Projekt: Datenbankparadigmen (MIN-DBP)

Ausarbeitung für das Projekt der Veranstaltung MIN-DBP im Wintersemester 2015/16
umgesetzt mit der Datenbank **CouchDB**

Hannover, 07.01.2016

Ausarbeitung, eingereicht von
Mirco Bartels
Johannes Hartmann
Eike Schwank
Lorenz Surkemper

Dozent
Prof. Dr. Carsten Kleiner

Inhaltsverzeichnis

1 Abstract

Today many IT companies use specialized graph databases to link information. The aim of this project was to try to implement a graph in the NoSQL database CouchDB and to evaluate the results. The project was realised in three steps:

1. Familiarization with the database and presentation
2. System design
3. System implementation

Data from reddit was used to create the graph. To fetch and upload the data to CouchDB a Java CLI client was programmed. The client was further extended to apply different graph algorithms to the data. Only the bridges algorithm had to be implemented purely in Java, while the other algorithms could directly use the former implemented CouchDB views. First queries take a larger amount of time than later queries, since the first query result is cached. Especially for frequent changing graph data this trait is unfavourable. In summary, existing graph databases are best to process graph data, since graphs are a very particular data type.

2 Einleitung

Dieses Gruppensemesterprojekt hatte zum Ziel, Daten aus dem Dienst Reddit in einer CouchDB-Datenbank als Graph abzubilden und darauf verschiedene Abfragen zu ermöglichen. Diese können z. B. sein: Alle Freunde eines Freundes oder den Grad der Zentralität eines Knotens zu ermitteln. Die Umsetzung des Projekts erfolgte in drei Teilschritten.

1. Darstellung und Einarbeitung in das Datenbanksystem
2. Entwurf des Systems
3. Implementierung des Systems

Jeder Teilschritt wurde durch Präsentationen in den Übungsstunden den Kommiliton_innen präsentiert. Für alle Projektteile haben wir als Informationsquellen folgende Quellen verwendet welche somit auch in diese Dokumentation mit einfließen. Für einen schnellen Einstieg in CouchDB möchten wir vor allem den Blog von Houssem (2010a) empfehlen.

Brown (2012)
Edlich u.a. (2011)
Anderson (2015)
Foundation (2015)
Couchdb (2015)

DB-Engines (2015)
geeksforgeeks.org (2015)
Houssem (2010a)
Houssem (2010b)
Houssem (2010c)
Wikipedia (2015)

3 Kurzbeschreibung des verwendeten Systems

Das verwendete System zur Abbildung der Redditdaten als Graph mit Hilfe der Dokumentendatenbank CouchDB besteht ganz grundlegend aus drei Komponenten. Diese sind zum einen die **Redditdaten** und eine **Java Anwendung** zum Abrufen und Aufbereiten der Daten für die **CouchDB Datenbank**, siehe Abb. 1.

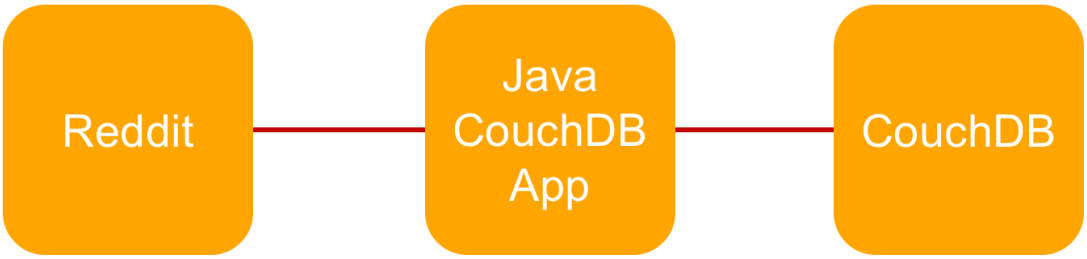

Abb. 1: Grundlegende Struktur des Systems

Es kann jeder Subreddit mittels Java Anwendung eingelesen und verarbeitet werden. Nach dem Einlesen werden in der Datenbank Dokumente für jeden Knoten und für jede Kante angelegt sowie z. B. die Anzahl an Kommentaren und die Freunde eines Knotens in jedem Knoten gespeichert. Sind Einlesen und Verarbeiten abgeschlossen, können auf dem Graphen folgende Anfragen mittels Command Line Interface (CLI) ausgeführt werden.

