NoSQL - FlockDB

Christopher Göbel
Universität Siegen
chris-goebel@web.de

ABSTRACT

Das weltweite Datenvolumen verdoppelt sich alle 2 Jahre. Die traditionelle Art Daten zu speichern sind relationale Datenbanken. Diese sind jedoch teilweise nicht mehr für solch riesigen Datenmengen, wie sie in unserer Zeit durch soziale Medien wie Facebook und Twitter produziert werden können, ausgelegt. Aufgrund dessen sind diverse nichtrelationale Datenbanktechnologien entstanden. Diese Arbeit soll die von Twitter 2010 veröfentlichte FlockDB vorstellen und ihre Vor- und Nachteile hervorheben. Desweiteren beschäftigt sich die Arbeit mit dem Datenmodell, der Architektur und der Installation dieser Graphendatenbank, welche die Verbindung zwischen den einzelnen Nutzern von Twitter darstellt. [9]

Keywords

Datenbank,NOSQL, FlockDB

1. EINLEITUNG

In Zeiten von sozialen Medien wie Facebook und Twitter werden riesige zusammenhängende Daten, auch genannt *Big Data*, immer mehr zu einem Problem für klassische relationale Datenbanken. Diese sind teilweise nicht dafür ausgelegt, Abfragen auf solch großen Datenmengen effizient auszuführen. Dieses Problem hat viele neue Datenbanktechnologien entstehen lassen, welche sich unter dem Begriff NoSQL-Datenbanken zusammenfinden. Thema der vorliegenden Seminararbeit ist es eine der Datenbanken vorzustellen und abschließend zu bewerten. Die Arbeit lässt sich in drei Teile gliedern. Im ersten Teil werden Graphdatenbanken im Allgemeinen betrachtet, um ein Grundwissen zu vermitteln. Es werden Begriffe wie *Datenmodell*, *Traversierung* und *Sharding* erläutert. Darauf aufbauend befasst sich der nächste Abschnitt mit der FlockDB. Zunächst wird der Aufbau der Datenbank sowie das Zusammenspiel der einzelnen Komponenten geklärt. Anschließend wird erklärt wie Daten hinzugefügt, gelöscht oder abgefragt werden. Abschließend fasst der letzte Teil die Erkenntnisse zusammen, woraus sich Vor- und Nachteile sowie Anwendungsmöglichkeiten der FlockDB ableiten lassen.[14]

2. GRAPHENDATENBANKEN

Graphdatenbanken sind Datenbanken, die Graphen verwenden um Informationen zu speichern und abzubilden. Ein solcher Graph besteht aus Knoten, welche durch Kanten verbunden sind. Beide können Eigenschaften besitzen. Solche Datenbanken bieten meistens eine Reihe von Graphalgorithmen an, um beispielsweise den kürzesten Weg von Knoten A zu Knoten B zu berechnen. Durch diesen speziellen Aufbau wird eine schnelle Traversierung[1] ermöglicht. Dadurch eignet sich eine Graphdatenbank gut für die Arbeit mit vernetzten Informationen. Im folgenden Abschnitt werden einige Eigenschaften und Arbeitsweisen von Graphdatenbanken geklärt, welche wichtig für den weiteren Verlauf dieser Arbeit sind.[13]

2.1 Graphmodelle

Jede Graphdatenbank besitzt mindestens ein oder mehrere spezielle Graphenmodelle. Dabei unterscheidet man zwischen dem *einfachen Graph-Modell*, *Property Graph- Modell* und dem *Hypergraph-Modell*. Je nach Anforderung an die Datenbank wird entschieden, welches Modell am sinnvollsten ist. Beim einfachen Graph-Model, sowie bei allen anderen Modellen, spricht man von gerichteten Kanten, welche für eine einseitige Beziehung stehen. Ungerichtete Kanten hingegen werden für eine beidseitige Beziehung verwendet. Einseitig bedeutet, wenn zwischen den Knoten A und B eine Verbindung besteht, muss diese nicht zwangsläufig in beide Richtungen führen sowie es bei ungerichteten Kanten der Fall ist. Soll eine Entfernung zwischen A und B modelliert werden, so wird eine ungerichtete Kante genutzt, da die Entfernung in beide Richtungen dieselbe ist. Bei der Darstellung einer Route im Straßenverkehr werden gerichtete Kanten verwendet, da Straßen auch teilweise nur in eine Richtung befahrbar sind. Property- und Hypergraphmodell bauen beide auf dem einfachen Graph-Modell auf. Das Property-Graph-Modell unterscheidet sich vom einfachen Graph-Modell dadurch, dass Knoten und Kanten als Objekte gespeichert werden. Dies bedeutet also, dass beide nun mehrere Attribute fassen können. Jeder Knoten erhält hierbei immer eine eindeutige ID[2]. Der Hypergraph führt eine spezielle Art der Kante ein, die Hyperkante. Diese verbindet bei einem gerichtetem Graph den Startknoten mit

