Hochschule für Technik, Wirtschaft und Kultur Leipzig
Fakultät Informatik, Mathematik und Naturwissenschaften

Cassandra als Beispiel eines Wide Column Store

im Oberseminar „Datenbanksysteme - Aktuelle Trends“

Von: Martin Louis Dearnley

Studiengang: Informatik Master | 17-INM

Datum: 9. Juli 2018

Inhaltsverzeichnis

1 Einleitung

Im Buch „Cassandra: The Definitive Guide" von Jeff Carpenter & Eben Hewitt wird Cassandra wie folgt beschrieben:

„Apache Cassandra is an open source, distributed, decentralized, elastically scalable, highly available, fault-tolerant, tuneably consistent, row-oriented database that bases its distribution design on Amazon's Dynamo and its data model on Google's Bigtable. Created at Facebook, it is now used at some of the most popular sites on the Web."

Cassandra ist ein verteiltes NoSQL-Datenbankmanagementsystem für die Verwaltung großer Mengen an Daten, die auf vielen Cluster verteilt sein können. Cassandra zielt darauf ab, auf einer Infrastruktur von Hunderten von Knoten, z. B. über verschiedene Datenzentren verteilt, zu laufen. Entwickelt wurde Cassandra bei Facebook von Avinash Lakshman and Prashant Malik um eine Inbox-Suche bereitzustellen, die es den Usern ermöglicht, nach gesendeten und erhaltenden Nachrichten ohne Verzögerung zu suchen. Hauptaugenmerk lag bei der Entwicklung auf Verfügbarkeit und der Skalierbarkeit der Server. Abstriche wurden dagegen auf die Konsistenz der Daten gemacht. Als Vorbild dienten die Wide Column Stores Amazon Dynamo und Google BigTable. Von Dynamo wurde das Architekturmodell übernommen, von BigTable das Datenmodell. Cassandra wurde im Juli 2008 als Open Source Projekt veröffentlich, im März 2009 wurde es in den Apache Incubator aufgenommen und im Februar 2010 als Top-Level-Projekt erklärt.

2 Datenmodell

Ein Cluster einspricht einem Datenbankserver und dieser enthält eine oder mehrere Datenbanken (Keyspaces). Daten werden dabei in Column Familes organisiert, welche aus beliebig vielen Row keys bestehen, die wiederum beliebig viele Schlüssel-Werte-Paare zugeordnet werden können. Zu jedem Wert wird noch ein Zeitstempel gespeichert. Im Gegensatz zu einem relationalen Datenbankmanagementsystem werden die Tabellen in Cassandra nicht normalisiert. Informationen werden redundant gespeichert, da JOINS nicht performant sind und deshalb nicht unterstützt werden.

3 Architektur

Cassandra-Datenbanken werden im Cluster, welches aus mehreren Knoten besteht. Cassandra ist konzipiert Daten auf mehreren verteilten Knoten zu speichern ohne Single Point of Failure, was bedeutet, dass im Cluster jeder Knoten die gleiche Rolle einnimmt, dadurch gibt es keine zentrale Steuerung, keinen Master-Knoten. Zur Datenverteilung kommunizieren die Knoten miteinander über dem Peer-to-Peer Protokoll Gossip. Daten werden über einen Hashwert verteilt. Anhand des Primärschlüssel eines Datensatzes wird ein Hashwert berechnet. Der Knoten, der für diesen Hashbereich zuständig ist, speichert diesen Datensatz. Der Bereich, für den ein Knoten zuständig ist, liegt zwischen -2^{63}

und $+2^{63}$ und wird über alle Knoten gleichmäßig verteilt. Ein Cassandra-Cluster ist horizontal skalierbar. Eine Verdoppelung der Knotenanzahl hat zur Folge, dass sich die Datenmenge einzelner Knoten halbiert. Da alle Knoten gleichberechtig sind, kann jeder Knoten Requests empfangen bzw. beantworten. Der erste Knoten ist dabei der Koordinatorknoten, welche die Schreib- oder Leseanfrage an die passenden Knoten weiterleitet.
Um den Single Point of Failure zu vermeiden, werden Daten Redundant in einem Cluster gespeichert. Dafür gibt es unterschiedliche Strategien zum Lesen, Schreiben und Duplizieren von Daten:

