Dokumentenbasierte Datenbanken mit CouchDB*

Marcel Fries
Universität Siegen
Weidenauer Strasse 263a
Siegen, Deutschland
marcel.fries@student.uni-siegen.de

ABSTRACT

CouchDB gehört zu den sogenannten NoSQL Datenbanksystemen, ich werde auf die Unterschiede zu konventionellen SQL Datenbanken, bzw. anderen dokumentenbasierten Datenbanken eingehen.

CouchDB benutzt Dokumente im JSON Format zur Speicherung von Daten, sowie Javascript um MapReduce Verfahren durchzuführen. Der Zugriff auf eine CouchDB Datenbank erfolgt über eine bereitgestellte HTTP API

Die Kombination von Javascript, HTTP und JSON Dokumenten macht CouchDB zu einem Idealen Datenbanksystem für Anwendungen im Web, da es sich hier um weit verbreitete und von allen gängigen Webbrowsern unterstützte Technologien handelt.

Ich werde sowohl die zugrunde liegende Architektur von CouchDB beschreiben, als auch Funktionsweise und Konzept des MapReduce Verfahrens, dass bei CouchDB Anwendung findet erklären.

Keywords

CouchDB, NoSQL, Database, MapReduce, Javascript, JSON, DocumentBased

1. EINFÜHRUNG

Im Laufe der letzten Jahre ist sind die Anforderungen an Datenbanksysteme stetig gestiegen, enorme Datenmengen, gestiegene Anforderungen an Performance, viele parallele Zugriffe und das Aufbauen verteilter Datenbanken. Auch die Anforderungen an die Form, in welcher Daten gespeichert werden können, hat sich geändert. In klassischen relationalen Datenbanken werden alle Daten in Tabellen abgelegt und müssen strengen Maßgaben bezüglich ihrer Form genügen. *Big Data* und die Nutzung von Datenbanken im Internet haben Raum für neue Arten von Datenbanksystemen geschaffen.
In dieser Arbeit werde ich auf CouchDB, einen Vertreter der sogenannten dokumentenbasierten Datenbanksystemen eingehen. Dabei wird zunächst ein Überblick über die Architektur von CouchDB Datenbanken und deren Vernetzung gegeben. Anschließend werden die Stärken und Schwächen der Technologie mit einem anderen Vertreter der dokumentenbasierten Datenbanken verglichen und Schlüsse über die Verschiedenen Anwendungsszenarien für CouchDB Datenbanken gezogen.

*Offizielle CouchDB Homepage: `https://couchdb.apache.org/`

2. GRUNDLAGEN

CouchDB greift auf bestehende Technologien und Verfahren zurück. In diesem Kapitel gehe ich auf die Grundlagen von *B-Bäumen* und dem *MapReduce* Verfahren ein. Die konkrete Implementation in CouchDB und spezifische Änderungen werden in Kapitel 3 erläutert .

2.1 B-Bäume

Das Durchsuchen von Datenbanken, besonderes wenn sie viele Dokumente enthalten ist ineffizient, wenn jedes einzelne Dokument durchsucht werden muss. Aus diesem Grund werden Indizes angelegt, die Referenzen auf die eigentlichen Dokumente enthalten.
In diesem Fall betrachten wir den Aufbau von *B+-Bäumen*. Diese speichern Referenzen auf Dokumente nur innerhalb der Blätter des Baumes. Die Knoten des *B+-Baums* enthalten lediglich Referenzen auf tiefer liegende Knoten. Um sicher zu stellen, dass die Blätter des Baums immer auf der gleichen Ebene liegen, wird der Baum *ausbalanciert*. Knoten innerhalb des Baumes haben eine maximale Größe. Um eine möglichst kleine Baumhöhe zu erzielen, muss ein Knoten immer mindestens zur Hälfte gefüllt sein. Das letzte Element in einem Blatt enthält immer eine Referenz auf das nächste Blatt. Dadurch entsteht eine sortierte Liste aller Elemente des Baums, was die Abfrage aller Elemente bzw. die Abfrage eines Bereichs von Elementen sehr effizient macht.[2]