[1] Verfahren für das Durchlaufen eines Graphen
[2] Eine eindeutige Nummer zur Identifikation

mehreren Zielknoten, sodass keine weiteren Kanten nötig sind. Beim ungerichteten Graphen verbindet sie eine Menge von Knoten. Ansonsten unterscheidet sich der Hypergraph nicht von dem einfachen Modell. Ein Beispiel für einen ungerichteten Graph, nach dem einfachen Modell, ist in Figure 1 dargestellt.[10]

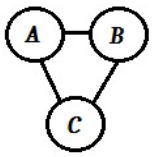

Figure 1: Graph nach einfachem Modell

2.2 Traversierung

Die Traversierung ist eine der Kernoperationen von graphorientierten Datenbanken. Bei ihr wird der Graph von einem Startknoten aus durchlaufen, meist um einen anderen Knoten innerhalb des Graph zu finden. Es existieren verschiedene Algorithmen, um eine solche Suche auszuführen. Zwei bekannte Suchoperationen sind Breiten- und Tiefensuche. Bei der Breitensuche werden zunächst alle ausgehenden Kanten des Startknotens betrachtet. Die nun erreichten Knoten werden in einer Warteschlange gespeichert, um nacheinander ihre ausgehenden Kanten zu überprüfen. Die Tiefensuche folgt einem Pfad solange, bis sie einen Knoten ohne ausgehende Kanten erreicht. Nun wird der Pfad rückwärts verfolgt, bis ein Knoten mit einer nicht makierten ausgehenden Kante gefunden wird. Dieser Ablauf wiederholt sich solange, bis der Startknoten keine unbesuchten Kanten mehr besitzt. Beide Operationen werden vorzeitig beendet, wenn der gesuchte Knoten gefunden wurde. Eine komplett durchgeführte Suche beider Algorithmen, welche zu keinem Ergebniss führt, wird in Figure 2 aufgezeigt. Dabei entsprechen die Zahlen der Reihenfolge der besuchten Knoten. [12]

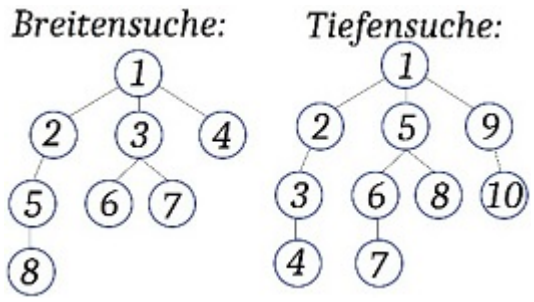

Figure 2: Breiten- und Tiefensuche

2.3 Adjazenzmatrix

Durch eine Adjazenzmatrix können Graphen dargestellt werden. Eine solche Matrix speichert für jeden Knoten die Nachbarn (ungerichtet) oder Nachfolger (gerichtet) ab. Sie besitzt für jeden Knoten eine Zeile und eine Spalte. Für n Knoten ergibt sich eine nXn Matrix. Die Darstellung des Graphen als Matrix hat zum Vorteil, dass sich auf diesen nun auch Methoden der Linearen Algebra anwenden lassen. Die Adjazenzmatrix für den in Bild 1 abgebildeten Graphen wird durch Tabelle 1 veranschaulicht. [2]

