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

Abstract

Wide Column Stores: Cassandra

Oberseminar Datenbanksysteme - Aktuelle Trends

vorgelegt von: Christofer Schwach
Studiengang: MIM-16

Leipzig, den 30.06.2017

1. Einleitung

Cassandra ist eine sehr auf Skalierung ausgelegte Open Source NoSQL Datenbank in Java, und wurde ursprünglich von Facebook entwickelt. Als Grundlage dienten die Datenbanken *Google Big Table* sowie *Amazon Dynamo*. Cassandra ist seit 2010 ein Apache Top-Level-Projekt.

Die Firma Datastax[1] bietet eine kommerzielle Enterprise Version von Cassandra an.

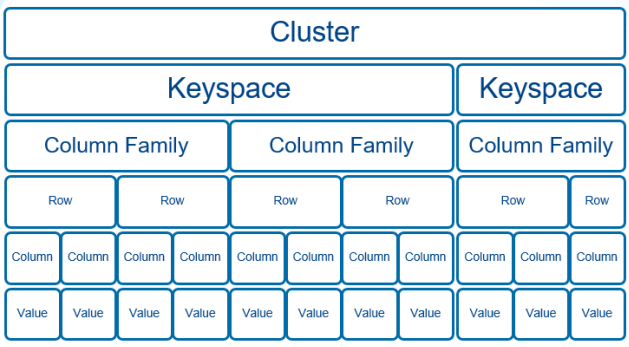
Gemäß DB-Engines.com[2] ist Cassandra die 8. meist verwendete Datenbank (total) sowie die meist verwendete Datenbank in der Kategorie *Wide Column Store*.

Für die Verwendung von Cassandra sprechen die folgenden acht Punkte:

1. High Availability Architektur
2. Lineare Skalierbarkeit
3. Spezialisiert auf das Schreiben von Daten
4. Dezentralisiert, kein *Single Point of Failure* (SPOF)
5. Replication und Fehlertoleranz
6. Multi Data Center Replication
7. Tunable Consistency
8. Hadoop Integration / MapReduce

2. Datenmodell

Das Datenmodell von Cassandra entspricht einer Containerstruktur. Der Cluster ist hierbei der Root-Container und kann als verteilter Datenbankserver betrachtet werden.



Die nächste Container-Ebene sind die **Keyspaces**, welche die Tabellen beinhalten (Tabelle = Column Family).

Jede **Tabelle** ist dabei eine sortierte Collection von Rows und jede **Row** ist eine sortierte Collection von maximal 2 Milliarden Columns. Jede **Column** ist ein Key:Value:Timestamp Objekt.

Es können auch statische Columns definiert werden, welche Row-übergreifend für die gesamte Tabelle gelten.

---

[1] www.datastax.com
[2] db-engines.com/en/ranking/ (Stand 30.06.2017)

Tabellen sind in **Partitionen** unterteilt, jede Partition kann auf einer anderen Node liegen und kann maximal 2 Milliarden Zellen (Spalten * Zeilen) besitzen.

In Cassandra werden Datensätze spaltenweise abgespeichert, jede Spalte ist dabei in einer eigenen Datei gespeichert. Neue Spalten können einfach dem Schema hinzugefügt oder entfernt werden. Die Adressierung von Attributswerten kann über die Position in der Spaltendatei oder über eine Binäre Relation erfolgen.

Vorteile dieser Speicherung ist Performance für einfache Operation auf wenigen Spalten sowie die Kompressionsmöglichkeiten (eigene Kompressionsklassen können geschrieben werden). **Nachteilig** ist die Performance für Operationen auf vielen verschiedenen Spalten.

In Cassandra besitzt jeder Datensatz einen **Primary Key**, welcher angibt wie und wo Daten gespeichert werden. Jeder Primary Key kann aus mehreren Teilen bestehen. Aufgrund der 2 Milliarden Zellen Limitierung pro Partition, können dem Primary Key neue **Partition Keys** hinzugefügt werden, welche eine Aufteilung der Partition in mehrere kleine Partitionen ermöglicht. Weiterhin können **Clustering Keys** dem Primary Key hinzugefügt werden, welche eine Sortierreihenfolge direkt im Schema definieren.

3. Datenverteilung

Das fundamentale Konzept von Cassandra ist die **Tokenrange**. Die Tokenrange ist ein 64Bit Integer und jede Cassandra Node besitzt einen Abschnitt dieser Tokenrange.