2.2 MapReduce

Das MapReduce Verfahren ist ein von Google eingeführtes Programmiermodell[3]. Das Framework dient der parallelisierten Verarbeitung großer Datenmengen, auch in verteilten Rechennetzen. Das Verfahren besteht dabei aus zwei für den Nutzer relevanten Funktionen, der *map* und der *reduce*. Die Map-Funktion wird genutzt um eine Liste aus Zwischenergebnissen zu erstellen, bzw. direkt Funktionen wie Auflistungen aller Elemente oder Suchen nach mehreren Elementen zu

realisieren. Die Reduce-Funktion nimmt das Ergebnis einer Map-Funktion als Eingabe. Ihre Aufgabe ist es das Ergebnis auf einen einzelnen Wert zu reduzieren. Häufige Anwendung für die Reduce-Funktion ist das Summieren aller Ergebnisse der Map-Funktion.
Der Vorteil beim Einsatz von MapReduce ist die Fähigkeit, Berechnungen parallel über verschiedene Rechner zu verteilen, was die Geschwindigkeit von Abfragen deutlich erhöht.

3. EIGENSCHAFTEN VON COUCHDB

In diesem Kapitel wird tiefer auf die CouchDB spezifischen Eigenschaften eingegangen. Neben der konkreten Implementation von MapReduce und der Indizierung mit Hilfe der B+-Bäume wird ebenfalls auf die Architektur von Dokumenten innerhalb von CouchDB und der Zugriff auf diese behandelt.

3.1 Dokumente

Anders als bei den relationalen Datenbanken, in denen Daten in Tabellen abgelegt werden, speichert CouchDB Daten in JSON Dokumenten ab.

```
1  {
2    "_id" : "cc3174e944a3931548d3b5281d00046d",
3    "_rev" : "1-10ff3d7b7bdab3e0a7b660775a05a356"
         ,
4    "title" : "Smoke on the Water",
5    "artist" : "Deep Purple",
6    "genre" : {
7      "id": "Heavy Metal"
8      "id": "Hard Rock"
9    }
10 }
```

Listing 1: JSON Dokument

Wichtig dabei ist, dass die Felder `"_id"` und `"_rev"` die einzigen Felder sind, die auf jeden Fall benötigt werden. Alle anderen Einträge sind optional und können von Dokument zu Dokument in der Datenbank unterschiedlich sein. Das `"_rev"` Feld zeigt dabei eine md5 gehashte Version des Dokuments mit einem Präfix für die Anzahl an Updates an dem Dokument an. Es Wird jedes mal beim Update eines Dokuments aktualisiert. Es wird Außerdem zur Kontrolle von Schreibzugriffen auf verteilten Datenbanken genutzt.
Das `"_id"` Feld enthält standardmäßig eine automatisch generierte UUID, sie definiert die URL unter der das Dokument in der Datenbank gefunden werden kann. Es ist möglich das `"_id"` Feld nach belieben auf den Anwendungsfall anzupassen. Eine beliebige Änderung der ID ist auf die gleiche Weise möglich, wie bei jedem anderen Attribut eines Dokuments. Auch beim Anlegen neuer Dokumente kann die `"_id"` frei gewählt werden. Es muss jedoch darauf geachtet werden, dass die `"_id"` immer einzigartig sein muss. Wird versucht ein Dokument mit einer bereits existierenden `"_id"` anzulegen wird dies von der in CouchDB eingebauten Kollisionserkennung verhindert.

