HTWK

Hochschule für Technik,
Wirtschaft und Kultur Leipzig

Abstract

Apache Cassandra

Oberseminar „Datenbanken - aktuelle Trends“

Leipzig, 2. Juli 2019

Bearbeiter: Jan Oelschlegel
Betreuer: Prof. Dr.-Ing. Thomas Kudraß

Inhaltsverzeichnis

1 Einleitung

Cassandra ist eine verteilte NoSQL-Datenbank zur Verwaltung großer Datensätze in einem Server-Cluster. Das Datenbankmanagement-System lässt sich in wenigen Worten folgendermaßen beschreiben:

> *"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."*[1]

Ziel dieser Arbeit ist es, einen Überblick über diesen Vertreter der NoSQL-Datenbanken zu verschaffen. Dabei soll zunächst geklärt werden, wieso Cassandra entstand und welche Eigenschaften das System besitzt. Das zugrundeliegende Datenmodell wird ebenfalls beleuchtet. Um die Daten schließlich abrufen zu können, wird auch auf die Anfrageschnittstellen und verfügbaren Datentypen eingegangen. Zum Schluss werden noch Use-Cases für Cassandra aufgezeigt und Produktivsysteme aufgezählt, welche diese Datenbank einsetzen.

[1] vgl. J. Carpenter und E. Hewitt. *Cassandra: The Definitive Guide: Distributed Data at Web Scale.* O'Reilly Media, 2016, S.17.

2 Motivation und Entstehung

2.1 RDBMS und Skalierung

Die relationalen Datenbanken besitzen mit SQL ein ausdrucksstarke aber auch simple Sprache. Es lassen sich komplexe Beziehungen modellieren und anschließend können die Daten abgefragt, aggregiert, gefiltert oder gruppiert werden. Eine Benutzer- bzw. Rechteverwaltung ist ebenfalls vorhanden. Der SQL-Standard wird mit der Zeit weiterentwickelt und SQL wird sogar durch proprietäre Ansätze wie zum Beispiel PL/SQL (Oracle) erweitert. Wenn keine dieser Erweiterungen benutzt wird, ist SQL universell einsetzbar und das benutzte Datenbankmanagement-System lässt sich theoretisch austauschen ohne dass Änderungen an den Queries vorgenommen werden müssen. Mit Transaktionen und der ACID-Konformität lassen sich Daten sicher und konsistent speichern.

Der Verkehr im Internet wächst immer mehr an, denn spätestens durch Web 2.0 und seinen Ausprägungen im Social-Media Bereich werden immer mehr Daten übertragen und zu einem Bruchteil in Datenbanken abgespeichert. Die Performance-Ansprüche an Datenbank-Server sind deshalb hoch, da die vielen Daten in möglichst kurzer Zeit abgespeichert bzw. abgerufen werden sollen. Mit wachsender Belastung des Systems ist früher oder später eine Skalierung der Server notwendig. Da die Leistung einzelner Serverkomponenten in den letzten Jahren aber nicht mehr so stark zunimmt, ist eine vertikale Skalierung nur bedingt möglich. Es bleibt nur noch die horizontale Skalierung, die auch eine gewisse Ausfallsicherheit mit sich bringt.

Bei verteilten Datenbanksystemen ist die Einhaltung der ACID-Konformität erschwert. Es müssen verteilte Transaktionen, zum Beispiel durch einen „2-Phase-Commit“, durchgeführt werden, die Ressourcen blockieren und eine gewisse Latenz verursachen. Außerdem könnten im Server-Verbund die Daten verteilt (Sharding) abgelegt werden, was wiederum die Auswahl eines passenden Sharding-Keys erfordert, da sonst die Belastung im Cluster nicht gleich verteilt ist. Des weiteren müssen bei komplexen Joins, die ohnehin schon sehr aufwendig sind, die Daten von verschiedenen Servern gelesen werden, was die Performance des relationalen Datenbanksystems weiter verschlechtert.