Aus dem Primary Key wird ein Token berechnet, welches die eindeutige Position in der Tokenrange wiederspiegelt. Je mehr Nodes dem Cluster hinzugefügt werden, desto geringer werden die Verantwortlichkeitsbereiche bzgl. Tokenrange pro Node.

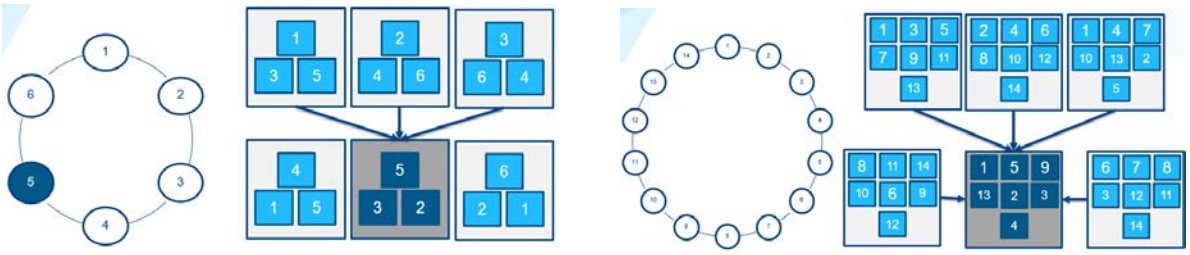
Neue Nodes können beliebig zwischen anderen Nodes eingefügt werden, im standardfall übermitteln die beiden äußeren Nodes jeweils die Hälfte ihrer Daten an die dazwischen hinzugefügte Node.

Zur Steigerung der Parallelisierbarkeit des Clusters wird jede Node in mehrere Virtuelle Nodes (VNodes) aufgeteilt. Die Tokenrange besteht daher aus vielen kleinen VNodes.

3.1 Beispiel: VNodes

Die folgende Tokenrange besitzt 6 Nodes und einen Replikationsfaktor von 3, daher hat das Cluster alle Datensätze mindestens 3x gespeichert.

Node5 ist jedoch ausgefallen. Da die Initialisierungs- und Reparaturprozesse in Cassandra auf „eine Node pro Node-ID“ beschränkt sind, kann die ausgefallene Node2-Replikation von Node5 nicht gleichzeitig von Node2 und Node6 wiederherstellt werden. Daher ist der Wiederherstellungsprozess nicht weiter parallelisierbar. Mit VNodes jedoch kann von allen reellen Nodes wiederhergestellt werden, standardmäßig hat jede Node 255 VNodes.



ohne VNodes mit VNodes

3.2 Clusterstruktur

Cassandra ordnet Nodes auf 2 Ebenen ein:

- Datencenter-Ebene (geographische Standorte)
- Rack-Ebene (Serverschränke)

Nodes im gleichen Rack zeichnet eine besondere Nähe aus (relevant z.B. für Vorhersagen bezüglich Performance und Ausfallwahrscheinlichkeit). Cassandra versucht standardmäßig, Replikationen von Daten möglichst auf Nodes in verschiedenen Racks zu verteilen.

Für jeden **Request** kommt in Cassandra eine **Snitch** zum Einsatz. Snitches sind Algorithmen welche die relative Nähe von Node bestimmen sollen (damit möglichst die Nodes ausgewählt werden, welche am schnellsten ein Ergebnis zurückliefern können).

3.3 Write

In Cassandra sind alle Nodes gleichberechtigt, jede Node kann daher Requests empfangen bzw. beantworten. Die erste Node ist dabei die Coordinator Node, welche die Schreibanfrage an die passenden Nodes weiterleitet (ggf. auch an sich selbst).

Mit Hilfe der **Tunable Consistency** kann das gewünschte Konsistenzniveau eingestellt werden (z.B. ONE, TWO, THREE…, ALL, ANY, QUORUM). Ein Schreibvorgang wird erst als erfolgreich bewertet, wenn das Konsistenzniveau erfüllt wurde.

Fällt eine Ziel-Node während des Schreibens aus oder ist zu langsam in ihrer Antwort, merkt sich die Coordinator Node mit Hilfe des **Hinted Handoffs** die ausgefallenen Replika-Nodes und schickt die Schreibanfrage erneut an die Ziel-Node, sobald diese wiederherstellt wurde. Hinted Handoffs gelten nicht als erfolgreiches schreiben, Letztendlich entscheidet jedoch der verwendete Replikationsfaktor sowie das eingestelltes Konsistenzniveau ob ein Schreibvorgang erfolgreich war oder nicht.