3.2 Relation zwischen Dokumenten

Eine weitere wichtige Eigenschaft, nicht nur von CouchDB, sondern von dokumentenbasierten Datenbanksystemen im Allgemeinen ist die Abgeschlossenheit von Dokumenten. Dokumente können immer für sich allein genommen existieren, ohne von Verweisen auf andere Dokumente abhängig zu sein. Im Gegensatz zu relationalen Datenbanken existiert in CouchDB kein eingebauter Mechanismus um Verknüpfungen zwischen Dokumenten herzustellen.
Sollten jedoch Relationen zwischen Dokumenten gebraucht werden, können diese frei von der Anwendung bestimmt werden. Das Beispiel aus [3.1] könnte um ein zusätzliches Feld `Album` erweitert werden, welches die ID eines anderen Dokuments enthält. In Kombination mit deinem Dokument, welches das *Album* beschreibt und Felder `"Title1": "Verweis"` enthält, kann so eine Relation zwischen den Dokumenten modelliert werden. Dabei ist aber zu beachten, dass die Interpretation der Relation vollkommen von der Anwendungslogik abhängt und nach belieben definiert werden kann.

3.3 Indizierung

CouchDB nutzt zur Indizierung von Dokumenten eine leicht abgewandelte Implementierung der gewöhnlichen[2] B-Baum Definition. Damit kann sichergestellt werden, dass Zugriffe auf eine Datenbank immer in kürzester Zeit beantwortet werden können. Selbst bei jenseits von Millionen Dokumenten in einer Datenbank bleibt die Höhe eines B-Baums noch im Einstelligen Bereich. Die Höhe des Baums hat enorme Auswirkungen auf die Zeit, die zum traversieren des gesamten Baums benötigt wird.
Bemerkenswert ist, dass B-Bäume in CouchDB nur *append only* implementiert sind. Das bedeutet, dass Änderungen an Dokumenten als auch Löschungen immer am Ende der Datei abgehangen werden. Dadurch ist gegeben, dass jederzeit ein Snapshot einer früheren Version der Datenbank vorhanden ist.
Des weiteren sorgt die *Multi-Version Concurrency Control* dafür, dass die Datenbank nicht für Lesezugriffe gesperrt ist, während ein Schreibvorgang läuft.

Es wird jeweils ein B-Baum pro Datenbank angelegt. Zusätzlich dazu wird auch jede View in einem eigenen B-Baum gespeichert. Das hat den Vorteil, dass beim erneuten Aufruf der View die Map bzw. Reduce Funktion nicht noch einmal ausgeführt werden muss. Dadurch wird die Geschwindigkeit von Lesezugriffen erheblich gesteigert. Kommt es zu Änderungen an den Dokumenten, wird es nötig die B-Bäume der entsprechenden Map Funktionen erneut zu erzeugen. Dies geschieht automatisch beim nächsten Aufruf der entsprechenden Suchanfrage.

3.4 MapReduce in CouchDB

Zum Durchführen von Suchen in der Datenbank wird das bereits in [2.2] erklärte MapReduce Verfahren genutzt. CouchDB Map Funktionen werden in Javascript geschrieben. Die Grundlegende Funktionsweise dabei ist, dass die Map Funktion automatisch mit jedem Dokument der Datenbank als Eingabe einmal ausgeführt wird. Der Javascript Code innerhalb der Map Funktion wählt die entsprechenden Attribute des Dokuments aus und und nutzt die `emit(key, value)` Funktion um diese auszugeben.

Da Dokumente keine feste Struktur enthalten, muss in jeder Map Funktion zunächst überprüft werden , ob die gesuchten Attribute in dem Dokument existieren. Die Map Funktion gibt immer exakt 2 Werte zurück `key` und `value`. Es ist vom

Anwendungsfall abhängig wie die Map Funktion diese Werte mit Attributen aus den Dokumenten belegt. Dabei sind auch Listen, oder andere JSON Konstrukte als Rückgabewerte zugelassen.

```
function(doc) {
  var store, price, key;
  if (doc.item && doc.prices) {
    for (store in doc.prices) {
      price = doc.prices[store];
      key = [doc.item, price];
      emit(key, store);
    }
  }
}
```

Listing 2: Beipiel für eine Map Funktion[4]