	a	b	c
a	0	1	1
b	1	0	1
c	1	1	0

Table 1: Adjazenzmatrix

2.4 Sharding

Unter Sharding versteht sich das Verteilen großer Datenmengen über mehrere zusammengeschaltete Systeme hinweg. Dazu werden zwei Methoden angewandt, die Partitionierung und die Replikation. Bei der Partitionierung werden die Daten anhand eines Sharding-Keys getrennt und auf einzelne Systeme aufgeteilt. Dieser Key beschreibt ein beliebiges Feld aus der Datenbank. Wenn nötig, kann auch eine Kombination aus beliebig vielen Feldern gewählt werden. Hierbei ist es wichtig einen sinnvollen Sharding-Key zu wählen, welcher eine gleichmäßige Verteilung der Daten ermöglicht. Bildlich gesprochen wird bei der Partitionierung einer Graphdatenbank der Graph an einer Stelle aufgetrennt und die einzelnen Teilgraphen auf verschiedene Systeme verteilt. Jedoch existiert keine exakte Methode um den Graph minimal aufzuteilen. Dadurch kann es vorkommen, dass Knoten in mehreren Teilgraphen liegen. Ist dies der Fall, handelt es sich um eine überlappende-Partitionierung. Die zweite Methode des Shardings ist die Replikation. Hierbei werden mehrere Kopien der Daten auf verschiedenen Systemen erstellt. So kann jedes System mit seiner Kopie effizient arbeiten. Zudem kann bei Ausfällen eines Systems die Kopie eines anderen verwendet werden.[7]

3. FLOCK DB

Auf Basis dieser Grundlagen über Graphdatenbanken wird nun im folgenden Abschnitt die FlockDB näher betrachtet. Die FlockDB ist eine graphorientierte Datenbank, welche im Jahre 2010 von der Social Media Plattform Twitter veröffentlicht wurde, um die Verbindung der einzelnen User zu speichern. Sie unterscheidet sich im wesentlichen durch den Verzicht auf Traversierungsmethoden von den meisten herkömmlichen Graphendatenbanken. Benötigt wird diese Funktion nicht, da Twitter nur an den direkten Verbindungen zwischen den Usern interessiert ist. Für die „Tweets“, also die Nachrichten welche auf Twitter gepostet werden, ist die FlockDB nicht zuständig. Im Vordergrund stehen bei Flock schnelle Lese- und Schreibzugriffe und das Arbeiten mit großen Adjazenzmatrizen. Um diese Matrizen zu speichern, verwendet FlockDB eine MySQL Datenbank als Basis, da diese während der Entwicklung ausgiebig getestet wurde. „[...]because we understand its behavior - in normal use as well as under extreme load and unusual failure conditions“[4]. Diese Tabellen sehen ähnlich wie die aus Abschnitt 2.3 aus. FlockDB kann bis zu 20.000 Schreibvorgänge und bis zu 100.000 Lesezugriffe pro Sekunde abwickeln. Diese Werte stammen aus dem Jahre 2010. Ob sie noch den aktuellen Stand widerspiegeln ist unklar. Auch ob Twitter ihre hauseigene Datenbank noch nutzt, ist nicht bekannt. Das letzte Update des Quellcodes auf der Website liegt jedoch 3 Jahre zurück. Im folgenden Abschnitt werden die einzelnen Komponenten der FlockDB näher betrachetet sowie das Datenmodell welches verwendet wird. Abschließend werden

noch anhand von Beispielen theoretische Abfragen auf der Datenbank ausgeführt.[4]

3.1 Datenmodell

Das Datenmodell der FlockDB ist fest vorgegeben und kann nicht angepasst werden. Es ist vom Typ Property-Graph-Modell. Der Graph besteht also aus Knoten und Kanten, welche mehrere Attribute besitzen können. Jeder Knoten wird mit einer 64-Bit integer ***source-id*** ausgestattet. Im Sozialen Netzwerk entspricht diese der eindeutigen ID eines Users. Die Kanten erhalten eine ***destination-id***, welche die eindeutige ID des User ist, dem gefolgt werden soll. Zusätzlich erhält sie eine **position-id**, in welcher ein Zeitstempel gespeichert wird. Durch diesen lässt sich später darstellen, in welcher Reihenfolge User einem anderen User „gefollowt" sind. Der Primärschlüssel für die MySQL-Datenbank ist eine Kombination aus den drei Feldern **source-id**, **state** und **position-id**. Im Feld ***state*** steht der Zustand einer Kante als 8-Bit integer. Kanten werden immer zweimal gespeichert, je Richtung einmal. So lässt sich darstellen „Wem folge ich?" und „Wer folgt mir?". Ersteres wird nach der **source-id** partitioniert, zweiteres nach der **destination-id**. So kann garantiert werden, dass diese zusammenhängenden Daten auf einer Partiotion liegen, um Abfragen effizient durchführen zu können. Jede Kante kann sich in 3 Zuständen befinden:

normal: Beide User sind aktiv und die Followerbeziehung besteht.