3.4 Read

Ähnlich den Write-Requests gelten die gleichen Voraussetzungen für Read-Requests. Die Coordinator Node fordert die eigentlichen Daten nur von einer Node an, von den restlichen Nodes wird lediglich ein Digest der Daten (=Hash) angefordert. Die Anzahl der Digest-Nodes ist abhängig vom eingestellten Konsistenzniveau (Tunable Consistency).

Sollte der Daten-Digest sowie die Digests der Digest-Node ungleich sein, wird die **Read Repair** gestartet. Die Timestamps der Daten geben Aufschluss darüber, welche Daten veraltet sind. Die Coordinator Node schickt Updateanfragen an so viele veraltete Node, bis das eingestellte Konsistenzniveau erreicht wurde.

Mit Hilfe der **Rapid Read Protection** werden extra Requests an weitere Replika-Nodes gestellt, wenn die ursprünglichen Replika-Nodes ausgefallen oder zu langsam in ihrer Antwort sind. Über den **speculative retry** kann eingestellt werden, zu welchen Auslastungsprozenten (der Replika-Nodes) oder nach welcher Zeit (in Millisekunden) die extra Requests gestellt werden sollen.

3. Cassandra Query Language (CQL)

Die CQL ist eine SQL-Ähnliche Abfragesprache für Cassandra. Es gibt in Cassandra keine JOINS und nur sehr einfache Transaktionen (beschränkt auf eine Partition). Weiterhin gibt es Views und Batches (für z.B. Sync. mehrerer Tabellen).

Ein **Index** ist auf unbegrenzt viele Spalten in Cassandra möglich. Dabei werden Indizes intern als Tabelle gespeichert. Es gibt mehrere Indexarten und es sind auch benutzerdefinierte Indizes möglich (als „JAR-Plugin“). Der SASI Index (Anhang:181) z.B. ermöglicht das Einsetzen des LIKE-Keywords zum Filtern von Tupeln.

Trigger zeigen in Cassandra auf externe JAR-Dateien, welche vom Anwender individuell gemäß Tabellenschemas als Java-Programm entwickelt werden müssen.

Die **einfachen Datentypen** in Cassandra umfassen alle gängigen Datentypen wie z.B. int, double, varchar, timestamp.

Cassandra **eigene Datentypen** sind z.B.:

- **inet**: für IPv4 oder IPv6 Adressen
- **frozen**: für Schachtelung komplexer Datentypen, intern als Blob gespeichert
- **counter:** auto-increment, kann nicht zum Primary-Key hinzugefügt werden!
- **uuid:** daraus kann das Token sowie ein Timestamp extrahiert werden
- **Set / List:** Array von Werten, Inhalt kann indiziert werden
- **Map:** Assoziatives Array, Keys und Values können beide separat indiziert werden

4. Demonstration

Ein Auszug des gezeigten CQL-Codes ist im Anhang wiederzufinden.

5. Literatur & Referenzen

- **Datastax:**
  http://docs.datastax.com
  https://www.youtube.com/user/DataStaxMedia/videos
  https://academy.datastax.com/resources/brief-introduction-apache-cassandra
- **Planet Cassandra:**
  https://www.youtube.com/user/PlanetCassandra
- **PandaForMe:**
  https://pandaforme.gitbooks.io/introduction-to-cassandra/content/
- **video2brain:**
  https://www.video2brain.com/de/videotraining/apache-cassandra-grundkurs
- **Cassandra Data Modeling and Analysis**
  C. Y. Kan, 2014, Packt Publishing
- **Cassandra Design Patterns**
  Sanjay Sharma, 2014, Packt Publishing
- **Cassandra 3.x High Availability, Second Edition**
  Robbie Strickland, 2016, Packt Publishing
- **Real-time Analytics with Storm and Cassandra**
  Shilpi Saxena, 2015, Packt Publishing