Eine Reduce Funktion wird benutzt um die Ausgabe der Map Funktion zu spezifizieren, bzw. zu verkleinern. Die Funktion erwartet 3 Argumente `key`, `value` und `rereduce`. Eine Reduce Funktion gibt immer nur einen Wert zurück geben. Häufige Anwendungsfälle sind z.B. das Zählen aller Ausgaben der Map Funktion bzw. das Aufsummieren dieser. Zu beachten dabei ist der `rereduce` Ausdruck der Funktion. Er kann genutzt werden um die Reduce Funktion mehrfach auszuführen.

3.5 HTTP API

Zugriff auf eine CouchDB Datenbank erfolgt über die eingebaute HTTP API. Dabei können Dokumente, Map Funktionen, Views usw. direkt über die entsprechende HTTP URL abgerufen werden.

```
curl -X GET http://adresse_der_datenbank/
    id_des_dokuments
{
  "_id" : "id_des_dokuments"
  "_rev" : "revision_des_dokuments"
  "attr" : "ein_attribut"
}
```

Listing 3: Einfacher HTTP GET Request auf ein Dokument der Datenbank

Auf die gleiche Weise lassen sich auch Dokumente oder Datenbanken erstellen:

```
curl -X PUT http://server/neuDatenbank
{"ok" : true}
```

CouchDB gibt zu jedem PUT Request eine Antwort im JSON Format mit einer Bestätigung, bzw. Fehlermeldung zurück:

```
curl -X PUT http://server/neuDatenbank
{"error":"file_exists", "reason":"The database
    could not be created, the file already
    exists."}
```

3.6 Design Dokumente

Um dem Nutzer der Datenbank Abfragen auf die Datenbank zu ermöglichen enthält jede CouchDB Datenbank ein *Design Dokument*. Es enthält, neben einigen Informationen über die Datenbank selbst (URL der Datenbank, Revision, etc), die Map und Reduce Funktionen die dem Nutzer zur Verfügung gestellt werden sollen. Sie werden *Views* gennant.
Die ID eines Design Dokuments hat immer das Namensschema:
`_design/name_der_anwendung`

```
{
  "_id": "_design/studenten",
  "_rev": "1-a8487fd2a8b3698a19a5ae6dd59b178d",
  "language": "javascript",
  "views": {
    "matrikel": {
      "map": "function(doc) {
        if(doc.matrikelNr){
          emit(doc._id, doc.martrikelNr);
        }
      }"
    }
  }
}
```

Listing 4: Simples Beispiel eines Design Dokuments

Die Map Funktion kann in diesem Beispiel direkt über die HTTP API aufgerufen werden:

```
curl -X GET http://server/datenbank_name/
    _design/studenten/_view/matrikel
{
  "id1" : "12375789",
  "id2" : "462311",
  // usw.
}
```

Listing 5: Abfrage der 'matrikel' View.

4. VERNETZUNG VON DATENBANKEN

Mit zunehmender Grösse der Datenbank und Menge an Zugriffen auf diese sinkt die Effizienz eines einzelnen Servers, weshalb es bald notwendig wird, die Datenbank über mehrere Server zu verteilen. Ein Netz von zusammenhängenden Datenbanken eignet sich zum Verteilen der Zugriffe und zur schnelleren Bearbeitung dieser.

Die Vernetzung von Datenbanken findet bei CouchDB über die Replikationsfunktion statt. Über einen simplen HTTP POST Request kann die Replikation einer Datenbank auf einen anderen Server, oder als lokale Kopie eingeleitet werden:

```
curl -X POST /_replicate HTTP/1.1
{"source":"datenbank","target":"http://server.
    tld/datenbank"}
```

CouchDB Datenbanken besitzen eine Sequenznummer, welche bei jeder Änderung inkrementiert wird. Alle Änderungen werden dabei aufgezeichnet(Löschungen, Updates, usw.) sodass nur die geänderten Teile übertragen werden müssen. Um auch künftige Änderungen automatisch zu übertragen kann `"continuous": true` Trigger gesetzt werden. Dieser bewirkt, dass nach dem Replizieren der Datenbank weiterhin automatisch Änderungen an der Datenbank übertragen werden. Auf diese Weise kann ein Netzwerk von Datenbanken erstellt werden, die sich mit Hilfe der Replikationsfunktion untereinander austauschen.