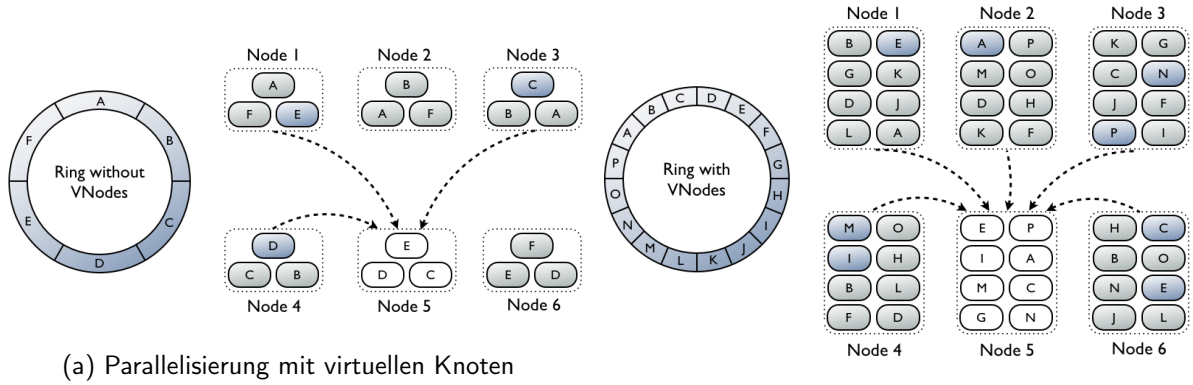
- **Replication Level (RL):** Gibt an, wie oft Daten auf andere Knoten dupliziert werden sollen. Ein RL von 3 bedeutet, dass in einem Cluster auf drei Knoten verteilt die Daten repliziert werden. Jede Replication ist gleichwertig.
- **Write Level:** Mit dem Write Level kann das Konsistenzniveau bestimmt werden, durch z. B. ONE, ALL, QUORUM, LOCAL_QUORUM. Je höher das Konsistenzniveau, desto wahrscheinlicher ist im Falle eines Knotenausfalles, dass der Datensatz, der auf anderen Knoten gespeichert ist, der aktuelle Datensatz ist. Bei einem Schreibprozess werden Daten zuerst in einen Commit log und dann in den Arbeitsspeicher geschrieben. Der Speicher wird regelmäßig geleert und in einer Sorted String Table (SSTable) gespeichert. Eine SSTable ist eine unveränderbare Datei. Mehrere Tabellen werden in periodischen Abständen miteinander verglichen und zu einer SSTable gemergt.
- **Read Level:** Wenn ein Datensatz von mehreren Knoten gelesen wird, ist die Wahrscheinlichkeit höher, dass der gelesene Datensatz, der aktuellste ist. Beispiel: Wenn ein Datensatz mit QUORUM in das Cluster geschrieben und anschließend mit QUORUM wieder gelesen wird, ist sichergestellt, dass an den Client den aktuellsten Datensatz weitergeleitet wird.

Nähere Informationen zu den Level finden sich hier: Datastax Dokumentation.