- friends: Ermitteln der direkten und indirekten Freunde
- degreeCentralityMinMax: Minimale und Maximale Zentralität des Graphen
- degreeCentrality: Zentralität eines Knotens
- bridges: Ermitteln der Brücken im Graph

Dabei wird von der Java-Anwendung auf verschiedene Views bzw. Ansichten der CouchDB-Datenbank zugegriffen, was die Geschwindigkeit der Anfragen deutlich erhöht. Berechnungen erfolgen in der Java-Anwendung.

4 Systemdarstellung

Daten in CouchDB werden als als JSON-Objekte in Dokumenten gespeichert. Dabei hat jedes Dokument eine eindeutige ID und eine Revision-ID. Um den Graphen abzubilden speichern wir jeden Knoten und alle dazugehörigen Kanten jeweils in einem Dokument. In den Knoten werden zusätzlich die Anzahl der Kommentare gespeichert und zusätzlich in einem Array die IDs der direkten Freunde eines Knotens, wie im folgenden dargestellt. In CouchDB können Designdokumente verwendet werden, um z. B. Ansichten zu definieren oder Beschränkungen bzw. Prüfungen für Datensätze umzusetzen.

Beispieldatensatz: Knoten

```
{ "_id": "person:Anna",
  "_rev": "1-7acd1b...",
  "comments": 1,
  "friends": ["person:Happy","person:Rehema","person:Michael"]
}
```

Beispieldatensatz: Kante

```
{ "_id": "friendship:person:Anna:with:person:Michael",
  "_rev": "1-ff652b587fcd0a79601d99751204bf07",
  "lastActive": 1446506810,  "title": "Schön dich als Freund_in zu
  ↪ haben!",
}
```

4.1 Beschränkungen im Datenmodell

Um Beschränkungen des logischen Datenmodells in CouchDB zu realisieren, kann in einem Designdokument eine Funktion mit dem Namen *validate_doc_update* definiert werden. Diese Funktion bekommt bis zu vier Parameter übergeben und wird vor jedem Speichern (Update) eines Dokumentes aufgerufen. Der erste Parameter ist das neue Dokument, dass gespeichert werden soll. Der zweite Parameter ist das aktuell gespeicherte Dokument, dieser Parameter ist leer, wenn noch kein Dokument gespeichert wurde. Als dritter Parameter wird ein Objekt mit Kontextinformationen und als vierter ein Objekt mit Sicherheitsinformationen übergeben.

Update Validation (validate_doc_update)

```
function (newDoc, oldDoc) {
        if (newDoc._id.match(/^friendship:/)) {
        var ids = newDoc._id.match(/person:([^:]*)/g);
        var one = ids[0].toLowerCase();
        var two = ids[1].toLowerCase();
                if(one > two) {
                        throw({forbidden: 'person1 must be
                        ↪ alphabetically lower than person2!'});
                }
        }
}
```

Es wurde definiert, dass der Schlüssel eines Kanten-Datensatzes dem Muster *friendship:person:<person1>:with:person:<person2>* (mit den Platzhaltern *<person1>* und *<person2>*) entspricht (z. B. *friendship:person:Anna:with:person:Michael*). Es ist wichtig, dass sichergestellt wird, dass ein Kanten-Datensatz nur einmal in der Datenbank existiert. Für die Schlüssel-Konstellation *friendship:person:<person2>:with:person:<person1>* darf kein weiteres Dokument angelegt werden.

Durch reguläre Ausdrücke wird sichergestellt, dass nur Dokumente, die einen Kanten-Datensatz darstellen betrachtet werden. Mit einem weiteren regulären Ausdruck werden die Werte hinter den Platzhaltern *<person1>* und *<person2>* selektiert. Diese Werte werden anhand ihrer Stelle im Alphabet verglichen. Sollte *<person1>* mit einem höher stelligen Buchstaben als *<person2>* beginnen, wird das Speichern mit einer Fehlermeldung abgebrochen.

4.2 Arbeiten mit Dokumenten

Die im Zusammenhang mit Datenbanksystemen bekannten Operationen zum Anlegen, Ausgeben, Bearbeiten und Löschen von Datensätzen (CRUD) werden in CouchDB auf die Operationen POST (**C**reate), GET (**R**ead), PUT (**U**pdate), **D**ELETE des HTTP-Protokolls abgebildet und als Schnittstelle für Nutzer angeboten. Im Folgenden werden Beispiel-Aufrufe zu diesen Operationen dargestellt.