4.1 Eventual Consistency

Bei der Verteilung einer Datenbank über mehrere Server kommt es auf 3 wichtige Parameter an die im sogenannten CAP-Theorem[5] festgehalten sind:

1. Konsistenz: Alle Datenbanken haben den gleichen Datenbestand
2. Verfügbarkeit: Jeder Client muss Zugriff auf die Daten haben
3. Partitionstoleranz: Das System kann selbst dann weiterarbeiten, wenn einzelne Knoten ausfallen

Laut CAP-Theorem[5] ist es jedoch nur möglich 2 dieser Aspekte gleichzeitig zu erfüllen (siehe Figure 1). CouchDB realisiert durch seine Replikation sowohl die Verfügbarkeit als auch die Partitionstoleranz des verteilten System. Da jeder Knoten des Netzes eine exakte Kopie der Datenbank enthält ist eine hohe Verfügbarkeit und schneller Zugriff auf die Daten gesichert. Weiterhin ermöglicht die sogenannte *Multi-Version Concurrency Control* Lesezugriffe auf eine Datenbank noch während ein Schreibvorgang läuft. In diesem Fall wird eine *alte* Version des Dokuments, ohne die Änderungen, herausgegeben, bis die Änderungen am Dokument abgeschlossen sind. Auf diese Weise kann die Anzahl an Abfragen auf eine Datenbank innerhalb eines Zeitfensters weiterhin erhöht werden.

Die hohe Zugriffsgeschwindigkeit und Verfügbarkeit hat allerdings den Nachteil, dass die Daten über das Netzwerk nicht ständig Konsistenz gehalten werden können. Da sich CouchDB Datenbanken in regelmässigen Abständen synchronisieren, kann es hier zu Kollisionen kommen wenn ein Dokument an zwei Knoten zur gleichen Zeit bearbeitet werden. Verteilen sich diese geänderten Dokumente nun über das Netzwerk erkennt CouchDB die Kollision automatisch und leitet eine automatische Lösung des Problems ein:

1. Die *Gewinner* Version wird als die neue aktuelle Version gespeichert.
2. Die *Verlierer* Version wird in der Historie des Dokuments gespeichert und kann jederzeit eingesehen oder wiederhergestellt werden.

Dieses Verfahren ist konsistent in dem gesamten System und hat immer das gleiche Ergebnis.
Genauere Fehlerbehandlung(z.B. das Vereinigen von kollidierenden Dokumenten) kann in der Anwendung manuell vorgenommen werden und muss auf den spezifischen Anwendungsfall angepasst werden.

5. VERGLEICH MIT MONGODB

MongoDB[1] eignet sich besonders für einen Vergleich, da es sich dabei ebenfalls ein dokumentenbasiertes System handelt.
MongoDB nutzt im Vergleich zu CouchDB Dokumente im BSON Format, wobei es sich um eine Binarcodierte Variante von JSON handelt. MongoDB fügt zusätzliche Datentypen zu seinen Dokumenten hinzu. CouchDB kennt hingegen keine Datentypen in seinen Dokumenten.
Beim Querying nutzt CouchDB reine MapReduce Funktionen, welche in Javascript geschrieben werden während MongoDB sich eher an der SQL-Syntax orientiert. Zugriff auf die Datenbank erfolgt bei CouchDB über eine HTTP Schnittstelle, während MongoDB ein binäres Protokoll benutzt.
Grosse Unterschiede gibt es bei der Art und Weise wie Server in den beiden Systemen vernetzt werden:

CouchDB baut ein Netz aus *Master-Master* Verbindungen auf, die in beide Richtungen synchronisieren. Jede Einzelne Datenbank ist dabei unabhängig und kann alleine funktionieren.
MongoDB erstellt sogenannte *replica sets*, in denen es jeweils genau einen Master Server gibt. Alle Schreibzugriffe müssen auf diesem Master passieren.

