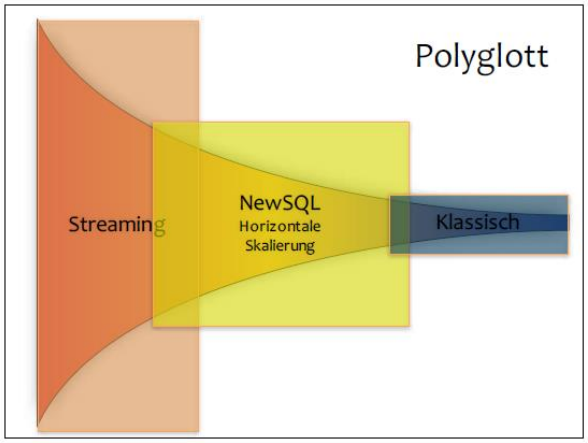
Lambda Architektur:

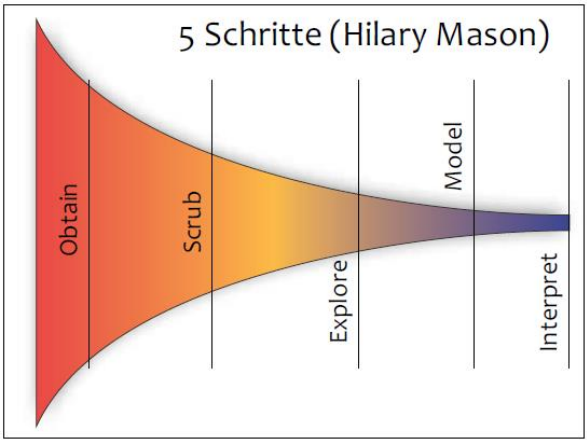
Immutable Data, Queries als Funktion ohne Seiteneffekte formulieren, „Lambda Kalkül“



Big Data:

Am besten unter Verwendung von Polyglotten Systemen eindampfen:





Obtain**:** Daten erfassen – sehr schwer, riesige Datenmengen

Scrub**:** Möglichst viele Daten frühzeitig wegewerfen

Explore**:** Bezug der Daten erkennen / Qualität bewerten / Analysewerkzeuge / Business Intelligence

Model**:** Validierung gegen Testmengen / Verstehen und Vorhersagen

Interpret**:** Verwenden der Daten, transformieren, generalisieren

4vs von Big Data:

- Volume: Datenmenge
- Variety: Unterschiedlichkeit
- Velocity: Mit hoher Geschwindigkeit
- Veracity: Fragwürdigkeit der Richtigkeit

4th Paradigm:

- Empirisch / Experimental Science: Beschreibung
- Theoretisch / Theoretical Science: Modellbildung
- Computer Science / Computational Science: Simulation
- Data-Intensive Science / eScience: Daten Analyse