Create: Datensatz erstellen

```
curl -X POST http://couchhost:5984/mydb -H 'Content-Type:
↪ application/json' -d '{"title":"Biking","body":"Mein Hobby ist
↪ Radfahren","date":"2009/01/30 18:04:11"}'
```

Rückgabe

```
{"ok":true,
 "id":"4464a2c95c4deeb4623cc95bc8000685",
 "rev":"1-35134234"}
```

Read: Datensatz lesen

```
curl -X GET
↪ http://couchhost:5984/mydb/4464a2c95c4deeb4623cc95bc8000685
```

Rückgabe

```
{"_id":"4464a2c95c4deeb4623cc95bc8000685",
 "_rev":"1-35134234",
 "title":"Biking",
 "body":"Mein Hobby ist Radfahren",
 "date":"2009/01/30 18:04:11"}
```

Update: Datensatz aktualisieren

```
curl -X PUT
↪ http://couchhost:5984/mydb/4464a2c95c4deeb4623cc95bc8000685 -H
↪ 'Content-Type: application/json' -d
↪ {"_rev":"1-35134234","title":"Biking","body":"Mein groesstes
↪ Hobby ist Radfahren.", "date":"2009/01/30 18:04:11"}
```

Rückgabe

```
{"ok":true,
 "id":"4464a2c95c4deeb4623cc95bc8000685",
 "rev":"2-6543522"}
```

Delete: Datensatz löschen

```
curl -X DELETE
↪ http://couchhost:5984/mydb/4464a2c95c4deeb4623cc95bc8000685
```

4.3 Views

Ansichten werden in CouchDB in einem Designdokument mit einem eindeutigen Namen als JSON-Objekt definiert. Jede Ansicht muss eine Map-Funktion definieren und kann optional eine Reduce-Funktion angeben.

Die Map-Funktion wird für jedes in der Datenbank vorhandene Dokument einmal aufgerufen und kann beliebig viele Schlüsselwertpaare auf Basis eines Dokumentes ausgeben.

Die Reduce-Funktion muss zwei verschiedene Aufruf-Szenarien unterscheiden. Im ersten reduce-Durchlauf wird die Reduce-Funktion für jedes ausgegebene Schlüsselwertpaar jeweils mit dem atomaren Wert aufgerufen. In folgenden Reduce-Durchläufen (rereduce) können für einen Schlüssel auch mehrere Wert Paare übergeben werden.

View: friends

friends: Map Die Map-Funktion der Ansicht *friends* stellt zunächst sicher, dass im Folgenden nur Kanten-Datensätze verwendet werden. Mit Hilfe eines regulären Ausdrucks werden die Schlüssel der beiden Knoten (Personen) selektiert. Die Map-Funktion gibt für jede Person (Schlüssel) die jeweils andere Person aus (Freundschaft).

```
1 function(doc) {
2   if (!doc._id.match(/^friendship:/)) return
3   var ids = doc._id.match(/person:([^:]*)/g)
4   var one = ids[0]
5   var two = ids[1]
6   emit(one, two);
7   emit(two, one);
8 }
```

friends: Reduce Die Reduce-Funktion prüft zunächst, ob es sich um einen rereduce-Aufruf handelt. Sollte dies nicht der Fall sein, wird der atomare Wert für die weitere Verarbeitung zurückgegeben. Das sorgt dafür, dass viele Schlüsselwertpaare zu einer Liste von Werten für einen Schlüssel aggregiert werden. Bei einem rereduce-Aufruf wird eine Menge von Werten zu einem Schlüssel übergeben. Mit Hilfe einer JavaScript-Map wird diese Menge auf Duplikate gefiltert. So kann sichergestellt werden, dass in den ausgegebenen Wertemengen keine Duplikate existieren. Nach dem Durchlaufen der Reduce-Funktion besteht die Ergebnismenge aus vielen Schlüsselwertpaaren, wobei die Wertemenge zu einem Schlüssel keine Duplikate mehr enthält.

```
function(key, values) {
  if(values.length) {
    var unique_friends = {};
    values.forEach(function(friend) {
      if(!unique_friends[friend]) {
        unique_friends[friend] = true;
      }    });
    return Object.keys(unique_friends);
  }
  return values;
}
```