Anhang

```
-- Keyspace
describe keyspaces;
create keyspace htwk with replication =
{'class':'SimpleStrategy','replication_factor':1};
use htwk;

---- ------------------------------------------------------------
-- nur 1 primary key
drop table modules;
create table modules (ein text,fak text,sg text,name text,ects int,primary key
(ein));
describe modules;

INSERT INTO modules (ein, fak, sg, name, ects) VALUES
('HTWK','IMN','Informatik','Human Computer Interaction',6);
INSERT INTO modules (ein, fak, sg, name, ects) VALUES
('HTWK','IMN','Medieninformatik','Human Computer Interaction',5);
INSERT INTO modules (ein, fak, sg, name, ects) VALUES
('HTWK','ME','Medienmanagement','Human Computer Interaction',6);
INSERT INTO modules (ein, fak, sg, name, ects) VALUES
('UNL','MI','Informatik','Human Computer Interaction',5);
INSERT INTO modules (ein, fak, sg, name, ects) VALUES
('HTWK','IMN','Medieninformatik','Netzwerk & Systemmanagement',5);
INSERT INTO modules (ein, fak, sg, name, ects) VALUES
('HTWK','IMN','Informatik','Netzwerk & Systemmanagement',6);
INSERT INTO modules (ein, fak, sg, name, ects) VALUES
('HTWK','IMN','Informatik','Data Warehousing',6);
INSERT INTO modules (ein, fak, sg, name, ects) VALUES
('HTWK','IMN','Medieninformatik','Data Warehousing',6);
INSERT INTO modules (ein, fak, sg, name, ects) VALUES
('HTWK','ME','Medienmanagement','Data Warehousing',5);
INSERT INTO modules (ein, fak, sg, name, ects) VALUES
('UNL','MI','Informatik','Data Warehousing',10);
select * from modules;

--> nur 2 rows (die letzten beiden)
--> ein = primary key .. jeder wert kann es nur 2x geben

---- ------------------------------------------------
--compound keys
drop table modules;
describe modules;
create table modules (ein text,fak text,sg text,name text,ects int,primary key
(ein, fak, sg, name));
INSERT INTO modules (ein, fak, sg, name, ects) VALUES
('HTWK','IMN','Informatik','Human Computer Interaction',6);
INSERT INTO modules (ein, fak, sg, name, ects) VALUES
('HTWK','IMN','Medieninformatik','Human Computer Interaction',5);
```

```
INSERT INTO modules (ein, fak, sg, name, ects) VALUES
('HTWK','ME','Medienmanagement','Human Computer Interaction',6);
INSERT INTO modules (ein, fak, sg, name, ects) VALUES
('UNL','MI','Informatik','Human Computer Interaction',5);
INSERT INTO modules (ein, fak, sg, name, ects) VALUES
('HTWK','IMN','Medieninformatik','Netzwerk & Systemmanagement',5);
INSERT INTO modules (ein, fak, sg, name, ects) VALUES
('HTWK','IMN','Informatik','Netzwerk & Systemmanagement',6);
INSERT INTO modules (ein, fak, sg, name, ects) VALUES
('HTWK','IMN','Informatik','Data Warehousing',6);
INSERT INTO modules (ein, fak, sg, name, ects) VALUES
('HTWK','IMN','Medieninformatik','Data Warehousing',6);
INSERT INTO modules (ein, fak, sg, name, ects) VALUES
('HTWK','ME','Medienmanagement','Data Warehousing',5);
INSERT INTO modules (ein, fak, sg, name, ects) VALUES
('UNL','MI','Informatik','Data Warehousing',10);
select * from modules;
-- ein = partition key
-- alle anderen keys = clustering keys
-- sortiert nach ein > fak > sg > name

---- --------------------------------------------------

--performance
select * from modules where fak = 'IMN';
--> error: performance kann nicht vorhergesehen werden
select * from modules where ein = 'HTWK' and fak = 'IMN';
--> ausgabe ok
select * from modules where ein = 'HTWK' and sg = 'IMN';
--> fehler, sg kann nicht restricted sein weil keine restriktion auf vorherigen
primary key
select * from modules where ein = 'HTWK' and fak = 'IMN' and sg = 'Informatik';
--> ausgabe ok

---- --------------------------------------------------

--tokenanzeige
select token(fak), fak, token(sg), sg from modules;

--außerhalb cqlsh (in cmd)
nodetool status
nodetool ring

---- --------------------------------------------------

--ttl
create table students (id int,matnr int,name text,sg text,primary key (id));
insert into students (id, matnr, name, sg) values (1, 1000,
'AAA','Medientechnik');
insert into students (id, matnr, name, sg) values (2, 2000,
'BBB','Medientechnik');
```

```
select * from students;
update students using ttl 10 set name = 'CCC' where uuid = 2;
select * from students;

alter table students with speculative_retry = '10ms', with compression =