removed: Löst ein User die Followerbeziehung auf, so erfolgt kein direktes löschen in der Datenbank. Die Kante wird nur entsprechend markiert.

archived: Wenn ein Account gelöscht wird, sei es vom Benutzer selbst oder von Twitter, so werden alle seine Beziehungen in Form der Kanten mit dem Status archived versehen. Wenn der Account innerhalb einer vorgegebenen Zeit reaktiviert wird, können alle Beziehungen wiederhergestellt werden. Ansonsten folgt ein endgültiges löschen der Daten.[4, 3]

3.2 Architektur

Die FlockDB ist aus mehreren Komponenten zusammengesetzt (Figure 3). In diesem Teilabschnitt sollen die Komponenten näher betrachtet werden.

Stellt ein Client eine Anfrage, wird sie von der Middleware angenommen. Diese besteht aus einer beliebigen Anzahl von FlockDB Application Servern, auch *Flapps* genannt. Für die Partitionierung auf die einzelnen persistenten[3] Datenspeicher, welche aus MySQL-Datenbanken bestehen, verwenden die Flapps das Framework *Gizzard*. Web-Clienten und Flapps kommunizieren über das *Apache Thrift Framework* als Schnittstelle. Neben den Anbindungen an Client und Datenspeicher besitzen die Application Server auch einen Port für die Administration der Server, welche über das Konsolentool *Gizzmo* abläuft.

[3]Persistenz ist in der Informatik die Fähigkeit Daten über einen langen Zeitraum zu speichern

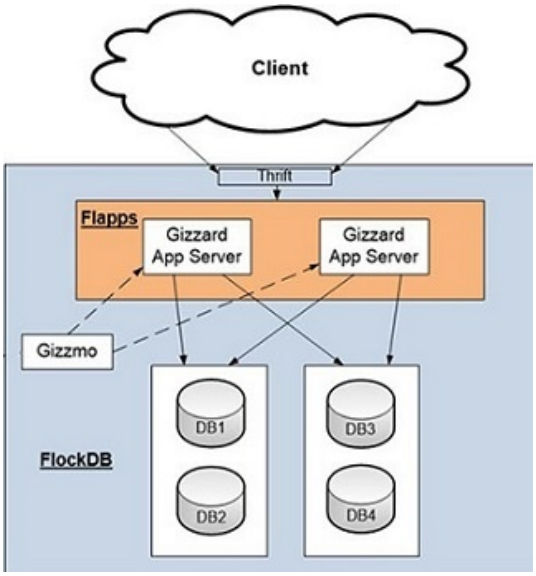

Figure 3: Architektur [1]

3.2.1 Gizzard

Gizzard ist ein von Twitter entwickeltes Framework, geschrieben in Scala, welches in einer JVM (Java Virtuell Machine) ausgeführt wird. Es soll bei dem Sharding, also Partitionierung und Replikation von großen Datenmengen, helfen. Dabei befindet sich Gizzard genau zwischen dem Nutzer und den Daten. Alles was aufgerufen oder geändert wird, läuft durch Gizzard. Dabei ist es egal welches Datenbanksystem zum Einsatz kommt, solange es netzwerkfähig ist. Es können also relationale Datenbanken wie MySQL oder Key-Value-Speicher wie Redis verwendet werden. Eine weitere Vorraussetzung ist jedoch, dass die Datenbank nur idempotente[4] sowie kommutative[5] Schreiboperationen verwendet. Gizzard kann nicht garantieren, dass Datensätze in zeitlich korrekter Reihenfolge abgelegt werden. Dadurch könnten Fehler bei Operationen, welche nicht idempotent und kommutativ sind, entstehen. Um das Sharding durchzuführen, nutzt Gizzard die zuvor beschriebenen Methoden der Partitionierung und Replikation.