View: friends2D

friends2D: Map Die Map-Funktion der Ansicht *friends2D* prüft, dass im Folgenden nur Knotendatensätze verwendet werden. Knotendatensätze enthalten eine Liste aller Knoten die über eine Kante verbunden sind (Freunde). Diese Liste wird in zwei verschachtelten Schleifen durchlaufen, um Beziehungen zweiten Grades zu finden. Knoten mit unterschiedlichen Schlüsseln sind über genau einen Knoten (das aktuelle Dokument) verbunden und werden für die weitere Verarbeitung als Schlüsselwertpaare ausgegeben. Zusätzlich wird in der höheren Schleife auch ein Schlüsselwertpaar zwischen aktuellem Knoten und je einem Nachbarn ausgegeben, um so in der Ergebnismenge alle Knoten bis zum zweiten Grad eines Knotens zu bestimmten (nicht nur exakt zweiter Grad).

```
function(doc) {
  if(!doc._id.match(/^person:/)) return
  if(doc.friends) {
    for(var f1 in doc.friends) {
      emit(doc._id, doc.friends[f1]);
      for(var f2 in doc.friends) {
        if(f1 !== f2) {
          emit(doc.friends[f2], doc.friends[f1]);
 } } } } }
```

friends2D: Reduce Auch an dieser Stelle wird die bereits von der *friends*-Ansicht bekannte Reduce-Funktion verwendet, um Duplikate aus den Ergebnis-Wertelisten zu entfernen.

```
function(key, values) {
  if(values.length) {
    var unique_friends = {};
    values.forEach(function(friend) {
      if(!unique_friends[friend]) {
        unique_friends[friend] = true;
      } });
    return Object.keys(unique_friends);
  } else {
    return values;
  } }
```

View: degree

degree: Map Die Ansicht *degree* verwendet eine Map-Funktion (ohne Reduce-Funktion), welche nur die Knotendatensäze für die weitere Verarbeitung verwendet. Ein Knoten kennt die Anzahl der an ihm anliegenden Kanten, diese wird als Wert in einem Schlüsselwertpaar ausgegeben.

```
function(doc) {
  if(!doc._id.match(/^person:/)) return
  if(doc.friends) {
    emit(doc._id, doc.friends.length);
  }
}
```

View: allEdges

allEdges: Map Die Ansicht *allEdges* wird verwendet, um die Gesamtzahl der Kanten zu bestimmen. Dafür werden die Kanten-Datensätze betrachtet und für jede Kante wird eine 1 zu dem festen Schlüssel „Kanten“ ausgegeben.

```
function(doc) {
  if (!doc._id.match(/^friendship:/)){
      emit("Kanten", 1);
  }
}
```

allEdges: Reduce Zum aufsummieren aller Schlüsselwertpaare wird die von CouchDB angebotene sum-Funktion genutzt, um die Gesamtzahl der Kanten zu bestimmen.

```
function(key, values) {
 return sum(values);
}
```

View: degreeMinMax

degreeMinMax: Map Mit Hilfe der Ansicht *degreeMinMax* werden alle Gradausprägungen der Knoten sowie die dazu gehörigen Knoten ausgegeben. Dafür werden im Folgenden die Knoten-Datensätze betrachtet. Als Wert wird "der Schlüssel des aktuellen Dokumentes" zu dem Schlüssel "Grad des aktuellen Knoten" ausgegeben.

```
function(doc) {
  if(!doc._id.match(/^person:/)) return
  if(doc.friends) {
    emit(doc.friends.length, doc._id)
  }
}
```

degreeMinMax: Reduce Auch an dieser Stelle wird die bereits von der *friends*-Ansicht bekannte Reduce-Funktion verwendet, um Duplikate aus den Ergebnis-Wertelisten zu entfernen. Mit Hilfe weiterer, von CouchDB angebotenen, Filter- und Sortiermöglichkeiten können alle Knoten zu einem bestimmten Grad ausgegeben werden (z. B. Minimal oder Maximal).

```
function(key, values) {
  if(values.length) {
    var unique_friends = {};
    values.forEach(function(friend) {
      if(!unique_friends[friend]) {
        unique_friends[friend] = true;
      }  });
    return Object.keys(unique_friends);
  } else {
    return values;
  } }
```