3.1 Virtuelle Knoten

Ein weiterer Bestandteil der Architektur von Cassandra ist es, virtuelle Knoten zu verwenden. Durch virtuelle Knoten ist nicht jeder Knoten für primär einen Hashbereich zuständig, sondern für mehrere kleine Bereiche. Wenn z. B. ein neuer Knoten hinzugefügt wird, dann verringert sich der Hash-Bereich des Knoten seines Vor- und Nachfolgers.
Die Verteilung der Hashwerte und die Parallelisierung von Cluster ohne virtuelle Knoten verhält sich wie folgt:

- **Verteilung der Bereiche:** Möchte man in einem Cluster ohne virtuelle Knoten einen neuen Knoten in dem Ring hinzufügen, müssten, damit die Bereiche der Knoten ausbalanciert werden, die Hashbereiche der Knoten manuell neu verteilt werden. Um diesen Prozess zu vereinfachen, wurden immer dieselbe Knotenanzahl eingefügt. Bei sechs Knoten in einem Cluster wurden weitere sechs Knoten hinzugefügt. Die Datenlast der Knoten halbiert sich dadurch.



(a) Parallelisierung mit virtuellen Knoten

(b) Parallelisierung mit virtuellen Knoten

Abbildung 1: Vergleich der Parallelisierung zwischen Cluster mit und ohne virtuelle Knoten. Quelle: Datastax

- **Parallelisierung:** Angenommen, ein Knoten von den sechs Knoten in einem Cluster fälllt aus. Bei einem Lese- oder Schreibzugriff wird die Last auf den vorhandenen Knoten, die eine Replication der Daten enthalten, verteilt. Sobald der ausgefallene Knoten wiederhergestellt ist, müssen diese Knoten erneut eine höhere Last auf sich nehmen, um Informationen an den wiederhergestellten Knoten neu verteilen zu können.

Durch virtuelle Knoten ist nicht jeder Knoten für primär einen Bereich zuständig, sondern für mehrere Hashbereiche.

- **Verteilung der Bereiche:** Wenn z. B. ein neuer Knoten hinzugefügt wird, dann werden die Hashbereiche in einem Cluster automatisch balanciert.
- **Parallelisierung:** Bei einem Ausfall eines Knoten in einem Cluster, verteilt sich die Last auf allen Knoten.

4 CAP Theorem

Das CAP Theorem besagt, dass es unmöglich ist, dass in einem verteilten System, gleichzeitig die Konsistenz, Verfügbarkeit und Ausfalltoleranz garantiert werden kann. Cassandra ist einstellbar konsistent. Das bedeutet, dass durch das read level, write level und dem Replikationsfaktor bestimmt werden kann, ob Cassandra Verfügbar und Ausfalltolerant (AP) oder Konsistenz oder Ausfalltolerant (CP) ist. Welche Einstellung zu erhöhten Konsistenz oder zur besseren Verfügbarkeit führt, lässt mit folgendem Webtool ausprobieren: CassandraCalculator.

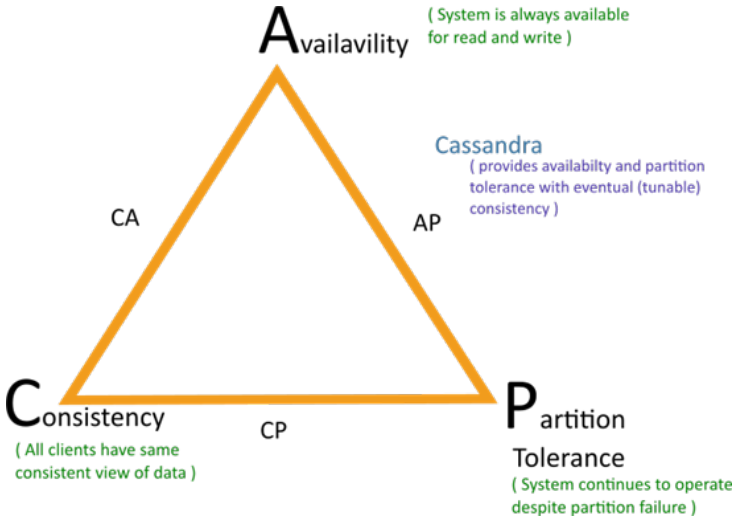


Abbildung 2: Cassandra wurde ursprünglich für AP konzepiert. Konsistenz ist aber regelbar. Quelle: toadworld