Bei der Partitionierung weist Gizzard ausgewählte Bereiche von Daten einzelnen Shards[6] zu. Welche Daten sich in welcher Shard befinden, wird in einer Weiterleitungstabelle gespeichert. In Figure 4 ist ein Beispiel einer solchen Weiterleitungstabelle zu sehen. Hier wurden die Daten 0-9 Shard 1 zugeordnet. 10-19 befinden sich in Shard 2. Gizzard wählt über eine Hash-Funktion einen Bereich aus, an der die Daten partiotioniert werden sollen. Diese Funktion lässt sich durch den Nutzer anpassen, wodurch sich die Verteilung der Daten optimieren lässt.

[4]Eine Funktion die bei mehrfachem Aufruf das gleiche Ergebnis liefert wie bei einem einzigen Aufruf
[5]Argumente in einer Operation können vertauscht werden, ohne dass sich das Ergebnis ändert
[6]Eine Partition von Daten in einer Datenbank

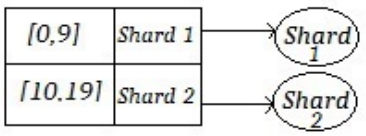

Figure 4: Partitionierung

Die Replikation wird über eine Baumstruktur geregelt. Jede Shard, die in der Weiterleitungstabelle vermerkt ist, kann entweder eine logische oder eine physische Shard sein. Eine physische Shard stellt einen Datenspeicher, wie eine SQL Datenbank, dar und ist immer ein Blattknoten in der Baumstruktur. Das bedeutet, unter ihr kann im Baum nichts mehr stehen. Eine logische Shard wiederum ist ein Elternknoten mit weiteren Shards als Kindknoten. Diese können logischem oder physichem Ursprungs sein. Logische Shards beinhalten normalerweise Regeln für den Umgang mit ihren Kindknoten. Es kann vorgegeben werden wie Lese- und Schreiboperationen ablaufen müssen. Gizzard liefert einige solcher Regeln und je nach Nutzen können weitere eingefügt werden. Ein möglicher Aufbau einer Baumstruktur ist in Figure 5 zu sehen.

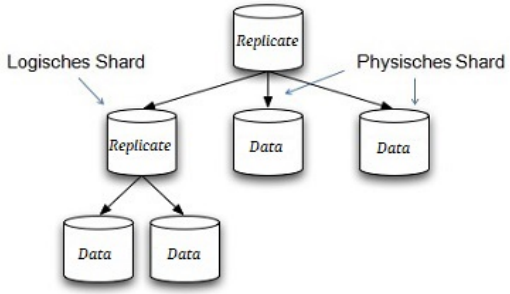

Figure 5: Replikation [1]

Um einen oder mehrere Server zu entlasten, ist es manchmal notwenig Daten auf einen neuen Server zu kopieren oder zu verschieben. Da die FlockDB horizontal skaliert, können nach belieben neue Server an das System angeschlossen werden. „[...]servers that we can expand as needed. Over the winter, we added 50% database capacity without anyone noticing.“[4] Sollen Daten von einem Server A auf einen Server B kopiert werden, kann es zu Fehlern führen, wenn unfertig kopierte Daten von B gelesen werden. In diesem Fall nutzt man eine logische Shard. Vor Server B wird eine Shard mit der Regel *write-only* gesetzt, bis der Kopiervorgang abgeschlossen ist. So ist der Lesezugriff für alle Nutzer ausgeschlossen.

Sollte eine Replika[7] ausfallen, so leitet Gizzard eine bestehende Anfrage direkt an die verbleibenden Replika weiter. Wenn jedoch alle Replikationen unerreichbar sind, kommt die Anfrage in eine Warteschleife und wird ausgeführt, sobald es wieder möglich ist. Wenn eine Replikation in der Shard vor den anderen erreichbar ist, wird die Operation direkt auf sie angewendet. Die anderen verweilen in der Warteschleife. An diesem Punkt ist es wichtig, dass alle Operationen der Datenbank idempotent und kommutativ sind, da sie in einem solchem Fall asynchron ausgeführt werden.