5 Darstellung der Implementierung

5.1 Interaktion mit dem System

Um das System möglichst einfach, aber dennoch flexibel genug zu halten, wurde für die Interaktion das Konzept eines Command Line Interfaces (CLI) gewählt. Das heißt, dass dem Programm beim Start verschiedene Startparameter übergeben werden können, welche anschließend zur Ausführung verschiedener Funktionen führen. Dabei hilft das Apache CLI Framework. Falls das Programm ohne Startparameter gestartet wurde, weißt die Nachricht „Enter Command:“ darauf hin, dass hier die Befehle auch noch manuell ausgeführt werden können. Die verfügbaren Operationen werden im Programm in folgendem Stil eingefügt:

```
1 addDetailedOption(options, "fetch", "nameOfSubReddit or
  ↪ urlToSubReddit", true, "fetches Subreddit to CouchDb");
```

Dabei entspricht *fetch* dem angefügten Befehl, nameOfSubReddit or urlToSubReddit ist eine Beschreibung des Parameters, true besagt dass ein Parameter übergeben werden muss und der letzte Parameter der Funktion ist eine Kurzbeschreibung für den Nutzer. In diesem System wurden die Befehle *fetch*, use, *friends*, *degreeCentrality*, *process*, *degreeCentralityMinMax*, *bridges* und *help* implementiert.

Das letztendliche Verarbeiten der übergebenen Befehle erfordert keine komplizierte Implementierung mehr. Es werden lediglich verschiedene if-Abfragen, welche Überprüfen, welche Befehle übergeben wurden benötigt. Diese rufen dann die entsprechenden Funktionalitäten in der CouchApp auf und achten auf eine gewisse Reihenfolge der Anweisungen. So muss beispielsweise für ein neues Subreddit immer zuerst *fetch* aufgerufen werden. Dies legt die Datenbank an, fügt das Designdokument hinzu und lädt alle Kanten aus dem Subreddit. Ist dieser Schritt bereits erledigt, kann alternativ auch mit dem Befehl *use* ein Subreddit ausgewählt werden. Bevor nun Graphenoperationen ausgeführt werden können, muss noch *process* ausgeführt werden. Dabei werden aus allen Kanten die entsprechenden Knoten (Reddit-Nutzer) ausgelesen und ebenfalls in die Datenbank gespeichert. Anschließend stehen auch die anderen Befehle zur Verfügung.

5.2 Laden der Reddit-Daten

Zunächst wird, falls für dieses Subreddit nicht schon vorhanden, eine gleichnamige Datenbank in Couchdb angelegt. Dieser wird über ein HTTP PUT das vorgestellte JSON Designdokument hinzugefügt, so dass bereits alle Views in der Datenbank vorhanden

sind. Anschließend werden die Daten mittels einer öffentlichen Reddit-REST-Schnittstelle eingelesen. Dabei werden JSON-Objekte zurückgeliefert. Wir können so z. B. den Subreddit „buecher“ über folgende URL in mehreren einzelnen Teilen `https://www.reddit.com/r/buecher/.json?limit=100&after=` einlesen. Die Anzahl der Datensätze ist durch das Attribut *limit* auf 100 beschränkt. Über das Attribut *after* erhalten wir Zugriff auf die weiteren Datensätze. Die Anzahl der so aus einem Subreddit geladenen Datensätze kann über das Ändern der Variable *MAX_ROUNDS* begrenzt werden.

Für das Weiterverarbeiten der geladenen Daten wird, um einen deutlichen Performancegewinn zu erreichen, jeder Reddit-Thread in einem separaten Java-Thread verarbeitet. Dies wird durch die mit Java 8 eingeführte Streambibliothek deutlich vereinfacht. Diese ist für die meisten in Java verwendeten Collections verfügbar, so auch für die zum Sammeln der Threads verwendete Datenstruktur Queue. Der folgende Einzeiler führt zum parallelen Verarbeiten der Reddit-Threads

```
209 idQueue.parallelStream().forEach(curId -> {
210         processComments(repo, curId);
```