5 Cassandra Query Language (CQL)

Die Cassandra Query Language ist eine an SQL angelehnte Anfragesprache, welche für die Cassandra Datenbanken verwendet wird. Die einfachste Art, mit Cassandra zu interagieren, ist die Verwendung der CQL-Shell, cqlsh. Cassandra bildet allerdings eine Ausnahme unter den Wide Column Stores bezüglich der Behandlung der Spalten. Hier müssen neue Spalten mit ALTER TABLE hinzugefügt werden.
Mit CQL lassen sich Datenstrukturen anlegen mit z. B. CREATE TABLE, Daten abfragen mittels SELECT oder Daten manipulieren durch INSERT, UPDATE, DELETE. Besonderheit ist, dass JOINS und GROUP BY oder FOREIGN KEY nicht unterstützt werden.
Informationen zu den unterstützten Datentypen findet sich hier: Datastax-Dokumentation.

6 Demonstration

Es wurde anhand von Twissandra, eine in Python geschriebenes Beispielprojekt, Anwendungsfälle von Cassandra gezeigt. Quelle: Twissandra.
Ein Auszug des gezeigten CQL-Codes ist im Anhang wiederzufinden.

7 Fazit

Mit Apache Cassandra lassen sich nicht, im Gegensatz zu der Wide Column Store Hbase, zur Laufzeit neue Spalten hinzufügen, sondern die Spalten müssen vorher definiert sei. Die Datenbank zeichnet sich dagegen durch schnelle Lese- und Schreibzugriffe, dass es keinen Single Point of Failure gibt und durch die horizontale Skalierbarkeit der Cluster aus. Wer jedoch ein relationales Datenbankmanage-

mentsystem gewohnt ist, muss sich beim Modellierung darauf einstellen, dass das Datenmodell nicht Normalisierung wird, da keine JOINS und Subqueries auf Performanzgründen unterstützt werden.

8 Anhang

```
describe keyspaces;
use twissandra;
describe tables;

select * from users;

insert into users(username, password) values ('trump', 'donald');

//typo: update
update users set password ='secret' where username = 'trump';

select * from users;

select token(username) from users;
select token(username) from users where username = 'trump';

//cmd
nodetool status
nodetool ring

//trump ttl
insert into tweets(tweet_id, body, username) values (uuid(),
    'Klimawandel? fake news', 'trump');
select * from tweets;
//id kopieren
insert into timeline(username, time, tweet_id) values
    ( 'trump', now(), 22fb8a4f-1c90-4176-b388-0a240ca53e8e) using TTL 10;

python manage.py fake_data 10 100;

select * from tweets;

//Error "Cannot execute this query as it might involve data filtering
```

```
    and thus may have unpredictable performance.
//If you want to execute this query despite the performance unpredictability,
    use ALLOW FILTERING"
select * from tweets where username = 'trump';
//geht, sollte aber nicht gemacht werden
select * from tweets where username = 'trump' allow filtering;

// eigenes twitter //

//Datenbank erstellen mit SimpleStrategy, was bedeutet:
//simpleStrategy: Use only for a single datacenter and one rack.
    Replicas im Uhrzeigersinn.
//NetzworkTopologyStrategy: Wenn cluster verteilt sind
create keyspace twitter with replication =
{'class':'SimpleStrategy', 'replication_factor':1};
use twitter;

//1 Primary Key un followers ist ein set. --sets -> werte unique
create table users(user_id text,followers set<text>,
        tweet_date timestamp, tweet_body text, first_name text,
PRIMARY KEY(user_id));

// insert records for 'Dearnley'
// insert records for 'Dearnley'
 insert into users(user_id, followers, tweet_date,tweet_body,first_name)
values('Dearnley',{'Trump', 'Obama'}, '2013-12-31','lol','Louis');
 insert into users(user_id,tweet_date,tweet_body,first_name)
values('Dearnley','2014-01-01','rofl','Louis');
 insert into users(user_id,tweet_date,tweet_body,first_name)
values('Dearnley','2014-01-04','lolol','Louis');

// insert records for 'Trump'
 insert into users(user_id,tweet_date,tweet_body,first_name)
values('Trump','2013-12-21','2017 bye bye','Donald');
 insert into users(user_id,tweet_date,tweet_body,first_name)
values('Trump','2014-01-01','2018! another exciting year','Donald');
 insert into users(user_id,tweet_date,tweet_body,first_name)
values('Trump','2014-03-25','Fake news!','Donald');
```

```
//2 Zeilen weil nur 1 Primary key
select * from users;

select token(user_id) from users;

//clusterin keys  --sets -> werte unique
drop table users;
create table users(user_id text,followers set<text>,
        tweet_date timestamp, tweet_body text, first_name text,
PRIMARY KEY(user_id,tweet_date, first_name));

// insert records for 'Dearnley'
 insert into users(user_id, followers, tweet_date,tweet_body,first_name)
values('Dearnley',{'Trump', 'Obama'}, '2013-12-31','lol','Louis');
 insert into users(user_id,tweet_date,tweet_body,first_name)
values('Dearnley','2014-01-01','rofl','Louis');
 insert into users(user_id,tweet_date,tweet_body,first_name)
values('Dearnley','2014-01-04','lolol','Louis');

// insert records for 'Trump'
 insert into users(user_id,tweet_date,tweet_body,first_name)
values('Trump','2013-12-21','2017 bye bye','Donald');
 insert into users(user_id,tweet_date,tweet_body,first_name)
values('Trump','2014-01-01','2018! another exciting year','Donald');
 insert into users(user_id,tweet_date,tweet_body,first_name)
values('Trump','2014-03-25','Fake news!','Donald');

select * from users;

select token(user_id) from users;
```