Um sämtliche Verhaltensweisen der Datenbank anzupassen, wird das Kommandozeilentool Gizzmo verwendet. Mit ihm lassen sich die Gizzard Application Server administrieren. Dies ermöglicht unteranderem das Sperren oder Freigeben von Shards. Ihre Gewichtung lässt sich auch ändern, was beeinflusst, welche Shard bei einer Anfrage zuerst aufgerufen wird. [5, 6]

3.2.2 Apache Thrift

Zwischen dem Nutzer und den Gizzard Application Servern liegt das Apache Thrift Framework. Dabei handelt es sich um ein Kommunikationsprotokoll zur Erstellung interoperabler[8] Services. Thrift wurde ursprünglich von der Social Media Seite Facebook entwickelt, ist nun aber ein Open-Source Apache Projekt und wurde unter der Apache-2.0-Lizenz veröffentlicht. Thrift hilft im Falle der FlockDB bei der Kommunikation zwischen Usern und Flapps. Wenn ein Client versucht mit der Datenbank zu kommunizieren, jedoch eine abweichende Programmiersprache von der FlockDB hat, wird seine Sprache angepasst. Thrift ist eine Ansammlung von Code-Generatoren, welche den betreffenden Quellcode in einen identischen Code einer ausgewählten Zielsprache umwandeln. Dafür müssen einige Angaben zu Services und Datentypen in einer Definitionsdatei vorgenommen werden. Thrift unterstützt Sprachen wie C++, Java, Python, Ruby, usw.[11]

3.3 Datenabfrage

Nach der Installation des Ruby-Clients kann mit der Datenbank kommuniziert werden. Zuerst muss eine Instanz der Datenbank erstellt werden. Dies könnte folgendermaßen aussehen: ***flock = Flock.new 'localhost:7915', :graphs => :follows => 1***
Der Name der Instanz lautet in unserem Beispiel flock und wird auf dem Server localhost, also auf dem momentan genutzen System erstellt. Der Port für die Verbindung ist 7915 und es wird der Graph follows betrachtet. Um der FlockDB Daten in Form von Kanten hinzuzufügen wird der Befehl ***Database.add (Source-id, :follow, Destination-id)*** verwendet. Dabei steht Database für den Namen der Datenbank, Source-id für die eindeutige Kennung des Nutzers welcher jemandem folgen will und Destionation-id für die Kennung des Nutzers dem gefolgt wird. Wollen wir nun das User 1 User 2 folgt, würde der Befehl ***flock.add(1, :follows , 2)*** lauten. Diese Verbindung wird jeweils in der Vorwärts- und der Rückwärtsrichtung gespeichert. Dadurch lässt sich herausfinden vom wem User 1 verfolgt wird und wem er folgt. Um eine solche Abfrage auszuführen wird der Befehl ***Database.select(nil, :follows, User-id)*** genutzt. Setzt man für User-id die 1 ein, erhält man alle follower von User 1. Werden *nil* und User-id getauscht, so erhält man alle User die von 1 verfolgt werden. *Nil* lässt sich hier als ein Platzhalter für die gesuchte Menge an Usern verstehen. Um eine Verbindung wieder zu löschen, existiert der Befehl ***Database.remove(Source-id, :follows, Destination-id)***. Dabei wird die Kante, wie zuvor erwähnt, nicht aus der

[7] Nachbildung

[8] Interoperabilität ist die Fähigkeit zur Zusammenarbeit von verschiedenen Systemen

MySQL-Datenbank gelöscht. Diese wird nur mit dem removed state versehen. Bei der Verwendung der FlockDB bei Twitter können die Result-Sets[9] teils sehr groß werden. Um nicht die Übersicht zu verlieren, lässt sich jede Abfrage auch seitenweise ausgeben. Dazu muss die Abfrage wie folgt geschrieben werden: ***pager = flock.Abfrage...***. Durch den Konsolenbefehl ***pager.next_page*** wird die nächste Seite der Abfrage aufgerufen.

Neben diesen grundlegenden Befehlen lassen sich auch Schnittmengen, Differenzen und Vereinigungen bilden. Dazu werden die Befehle ***.union*** für die Vereinigung, ***.difference*** als Differenz und ***.intersect*** für die Schnittmenge verwendet. [8]