Die Unterschiede werden weiterhin deutlich, wenn man sich die beiden Datenbanksysteme unter dem Aspekt des CAP-Theorems anschaut(siehe Figure 1).

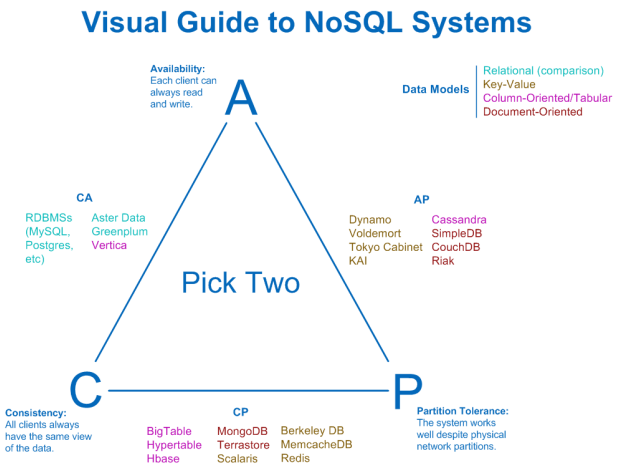

Figure 1: Das CAP Dreieck mit den entsprechenden Datenbanken. [6]

CouchDB präferiert die Verfügbarkeit von Daten gegenüber der Konsistenz und MongoDB im Gegenzug die Konsistenz seiner Daten deren Verfügbarkeit. Im Anwendungsfall bedeutet das, dass es in MongoDB Systemen zu sperren kommen kann weil gerade Daten geschrieben werden, während in CouchDB Systemen Lesezugriffe jederzeit möglich sind, es jedoch passieren kann, dass das erhaltene Dokument veraltet ist.

Performance unterscheidet die beiden Datenbanksysteme ebenfalls:
CouchDB führt bei jeder Schreiboperation einen vollen `fsync` aus, was höhere Performanceeinbusen mit sich bringt. Ausserdem verlangsamt die strikte *copy-on-write* und *append only* Policy Schreibvorgänge weiterhin.
MongoDB auf der anderen Seite kann hier durch *in-place* Updates, direkt auf dem Speicher, sehr hohe Performance erzielen.

[1]Offizielle Webseite: `https://www.mongodb.org/`

Auf Seiten der Sicherheit steht damit CouchDB allerdings vorne, da durch das ständige Wiederkopieren und Anhängen von Änderungen so gut wie keine Fehler bei der Datenhaltung auftreten können, was die Datenbank vor Fehlern beim Speichern macht. Bei MongoDB besteht durch die verwendeten *in-place* Updates die Möglichkeit, alte Daten zu zerstören.

MongoDB grenzt sich also von CouchDB vor allem durch die deutlich höhere Performance, den anderen Umgang mit verteilten Servern und der Art und Weise wie Queries erstellt werden ab. Eine genaue Aussage dazu, welches System besser ist als das andere, lässt sich nicht treffen.

