# Was ist los?

Erhebung und Visualisierung von empirischen Daten verschiedener Internet-Zeitungen

*Halbjahresarbeit 2016/2107 von* Jaro Habiger

# Inhaltsverzeichnis


1 Idee. . . . . 1
2 1 Tasse Tee . . . . . 1
3 Umsetzung . . . . . 1
3.1 Daten sammeln . . . . . 2
3.2 Speicherung . . . . . 5
3.3 MapReduce . . . . . 5
3.4 Datenbank . . . . . 7
3.5 Middleware . . . . . 8
3.6 Visualisierung . . . . . 8
3.7 Zusammenführung . . . . . 8
4 Beobachtungen . . . . . 9
4.1 Evaluation des Verfahrens . . . . . 9
4.1.1 Ergebniss . . . . . 9
4.1.2 Technisch . . . . . 9
4.2 Verschiedene Zeitungen im Vergleich . . . . . 9
5 Fazit . . . . . 9

# 1 Idee

- ich -> Programmieren
- ich -> Zeitung -> klo -> spiegel -> wortwolke
- ich -> politisch
- eig TR website
- wie funktionieren Zeitungen?
- anschpruchsvoll
- -> was denken medien
- anwendungsmöglichkeinten

Relativ schnell war mir klar, das ich mich mit Datenvisualisierung beschäftigen will. Anfangs war allerdings nicht ganz klar, welchen Datensatz ich auf welche Aspekte hin untersuchen will. Eines tages nahm ich dann eine Ausgabe des Spiegels zur Hand und sah auf der ersten seite eine sogenannte Wortwolke. Diese form der Darstellungy

Am Anfang dieser Halbjahresarbeit stand die Idee, die aktuelle Nachrichtenlage visuell sichtbar zu machen. Dies sollte möglichst intuitiv und aufschlussreich sein. Außerdem

# 2 1 Tasse Tee

- Generelle Gedanken
- Vortrag über spon
- Teilweise sehr schwierig zu verstehen -> Datenstrukturen ineinander umwandeln: wenig greifbar

# 3 Umsetzung

Die Umsetzung lässt sich sehr gut in einzelne Teilprobleme unterteilen, was ich bei meiner Umsetzung auch sehr strikt beachtet habe. Hierbei ist es am einfachsten, den Datenfluss von den verschiedenen Nachrichtenquellen zur fertigen Visualisierung zu betrachten. In den nachfolgenden Abschnitten wird dieser Verarbeitungsprozess beschrieben.

toDO: Diagramm

## 3.1 Daten sammeln

Als erstes müssen Daten zur weiteren Verwertung von den verschiedenen Nachrichtenquelln gesammelt werden. Dies geschieht über die sogenannten "RSS-Feeds". Bei diesen handelt es sich um ein standardisiertes Format, über das Nachrichtenanbieter ihre Artikel, inklusive Metadaten wie z.B. den Zeitpunkt der Veröffentlichung, in maschinenlesbarer Form bereitstellen. Im Prinzip werden also die gleichen Daten bereitgestellt wie auf der normalen Internetseite, mit dem Unterschied, dass sie einfacher mit Programmen verarbeitet werden können.