4. FAZIT

In dieser Arbeit wurden zunächst die Grundlagen von Graphendatenbanken erklärt. Dazu gehört das Sharding, also das Aufteilen von großen Datenmengen auf verschiedene Partitionen. Außerdem die Traversierung, welche eigentlich eine Kernoperationen für graphenbasierte Datenbanken darstellt. Für die FlockDB ist sie eher unwichtig, aber dennoch erwähnenswert. Nach dem allgemeinen Teil wurde die Twitter Datenbank genauer betrachtet. Ihr Datenmodell ist vom Typ *Property-Graph* und sie verwendet eine MySQL-Datenbank als Grundlage für die Datenspeicherung. In dieser können jedoch nur die Adjazenzmatrizen, welche die Follower Beziehung enthalten, gespeichert werden. Nun wurden noch die einzelnen Komponenten betrachtet, welche zusammen die FlockDB darstellen. Ein Teil davon, das Gizzard Framework, welches auch von Twitter entwickelt wurde, wird für das Sharding verwendet. Daten werden hier mittels einer anpassbaren Hash-Funktion partitioniert. Eine Replikation erfolgt durch eine Baumstruktur, bestehend aus physischen und logischen Shards. Abschließend wurden einige Beispiele zur Datenabfrage aufgeführt.

Im Verlauf der Arbeit wurde schnell deutlich, dass die Anwendungsgebiete der FlockDB sehr begrenzt sind. Dies resultiert aus ihrem speziellem Aufbau, der keine Traversierung unterstützt. Somit kommen fast nur Anwendungen in Frage, welche eine ähnliche Struktur wie Twitter aufweisen. Es muss also Nutzer geben, die in irgendeiner Beziehung zueinander stehen. Die FlockDB ist für diesen konkreten Aufgabenbereich optimiert worden. Sie verfügt über schnelle Zugriffszeiten mit ca. 20.000 Schreibvorgänge und 100.000 Lesezugriffe in der Sekunde. Da es sich bei der Twitter Datenbank um ein OpenSource Projekt handelt, ist sie zudem kostenlos erhältlich. Einen Support seitens der Entwickler kann man allerdings nicht erwarten. Dies könnte sich als problematisch bei der Nutzung herausstellen, da kaum Dokumentationen über die Datenbank im Netz existieren. Besitzt man nun eine Anwendung, die eine Beziehung zwischen den Nutzern darstellen möchte, ist die FlockDB eine gute Wahl. Für andere Problemstellungen jedoch ist sie eher ungeeignet.

[9]Die Ausgabe der gesuchten Daten

5. REFERENCES

[1] FH Köln - FlockDB. Abgerufen Dezember 2015 von http://wikis.gm.fh-koeln.de/wiki_db/Datenbanken/FlockDB

[2] Reinhard Diestel: Graphentheorie. 4. Auflage. Springer, Berlin u. a. 2010, ISBN 978-3-642-14911-5.

[3] GitHub - FlockDB. Abgerufen Dezember 2015 von https://github.com/twitter/flockdb

[4] Twitter - Blog - Introducing FlockDB. Abgerufen Dezember 2015 von https://blog.twitter.com/2010/introducing-flockdb

[5] GitHub - Gizzard. Abgerufen Dezember 2015 von https://github.com/twitter/gizzard

[6] GitHub - Gizzmo. (2012). Abgerufen Dezember 2015 von https://github.com/twitter/gizzmo/

[7] FH Köln - Sharding. Abgerufen Dezember 2015 von http://wikis.gm.fh-koeln.de/wiki_db/Datenbanken/Sharding

[8] GitHub - FlockDB Demo. Abgerufen Dezember 2015 von https://github.com/twitter/flockdb/blob/master/doc/demo.markdown

[9] EMC Datenvolumen Studie. Abgerufen Dezember 2015 von http://www.emc.com/leadership/digital-universe/2014iview/executive-summary.htm

[10] FH Köln - Graphmodelle. Abgerufen Dezember 2015 von http://lwibs01.gm.fh-koeln.de/wikis/wiki_db/index.php?n=Datenbanken.Graphenmodell

[11] Apache Thrift Tutorial. Abgerufen am 08. Dezember 2015 von http://thrift-tutorial.readthedocs.org/en/latest/intro.html

[12] FH Köln - Breiten und Tiefensuche. Abgerufen Dezember 2015 von http://lwibs01.gm.fh-koeln.de/wikis/wiki_db/index.php?n=Datenbanken.BreitenTiefensuche

[13] FH Köln - Graphdatenbank. Abgerufen Dezember 2015 von http://lwibs01.gm.fh-koeln.de/wikis/wiki_db/index.php?n=Datenbanken.Graphendatenbank

[14] BigData und NoSQL - Abgerufen Dezember 2015 von http://www.bigdata-insider.de/relationale-datenbanken-sind-nicht-immer-ideal-a-472678/