6. ANWENDUNGSSZENARIEN

CouchDB hat seine Vorteile in der ausgiebigen Versionierung seiner Dokumente und der *Master-Master* Vernetzung in verteilten Systemen. Aufgrund des hohen Aufwandes beim Ändern von Daten, aber auf der anderen Seite der hohen Performance beim Lesen ist CouchDB vor allem für Anwendungen geeignet, in denen Daten selten Updates erfahren. Die Art und Weise wie Vernetzung in CouchDB funktioniert macht es ideal zum Verteilen von großen Datenbanken, die von vielen Orten aus gleichzeitig gelesen werden sollen. Als Anwendungsfall käme ein CDN(*Content Distribution Network*), bei dem Datenbanken auf Servern über die ganze Welt hinweg verteilt werden um lokal schnelleren Zugriff auf die Daten zu bekommen infrage. Das Verteilen von Programmcode (Patches, Updates, Code Repositories) bietet sich dank der implementierten Versionierung, parallelem Zugriff und der manuell anpassbaren Kollisionsbehandlung ebenfalls an.
Aufgrund der auf JSON, HTTP und Javascript aufgebauten Architektur sind CouchDB Datenbanken besonders für den Einsatz in webbasierten Anwendungen geeignet. Die Verwendung von nativen Webtechnologien wie dem HTTP Protokoll zur Interaktion mit der Datenbank, als auch webkomatible Formate wie JSON machen CouchDB ideal zur Nutzung für webbasierte Dienste und Anwendungen. Die Anwendung als Backend für Webdienste wird ebenfalls durch die simple Verteilung einer Datenbank auf mehrere Server und die hohe Geschwindigkeit beim Lesen von Daten befördert. Ein gutes Beispiel für eine auf CouchDB basierende Anwendung ist die Blogging Software Sofa[2], welche rein auf den Features von CouchDB aufbaut.

7. FAZIT

Wie in dieser Arbeit gezeigt wurde, handelt es sich bei dem CouchDB Datenbanksystem um eine Technologie, die in Sachen Geschwindigkeit und Verteilung mit der Menge an verwalteten Daten skaliert. Die einfach verständliche und mächtige HTTP API macht CouchDB optimal für Anwendungen im Web Bereich, was ebenfalls durch die leichte Replikation und dauerhafte Vernetzung von Datenbanken bestärkt wird.
Das Speichern von Daten in formlosen, sich selbst tragenden Dokumenten, statt in Tabellenform mit Referenzen auf andere Objekte innerhalb einer Datenbank zeigt seine Vorteile gerade im Bereich enormer Datenmengen. Probleme, wie sich im Laufe der Zeit ändernde Form von Daten, das Wegfallen oder Hinzukommen von Attributen, treten bei CouchDB dank der in sich abgeschlossenen Natur von Dokumenten nicht auf.
Abstriche müssen bei CouchDB jedoch bei der Performance gemacht werden. CouchDB priorisiert Datenstabilität, was viele Operationen zur Sicherung und Versionierung von Dokumenten erfordert und damit die Geschwindigkeit von Zugriffen im Vergleich mit anderen dokumentenbasierten Systemen deutlich senkt. Erwartet man also hohen Schreib- und Leseverkehr auf seiner Datenbank ist von CouchDB abzuraten.

[2]Sofa Github Repository: `https://github.com/jchris/sofa`

8. REFERENCES

[1] Offizielle CouchDB Hompage: `https://couchdb.apache.org/`.

[2] Definition von *B+-Bäumen*: `http://www.cburch.com/cs/340/reading/btree/`

[3] Jeffrey Dean and Sanjay Ghemawat *MapReduce:Simplified Data Processing on Large Clusters* 2004: `https://static.googleusercontent.com/media/research.google.com/en//archive/mapreduce-osdi04.pdf`

[4] J. Chris Anderson, Jan Lehnardt, Noah Slater *CouchDB: The Definitive Guide 1st Edition* 2010: `http://guide.couchdb.org/`

[5] Seth Gilbert, Nancy Lynchi *Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services* 2002 : ACM SIGACT News, Volume 33 Issue 2, June 2002, Pages 51-59 `https://dl.acm.org/citation.cfm?id=564601`

[6] Blog Post zum Thema CAP-Theroem `http://blog.scottlogic.com/2014/08/04/mongodb-vs-couchdb.html`

[7] Offizielle MongoDB Hompage: `https://www.mongodb.org/`

[8] Sofa GitHub Repository: `https://github.com/jchris/sofa/`