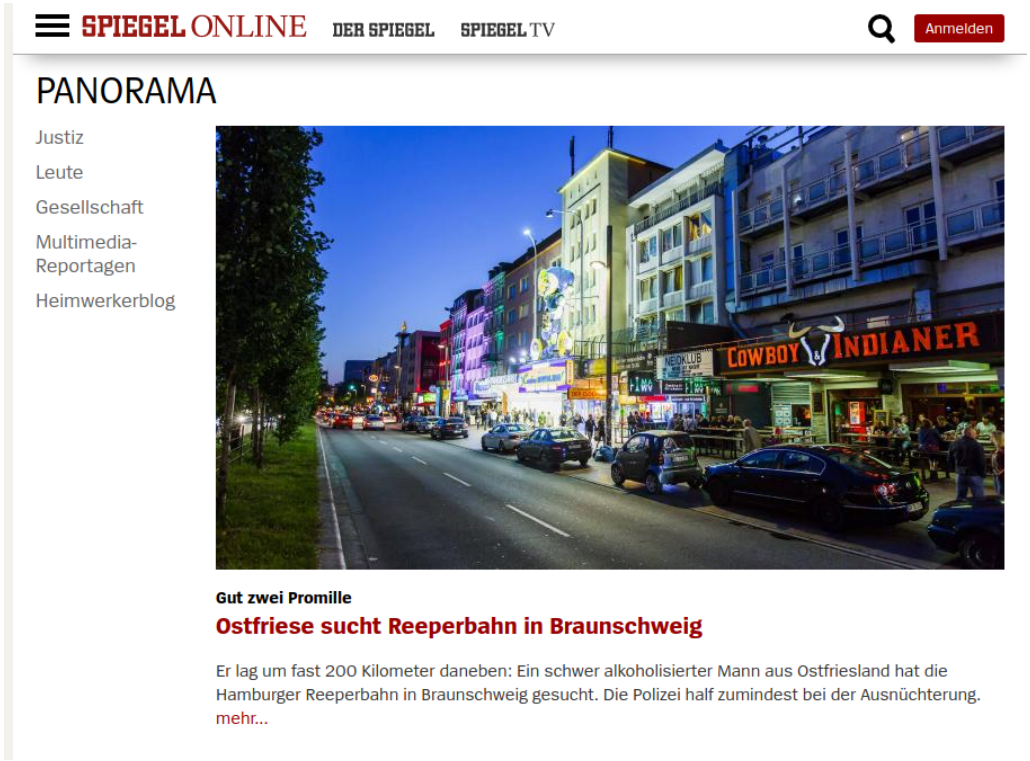
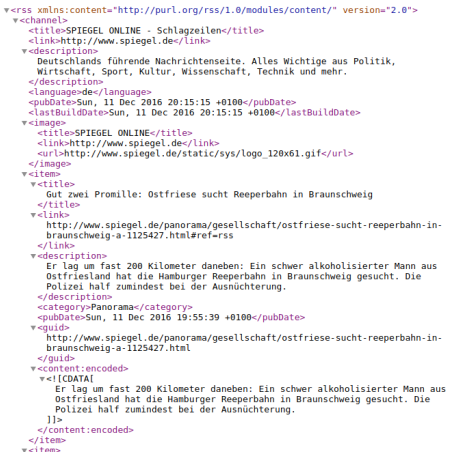



Fig. 1: Ein Artikel einmal in der normalen Darstellung...

Diese Darstellung als RSS-Feed ermöglicht es, die Artikel verschiedener Online-



```
<rss xmlns:content="http://purl.org/rss/1.0/modules/content/" version="2.0">
  <channel>
    <title>SPIEGEL ONLINE - Schlagzeilen</title>
    <link>http://www.spiegel.de</link>
    <description>
      Deutschlands führende Nachrichtenseite. Alles Wichtige aus Politik,
      Wirtschaft, Sport, Kultur, Wissenschaft, Technik und mehr.
    </description>
    <language>de</language>
    <pubDate>Sun, 11 Dec 2016 20:15:15 +0100</pubDate>
    <lastBuildDate>Sun, 11 Dec 2016 20:15:15 +0100</lastBuildDate>
    <image>
      <title>SPIEGEL ONLINE</title>
      <link>http://www.spiegel.de</link>
      <url>http://www.spiegel.de/static/sys/logo_120x61.gif</url>
    </image>
    <item>
      <title>
        Gut zwei Promille: Ostfriese sucht Reeperbahn in Braunschweig
      </title>
      <link>
        http://www.spiegel.de/panorama/gesellschaft/ostfriese-sucht-reeperbahn-in-
        braunschweig-a-1125427.html#ref=rss
      </link>
      <description>
        Er lag um fast 200 Kilometer daneben: Ein schwer alkoholisierter Mann aus
        Ostfriesland hat die Hamburger Reeperbahn in Braunschweig gesucht. Die
        Polizei half zumindest bei der Ausnüchterung.
      </description>
      <category>Panorama</category>
      <pubDate>Sun, 11 Dec 2016 19:55:39 +0100</pubDate>
      <guid>
        http://www.spiegel.de/panorama/gesellschaft/ostfriese-sucht-reeperbahn-in-
        braunschweig-a-1125427.html
      </guid>
      <content:encoded>
        <![CDATA[
          Er lag um fast 200 Kilometer daneben: Ein schwer alkoholisierter Mann aus
          Ostfriesland hat die Hamburger Reeperbahn in Braunschweig gesucht. Die
          Polizei half zumindest bei der Ausnüchterung.
        ]]>
      </content:encoded>
    </item>
    <item>
```


Fig. 2: ... und einmal als Teil eines RSS-Feeds

Zeitungen zu betrachten, ohne für jede Zeitung einen komplett neuen Datensammler programmieren zu müssen.