Abbildungsverzeichnis

Literaturquellen

1]

Bücher

[CH17] Jeff Carpenter und Eben Hewitt, Hrsg. *Cassandra The Definitve Guide*. O'Reilly Media, 2017. ISBN: 978-1-491-93366-4.

[Mis14] Vivek Mishra, Hrsg. *Beginning Apache Cassandra Development*. Springer, 2014. ISBN: 978-1-4842-0143-5.

Artikel

[Phi15] Marc Jansing Phillip Ghadir. "Apache Cassandra". In: (Feb. 2015), S. 62–65. URL: https://www.sigs-datacom.de/uploads/tx_dmjournals/ghadir_jansing_JS_02_15_tJPH.pdf.

[Phi16] Martin Hemes und Philip Stroh. "Apache Cassandra im Projekteinsatz". In: (Mai 2016), S. 1–9. URL: https://www.timocom.de/STATIC/binary/900000/pdf/de/Sonderdruck_TimCom_JM5_16_Stroh_Herms_36433_v1_druck.pdf.

Weblinks

[AP] Facebook Avinash Lakshman und Facebook Prashant Malik. *Cassandra A Decentralized Structured Storage System*. URL: https://docs.datastax.com/en/articles/cassandra/cassandrathenandnow.html.

[Dat] DataStax. *DataStax Docs*. URL: https://docs.datastax.com/en/landing_page/doc/landing_page/current.html.

[Hur10] Nathan Hurst. *Visual Guide to NoSQL Systems*. 2010. URL: http://blog.nahurst.com/visual-guide-to-nosql-systems.

[mic] michellebanzondarling. *Cassandra tutorial*. URL: https://de.slideshare.net%20/michellebanzondarling/cassandratutorialfinalversion.

[Sop] Maitrey J. Soparia. *Apache Cassandra (Distributed Hash Table)*. URL: http://salsahpc.indiana.edu/b534projects/sites/default/files/public/1_Cassandra%20Distributed%20Hash%20Table_Soparia%2C%20Maitrey%20Jagdish.pdf.