Die Funktion *processComments* liest im wesentlich den als Pfad übergebenen Reddit-Thread als JSON-Array ein und liest für das System wichtige Informationen aus.
Die für uns an dieser Stelle wichtigen Informationen stellen die Kanten in dem Graphen dar, welcher in der Datenbank abgebildet werden soll. Dabei wird der Prozedur zum Speichern unter anderem immer der letzte Autor übergeben. Pro Kommentar werden dann die Attribute *id, lastActive* sowie *title* in der Datenstruktur *Edge* gesetzt. Die Id setzt sich dabei wie bereits erläutert aus einer alphabetisch sortierten Kombination der beiden Authoren zusammen. Das Attribut *lastActive* beschreibt den Zeitpunkt des letzten Kommentars, denn nur die neuste Interaktion soll in der Datenbank gespeichert werden. Übrig bleibt das Attribut *title*. Da sich herausstellte, dass Kommentare keinen Titel haben, werden an dieser Stelle die ersten 10 Zeichen des Kommentars selbst gespeichert, um Speicherplatz zu sparen. Daraufhin wird für diesen Nutzer die Kommentar Anzahl um eins erhöht. Die Anzahl der Kommentare pro Nutzer werden in einer *HashMap* gespeichert, welche die Anzahl der Kommentare pro Nutzer enthält, sofern mindestens einer vorhanden ist. Das erstellen eines Threads zählt in diesem Fall nicht als Kommentar. Anschließend wird die so angelegte *Edge* in der Datenbank gespeichert und überprüft, ob dieser Kommentar eigene Kommentare hat. In diesem Fall erfolgt ein rekursives Speichern der kommentierten Kommentare, welche sich unter dem JSON-Datensatz *replies* verbergen.

5.3 Speichern von Java-Daten in CouchDB

Um die im Javaprogramm angelegten Daten in CouchDB etwas bequemer abspeichern zu können, wurde in diesem Projekt das Framework Ektorp verwendet. Ektorp bildet die Schnittstelle zwischen Javaprogramm und CouchDB und lässt Nutzern mit relativ wenig Aufwand mit der Datenbank interagieren. Um nun konkret die bereits erwähnte Datenstruktur *Edge* abzuspeichern muss diese zwei Eigenschaften erfüllen. Zum ersten muss sie von der Ektorpklasse *CouchDBDocument* erben. Weitere Anforderungen bestehen an das Dokument zwar nicht, doch um damit arbeiten zu können wird noch ein Repository je Dokument benötigt. Ein solches Repository wird in folgendem Code abgebildet

```
6  public class EdgeRepository extends CouchDbRepositorySupport<Edge>{
7
8          public EdgeRepository(Class<Edge> type, CouchDbConnector db)
           ↪  {
9                  super(type, db);
10         }
11
12 }
```

Nun können Dokumente der Struktur *Edge* ohne Probleme gespeichert werden. Ein einfaches

```
337 repo.add(e);
```

genügt dazu bereits. Wobei wir voraussetzen, dass es vom Datentyp *Edge* ist und repo ein *Edgereposity* mit Verbindung zur CouchDB. Im Fall dieses Systems wird vorher jedoch noch überprüft, ob es eine solche Kante bereits gibt und wann die letzte Aktivität war, falls sie bereits vorhanden war. Wenn eine neuere Kante in der Datenbank vorhanden ist, passiert nichts. Anderenfalls wird ein Update oder ein Speichern ausgeführt über die Funktinoen textitadd/update des Repository.

5.4 Das Verarbeiten der Kanten

Eine Besonderheit in dem hier entwickelten System stellt der Befehl *process* dar. Eigentlich werden, wie bereits in vorherigen Kapiteln ausgeführt, nur die Kanten aus dem Subreddit geladen. Für verschiedene Map- und Reduce-Funktionen sind Knoten in Form von Reddit-Nutzern unverzichtbar, da wir in diesen die Freunde der Nutzer speichern. Für diesen Schritt muss der in der Datenbank abgelegte View *friends* ausgeführt werden. Auch dies geschieht über das Ektorp Framework. Wir erhalten eine Tabelle mit Nutzern und deren Freunden, also Reddit-Nutzer, welche sie kommentiert haben. Da es sich auch

hier um riesige Datenmengen handelt, wird erneut mit der Java 8 Stream API parallel jede Zeile verarbeitet. Das Verarbeiten beinhaltet dabei das Erstellen eines *Nodes*, das Setzten von Ids und Kommentaren sowie dem Umwandeln des JSON-Arrays mit den Freunden in eine Java freundlichere Liste, welche ebenfalls im *Node* gesetzt wird. Das anschließende Speichern in der Datenbank entspricht prinzipiell dem bereits im Detail vorgestellten Speichern einer Kante.