Der letztendliche Datensammler ist ein Programm, welches ich in Python geschrieben habe. Python ist eine Programmiersprache, die auf Einfachheit und Flexibilität optimiert ist. Das Datensammelprogramm lädt sich den RSS-Feed einer einzelnen Nachrichtenquelle periodisch herunter und verarbeitet ihn weiter. Ich habe mich dazu entschieden, jeden RSS-Feed alle 10s neu zu analysieren. Im ersten Schritt der Verarbeitung lädt das Programm den Feed als Datei von den Servern der jeweiligen Online-Zeitung herunter. Diese Datei ist eine sogenannte XML-Datei. In ihr werden Daten als Baumstruktur abgebildet. Hierbei muss man sich die Datei als “Stamm” des Baums vorstellen. Die ersten “Äste” der Baumstruktur beinhalten die Metadaten, wie z.B. den Erstellungszeitpunkt und den Herausgeber des Feeds. Die nachfolgenden “Äste” enthalten jeweils einen Artikel. Diese Artikel wiederum beinhalten verschiedene Unterelemente, d.h. die “Äste” verzweigen sich in weitere, kleinere “Ästchen”. In diesen stehen nun z.B. der Autor des Textes, der Titel, oder eben der eigentliche Inhalt des Textes. Diese textuelle Repräsentation einer Baumstruktur gilt es nun in eine einfacher verwendbare Repräsentation im Speicher des Python Programms umzuwandeln. Hierfür wird eine Programmbiliothek verwendet, die einen sogenannten XML-Parser beinhaltet, der genau dies erreicht. Nach diesem Schritt können die einzelnen Artikel betrachtet werden. Hierbei wird zuallererst überprüft, welche Artikel schon

einmal verarbeitet wurden. Diese werden verworfen. Die übriggebliebenen, also neuen Artikel werden in eine allgemeinere Repräsentation für Neuigkeiten und ihre Metadaten gebracht, die ich mir überlegt habe. Diese ist auch wieder eine Baumstruktur und sieht aus wie in **??**ir).


Fig. 3: Die Zwischenrepräsentation der Artikel

Diese Umwandlung ist nötig, da der RSS-Standard zwar die grobe Struktur und ihre Repräsentation als XML-Datei spezifiziert, letzten Endes jeder Nachichtenanbieter allerdings doch nicht ganz kompatible Feeds ausliefert. Diese kleinen Unterschiede werden hier also angeglichen, damit die weitere Verarbeitung leichter vonstatten gehen kann, und in der weiteren Verarbeitung keine Unterschiede mehr beachtet werden müssen.

Ein weiteres Problem der RSS-Feeds ist, dass sie nur Kurzfassungen der Artikel enthalten. In der Umwandlung müssen also noch die vollständigen Artikel von der Internetseite des Nachrichtenanbieters heruntergeladen und der Volltext aus dieser extrahiert werden. Hierzu wird zuerst das HTML-Dokument der Seite “geparsed”. Das Wort “parsen” ist sehr schwierig ins Deutsche zu übersetzen. Es beschreibt den Sachverhalt, die textuelle Repräsentation eines strukturierten Datensatzes in die “native” Repräsentation einer Programmiersprache für diesen Datensatz zu bringen. Die “native” Repräsentation eines Datensatzes in einer Programmiersprache ist die Standardform, diesen Datensatz abzubilden. Diese hat oft bestimmte Funktionen, die das Programmieren vereinfachen, wie z.B. die Zugriffsmöglichkeit auf einzelne Elemente bei Listen.

Aus dem "geparsten" HTML-Dokument werden nun die relevanten Teile, zu denen z.B. nicht die Kopf- oder Fußzeile gehören, mithilfe sogenannter "CSS-Selektoren" identifiziert. Hiernach werden alle HTML-Elemente dieser Bereiche in reinen Text umgewandelt.

Architektonisch ist der Datensammler ein Programm, in das weitere kleinere Module "hereingesteckt" werden, die die einzelnen Nachrichtenseiten ansprechen.