{ 'sstable_compression' : 'LZ4Compressor' };
describe students;

---- -------------------------------------------------

--sets -> werte unique
drop table students;
create table students (id int,matnr set<int>,name text,sg text,primary key (id));
insert into students (id, matnr, name, sg) values (1, {1000,3000},
'AAA','Medientechnik');
select * from students;
--sets append
update students set matnr = matnr + {9999} where id = 1;
select * from students;
--sets remove
update students set matnr = matnr - {3000} where id = 1;
select * from students;
--list -> werte sortiert abgelegt
alter table students add email list<text>;
select * from students;
update students set email = ['a@a.a'] where id = 1;
update students set email = email + ['b@b.b'] where id = 1;
select * from students;
update students set email[0] = 'c@c.c' where id = 1;
select * from students;
--map -> assoc
alter table students add mat map<int,text>;
update students set mat = {1000:'MIB',2000:'MTB',3000:'MIM'} where id = 1;
select * from students;
update students set mat[1000] = 'Medieninformatik (Bachelor)' where id = 1;
select * from students;
delete mat[1000] from students where id = 1;
select * from students;

---- -------------------------------------------------
--user types, frozen -> schachtelung von komplexen daten
drop table students;
create type semestercomment (semester int, comment text);
create table students(id int primary key,comments list<frozen<semestercomment>>);
insert into students (id, comments) values (1,
[{semester:1,comment:'analysis :('}] );
select * from students;
update students set comments = comments + [{semester:2,comment:'everything went
better than expected :)'}] where id=1;

---- -------------------------------------------------
```

```
--index
drop table students;
create table students (id uuid,matnr int,name text, vorname text, info
map<text,text>, primary key(id));
insert into students (id, matnr, name, vorname, info) values (uuid(), 1000,
'Ason', 'A', {'mzpin':'5555','li304pin':'4444'});
insert into students (id, matnr, name, vorname, info) values (uuid(), 2000,
'Bson', 'B', {'mzpin':'3333','li304pin':'2222'});
insert into students (id, matnr, name, vorname, info) values (uuid(), 3000,
'Cson', 'C', {'li114pin':'1234','z229pin':'4321'});
select * from students where name = 'Ason';
select * from students where name = 'Ason' allow filtering;
create index on students(name);
select * from students where name = 'Ason';
--key index
create index on students(keys(info));
select * from students where info contains key 'mzpin';
--filtering
create index on students(vorname);
select * from students where vorname like 'AA%';
--SASI
insert into students (id, matnr, name, vorname, info) values (uuid(), 4000,
'AAson', 'AA', {});
insert into students (id, matnr, name, vorname, info) values (uuid(), 5000,
'ABson', 'AB', {});
insert into students (id, matnr, name, vorname, info) values (uuid(), 6000,
'ACson', 'AC', {});
insert into students (id, matnr, name, vorname, info) values (uuid(), 7000,
'AACson', 'AAC', {});
create custom index on students(name) using
'org.apache.cassandra.index.sasi.SASIIndex';
select * from students where name like 'AA%';

---- --------------------------------------------------
--views
create table scardlogin (id timeuuid,matnr int,room text,e1 int,e2 int,primary
key(id));
insert into scardlogin (id, matnr, room, e1, e2) values (now(), 4000, 'LI-304',
10, 20);
insert into scardlogin (id, matnr, room, e1, e2) values (now(), 5000, 'LI-304',
10, 20);
insert into scardlogin (id, matnr, room, e1, e2) values (now(), 5000, 'Z-224', 10,
20);
insert into scardlogin (id, matnr, room, e1, e2) values (now(), 3000, 'LI-304',
10, 20);
insert into scardlogin (id, matnr, room, e1, e2) values (now(), 4000, 'LI-304',
10, 20);
insert into scardlogin (id, matnr, room, e1, e2) values (now(), 2000, 'LI-304',
10, 20);
insert into scardlogin (id, matnr, room, e1, e2) values (now(), 2000, 'Z-224', 10,
20);
```

```
select * from scardlogin;
create materialized view scardlv as select room, id, matnr, e1 from scardlogin
where room is not null and id is not null primary key (room, id);
select dateOf(id), matnr, e1 from scardlv where room = 'Z-224';
select sum(e1) from scardlv where room = 'Z-224';
```