5.5 Funktionen der CouchApp und Abrufen der Views

Mit Ausnahme der Brücken, konnten alle geforderten Graphenoperationen bereits über Map- und Reduce-Funktionen in der CouchDB umgesetzt werden. Infolge dessen bestehen diese Funktionen in der CouchApp nur darin Views abzurufen, möglicherweise zu filtern und in einem lesbaren Format auszugeben. Als Beispiel soll dazu die Zentralität eines bestimmten Nutzers dienen:

```
171 public void degreeCentrality(String keyToUser) {
172         keyToUser = rebuildUserKey(keyToUser);
173 
174         ViewQuery degreeQuery = new
            ↪  ViewQuery().designDocId("_design/graphQueries"
            ↪  ).viewName("degree").key(keyToUser);
175 
176         ViewQuery edgesQuery = new
            ↪  ViewQuery().designDocId("_design/graphQueries"
            ↪  ).viewName("allEdges");
177 
178         int degree =
            ↪  dbfetcher.getDb().queryView(degreeQuery).getRows(
            ↪  ).get(0).getValueAsInt();
179 
180         int edges = dbfetcher.getDb().queryView(edgesQuery).getRows(
            ↪  ).get(0).getValueAsInt();
181 
182         System.out.println((double) degree / edges);
183 }
```

Wie bereits im Zusammenhang mit dem View *friends* erwähnt, werden Views über Ektorp abgerufen. Dazu dient die Klasse *ViewQuery* in der wie oben abgebildet alle wichtigen Informationen javatypisch angewendet werden können. Um ein Resultat zu erhalten, muss

dieser Query in der Datenbank ausgeführt werden. Anschließend werden die erwarteten Werte extrahiert und auf der Konsole ausgeben.

5.6 Berechnen der Brücken

Für die Berechnung der Brücken werden alle Kanten benötigt. Diese werden durch eine Abfrage der edges View aus CouchDB erhalten. Anschließend werden die Strings aufgetrennt, sodass Knotenpaare vorliegen. Hierbei werden Ecken bidirektional betrachtet und Doppelungen aussortiert. Mit der Anzahl der Knoten wird ein BridgeFinder initialisiert. Mit der Methode addEdgeString werden Knoten, die eine Kante bilden, als Strings übergeben.
Der BridgeFinder nummeriert alle Kanten und bildet diese Nummer mittels einer Hashmap auf die jeweilige Kante ab. Für jede Kante werden folgende Schritte ausgeführt:

- Entferne die Kante aus dem Graph
- Untersuche, ob der Graph noch verbunden ist
- Füge die Kante wieder zum Graph hinzu

(geeksforgeeks.org, 2015)

6 Laufzeitevaluation

Bei unserem System muss im wesentlichen immer zwischen einer Erstabfrage und einer Folgeabfrage unterschieden werden. Dabei ist die erstmalige Abfrage auf einem Datensatz um einen wesentlichen Zeitfaktor länger als Folgeabfragen. Wie unsere Laufzeitmessungen in Tabelle 1 beispielhaft zeigt.
Die Geschwindigkeit unseres System wird primär durch vier verschiedene Laufzeitkomponenten bestimmt. Die Reihenfolge stellt den Einfluss auf die Laufzeit dar *1. (starke Laufzeitverlängerung)* bis *4. (geringe Laufzeitverlängerung)* diese sind:

1. ***Erzeugen der CouchDB Ansichten***: Indexierung der Daten durch Datenbank
2. *process*: Vorbereiten der Daten für Abfragen
3. *fetch*: Abrufen der Daten aus Reddit
4. *Berechnungen* der Java-Anwendung auf Grundlage der CouchDB Views

Wir haben unser System mit dem kleinen Subreddit **mobileweb** und dem ca. 32 Fach größeren (bezogen auf die Anzahl der Dokumente) Subreddit **nfl** getestet und gemessen. Unsere Testszenarien haben wir wie folgt aufgebaut.

Testszenario *friends*

1. *fetch* ausführen und messen
2. *process* ausführen und messen
3. direkt den Befehl *friends* mit einer Beliebigen Person aufrufen und messen
4. *friends* und *bridges* aufrufen und messen

Testszenario *bridges*

1. *fetch* ausführen und messen
2. *process* ausführen und messen
3. direkt den Befehl *bridges* ausführen
4. *friends* und *bridges* aufrufen und messen

Die Tabelle 1 stellt die durchgeführten Testszenarien zweier Subreddits und die erhobenen Zeiten gegenüber. Dabei wird deutlich, dass vor allem während des *process* Schritts und bei jeglicher Erstabfrage deutlicher Rechenbedarf besteht. Dabei macht es kaum einen Unterschied, ob die Erstabfrage *friends* oder *bridges* ist. Der hier gemessene Unterschied kann in unserem Messumfeld auch auf Messungenauigkeiten und andere Einflussfaktoren zurückgeführt werden und müsste für eine eingehende Betrachtung mit mehreren Messreihen untersucht werden.