Am Ende der Datenaquirierung werden die gesammelten Daten an die nächste Stufe weitergegeben, in der die Daten gespeichert werden.

## 3.2 Speicherung

Die nun gesammelten Daten müssen vor der weiteren Verwendung ersteinmal strukturiert zwischenggespeichert werden. Dies geschieht in einer Datenbank. Ich habe mich dafür entschieden, eine sogenannte "NoSQL-Datenbank" zu verwenden, da diese flexiblere Abfragemethoden ermöglichen. "NoSQL" steht hierbei dafür, dass die Datenbank nicht über die Datenbanksprache SQL angesprochen wird, wie es normalerweise der Fall ist, sondern eine andere Abfragemöglichkeit bietet. Relativ früh habe ich mich dafür entschieden, dass ich meine Datenbankabfragen als "MapReduce"-Funktionen formulieren möchte.

## 3.3 MapReduce

Das MapReduce Verfahren hat einige große stärken, wie z.B. die hohe parallelisierbarkeit und die damit verbundene hohe geschwindigkeit, bei gleichzeitig hoher flexibilität. MapReduce ist ein von der firma Google eingeführtes Programmiermodell, welches wie folgt funktioniert:

1. Am Anfang des Przesses steht eine Menge aus N Eingangsdatensätzen. In meinem Fall sind das die Zeitungsartike, die als Objekte mit der oben beschriebenen Datenstruktur vorliegen. Für jeden dieser Artikel wird jetzt die map funktion ausgeführt. Diese ordnet jedem Artikel N Schlüssel-Wert Paare zu. In dem Fall, das wir die zu jedem vorkommenden Wort die Häufigkeit bestimmen wollen, wordnet die map Funktion

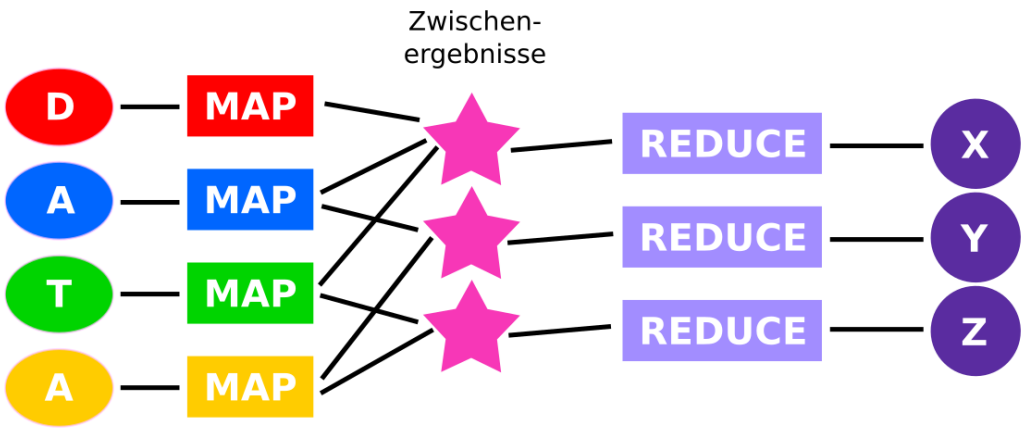



Fig. 4: Der MapReduce prozess als Grafik

also jedem Artikel die Menge der Darin enthaltenen Wörter zu. Da diese Funktion auf jeden Artikel andgewnd wird, kann man sich in diesem Fall den gesammten map Prozess als eine Funktion vorstellen, in die die die Menge aller Artikel der Menge der darin enthaltenen Worte zuordnet. Der hierzu zugerhöige Code der map funktion ist:

```
function map() {
var text = this.text.replace(/[^A-Za-zÄäÖöÜüß ]/g, " ");  // entferne alle üb
var words = text.split(" ");  // Teile den text in Wörter
for (var i = 0; i < words.length; i++) {  // für jedes Wort
    var word = words[i];
    if(word) {
      emit(word, 1);  // füge der Ausgabemenge das Wort hinzu
    }
}
}
```