Beschreibung	**mobileweb**	**nfl**
Datenbankgröße	4,1 MB	182,8 MB
Dokumente (Knoten und Kanten) Insgesamt	1446	46203
-fetch	1,3 Min.	2,75 Min.
-process	ca. 10 Sek.	13,75 Min.
-friends als Erstabfrage	ca. 3 Sek.	21,5 Min.
-bridges als Erstabfrage	ca. 5 Sek.	24 Min.
-friends als Folgeabfrage	Sofort	Sofort
-bridges als Folgeabfrage	Sofort	ca. 10 Sek.

Tab. 1: Gegenüberstellung der Subreddits von mobileweb und nfl

Hintergrund hierfür ist vor allem, dass während des Ausführen von *process* die Datenbank neu indexiert wird, weil die Ansicht *friends* benötigt wird. Bei der Erstabfrage müssen sodann von der Datenbank die Ansichten *friends* und *friends2D* erstellt werden, was ebenfalls einen erheblichen Aufwand bedeutet. Hierbei wird deutlich, dass CouchDB bei großen, sich oft ändernden Graphen nicht unbedingt die beste Lösung ist. Zudem zeigt sich, dass die Erzeugung und Zwischenspeicherung der Views eine erhebliche Performance Steigerung bedeutet.

7 Zusammenfassung

Grundlegend kann festgehalten werden, dass CouchDB nicht für die Verwendung mit Graphen konzipiert ist. Das Hauptproblem ist die Erstellung der Ansichten, welche sehr viel Zeit in Anspruch nimmt.
Ein interessanter Ansatz wäre zu prüfen, ob bei einem Verzicht auf die Verwendung der Views und einer kompletten Umsetzung in einer externen Anwendung ein Performancegewinn erreicht werden könnte. Insgesamt hat uns das Arbeiten mit CouchDB sehr viel Spaß bereitet. Der Einstieg in das Datenbanksystem war schnell und zielstrebig möglich. Etwas Übung und Ausdauer erfordert die Implementierung der Map- und Reduce-Funktionen. Abschließend bleibt uns nur zu sagen, dass wir während des Projekts einiges zu CouchDB gelernt haben und eine Menge Spaß dabei hatten.

Literatur

Anderson, J. C. (2015): **Daten finden mit Views**.
http://guide.couchdb.org/editions/1/de/views.html, Letzter Zugriff am: 2016-01-06.

Brown, M. C. (2012): Getting Started with CouchDB 1st edition by Brown, MC (2012) Taschenbuch. O'Reilly Media.

Couchdb, W. (2015): **Built-In_Reduce_Functions - Couchdb Wiki**.
http://wiki.apache.org/couchdb/Built-In_Reduce_Functions, Letzter Zugriff am: 2016-01-06.

DB-Engines (2015): **CouchDB vs. Redis Vergleich**.
http://db-engines.com/de/system/CouchDB%3BRedis, Letzter Zugriff am: 2016-01-06.

Edlich, S.; Friedland, A.; Hampe, J.; Brauer, B. und Brückner, M. (2011): NoSQL: Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken. Carl Hanser Verlag GmbH & Co. KG, 2., aktualisierte und erweiterte auflage Aufl.

Foundation, A. S. (2015): **Overview — Apache CouchDB 1.6 Documentation**.
http://docs.couchdb.org/en/1.6.1/, Letzter Zugriff am: 2016-01-06.

geeksforgeeks.org (2015): **Bridges in a graph**.
http://www.geeksforgeeks.org/bridge-in-a-graph/, Letzter Zugriff am: 2016-01-06.

Houssem (2010a): **CouchDb, Creating views**.
http://www.hbensalem.com/nosql/couchdb-creating-views/, Letzter Zugriff am: 2016-01-06.

Houssem (2010b): **CouchDB, Design Documents**.
http://www.hbensalem.com/nosql/couchdb-design-documents/, Letzter Zugriff am: 2016-01-06.

Houssem (2010c): **CouchDB, Validation functions**.
http://www.hbensalem.com/nosql/couchdb-validation-functions/, Letzter Zugriff am: 2016-01-06.

Wikipedia (2015): **CouchDB**.
https://de.wikipedia.org/w/index.php?title=CouchDB&oldid=148323130, Letzter Zugriff am: 2016-01-06.

Tabellenverzeichnis

Abbildungsverzeichnis