2. Im nächsten Schritt werden Elemente mit Gleichen Schlüsseln Gruppiert. Nach diesem Schritt also eine Menge aus Schlüssel-Wertmengenpaaren (Auch wenn ich immer wieder das Wort Menge verwede, meine ich eigentlich Multimengen, da es relevent ist, wie oft ein bestimmtes Element in der Menge vorkommt). Dieser Schritt ist, anders als die anderen beiden Schritte bei allen anderen Abfragen gleich. Da in unserem Beispiel das Wort als Schlüssel verwendet wurde Sähe eine Beispielhafte Menge nach

diesem schritt wie folgt aus:

```
{
  "der": {1; 1; 1; 1; 1; 1; 1; 1; 1};
  "die": {1; 1; 1; 1};
  ...
}
```

3. Im dritten und letzten Schritt wird für jeden Schlüssel die sogenannte "reduce" funktion angewandt. Diese reduziert die Mengen, die den Schlüsseln zugeordnet sind auf einen Wert. In unserem Beispiel fällt die reduce-funktion relativ einfach aus, da sie einfach nur die elemente der Menge aufummieren muss:

   ```
   reduce(key, values) {
   return values.reduce((previousValue, currentValue) => currentValue + previous
   }
   ```

4. Der vierte und letzte schritt gehört eigentlich nicht mehr zum MapReduce verfahren. Dieser wird nach diesem ausgeführt und dient dazu die ergebisse zu sortieren und eventuell feinheiten zu verbessern. So ist es zum beispiel möglich selten genutzte- oder sogenannte stoppworte in diesem schritt auszusortieren:

   ```
   function filter(data) {
   return data.filter(word => stopwords.indexOf(word["_id"].toLowerCase()) < 0)
   }
   ```

   Des weiteren wird in diesem schritt versucht, Wörter mit dem gleichen Stamm, und somit mit der gleichen Bedeutung zusammenzuführen, auch wenn diese unterschiedliche endungen Haben. Ein beispiel hierfür ist, das die Wörter "Trump" und "Trumps" zusammengezählt werden. Hierbei wird immer das Kürzeste der Worte behalten, da dies meist die Grundform ist.

## 3.4 Datenbank

Nachdem klar war, wie die Abfragen formuliert werden sollten, habe ich verschiedene Datenbanken in betracht gezogen. Zuerst habe ich angefangen mit der Datenbank "MongoDB" zu arbeite, störte mich aber shr stark an der

Komplexität von dieser. Außerdem bietet MongoDB keine möglichkeit, diese über das HTTP Protokoll anzusprechen, was für die Visualisierung allerdings sehr wichtig ist. Also sah ich mich nach anderen Alternativen um und fand "CouchDB", welche vom Apache Projekt entwicket wird. Diese erfüllte die meisten meiner anforderungen relativ gut, weshalbi ich meinen gesammten bis dahin existierenden Code an CoucDB anpasste. Nach einiger Zeit des Testens stellte sich allerdings heraus, das CouchDB wahrchienlich zu langsam sein würde und ein unpassendes Rechteverwaltungssystem mitbringt, was die Arbeitsersparniss im gegensatz zu MongoDB zu nichte machen würde. In Folge dieser Erkentniss entschied ich mich dazu, meine Software zurück auf MongoDB zu portieren und für diese Eine HTTP schnitschtelle zu schreiben.

## 3.5 Middleware

Diese HTPP API ist in gewissermaßen das Bindeglied zwischen der Datenbank und der Visualisierung, weshalb es im nacholgenden "Middleware" genannt wird. Sie nimmt HTTP anfragen vom Frontend entgegen, leitet diese an die Datenbank weiter und schickt die Antwort an das Frontend zurück. In sofern könnte man sie auch als übersetzer zwischen verschiedenen Protokollen verstehen. Diese ist notwendig, da ich die Visualisierung im Browser implementieren möchte und im Browser nur sehr wenige Protokolle verfügbar sind. HTTP ist eines dieser wenigen Möglichkeiten und bietet scih an, da es verhältnissmäßig einfach zu implementieren ist.

## 3.6 Visualisierung

Ich habe mich entschieden die Visualisierung im Browser zu schreiben. Dies

- d3.js
- Flexibel

## 3.7 Zusammenführung

Um die einzelen Teilkomponenten oder auch "Microservices" zusammenzuführen habe ich Docker verwendet.

- Mehr details

# 4 Beobachtungen

## 4.1 Evaluation des Verfahrens

### 4.1.1 Ergebniss

### 4.1.2 Technisch

- Map/Reduce bei jedem Query skaliert nicht :(

## 4.2 Verschiedene Zeitungen im Vergleich

# 5 Fazit