2.2 Herkunft

Im Jahr 2007 hat Facebook hat neues Feature entwickelt, welches die Suche nach Wörtern im gesamten Nachrichtenverlauf aller Freunde ermöglicht. Diese Suche ist aber so aufwendig, dass diese mit dem herkömmlichen, relationalen Datenbanksystem nicht mehr bewältigt werden konnte. Das Problem wurde auch „Inbox-Search-Problem“ genannt. Fa-

cebook beauftragte Avinash Lakshman und Prashant Malik mit der Entwicklung eines passenden Datenbankmanagement-Systems. Folgende Anforderungen wurden an das System gestellt: Die Unterstützung einer hohen Schreibrate, da in jeder Sekunden viele Menschen über Facebook miteinander schreiben. Das Inbox-Search-Problem soll gelöst werden und das System sollte mit der Anzahl der Facebook-User skalieren können bzw. tolerant gegenüber einer Verteilung über Rechenzentren hinweg sein. Ein Jahr später hat das Team ein in Java geschriebenes No-SQL Datenbanksystem namens „Cassandra“ fertiggestellt und produktiv eingesetzt. Es wurde wenig später ein „Open Source Google Project“, an dem wenige Firmen, wie zum Beispiel IBM, Rackspace oder Twitter, mitentwickelt haben. Im Jahr 2009 wurde es in den „Apache Incubator“ aufgenommen und nach ca. einem Jahr wurde Cassandra ein „Apache Top Level Project“.

3 Eigenschaften

3.1 Keyfacts

Cassandra in der Major-Version 3 besitzt folgende Eigenschaften[2]:

- **Distributed:** Die Verteilung über mehrere Server ist möglich und zur Steigerung der Performance sogar erwünscht. Von außen wirkt der Verbund wie eine Einheit, selbst über Rechenzentren hinweg.
- **Decentralized:** Alle Knoten sind identisch, das heißt keiner übernimmt organisatorische Operationen (wie zum Beispiel Master-Slave Konstrukt). Einsatz eines Peer-to-Peer-Protokolls für den Datenaustausch und zur Identifikation ausgefallener Knoten.
- **Elastic Scalability:** Nahtloses Hinzufügen/Entfernen von Servern möglich. Es werden keine Prozess-Neustarts, keine Query-Änderung oder ein manuelles Rebalancing benötigt.
- **High Availability:** Nicht erreichbare Knoten werden ohne Downtime aus dem Cluster entfernt.
- **Tuneable Consistency:** Durch CAP-Theorem beschrieben können bei verteilten Datenbanksystemen immer nur zwei Eigenschaften von *Consistency*, *Availability* und *Partition Tolerance* erreicht werden. Die Hochverfügbarkeit bzw. Partitionstoleranz waren vorgaben für das System, sodass Abstriche bei der Konsistenz gemacht werden. Diese ist über die Festlegung des Replikationsfaktors bzw. des Konsistenzlevels einstellbar.

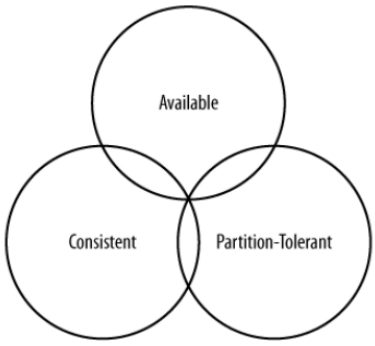

Abbildung 3.1: CAP-Theorem [3]

[2]Die Eigenschaften haben sich über die Entwicklungszeit an einigen Stellen verändert
[3]vgl. Carpenter und Hewitt, *Cassandra: The Definitive Guide: Distributed Data at Web Scale*, S. 24

- **Row-oriented:** Meist als „column-oriented“ bezeichnet, was an sich nicht falsch ist, da die kleinste Einheit von Cassandra eine *Column* ist. Die Daten werden in einer multidimensionalen Hashtabelle gespeichert. Über eine eindeutigen Schlüssel ist jede *Row* abrufbar. Eine *Row* kann eine unterschiedliche Anzahl an *Columns* besitzen. Insgesamt lässt sich Cassandra eher als einen indexierten, reihenorientierten Datenspeicher bezeichnen.

- **Schema-flexible:** Die Einführung der *Cassandra Query Language* brachte eine Schemadefinition mit sich. Diese ist aber nicht ganz starr, da Listen und Maps als Datentyp eingesetzt werden können.

- **High performance:** Cassandra ist von Grund auf für Multiprozessor-Systeme optimiert. Es sind aktuell Cluster mit hunderten Terabytes im produktiven Einsatz, welche auch unter hoher Last ihre Leistung bringen.

3.2 Unterschied zu RDBMS

Cassandra als Vertreter einer NoSQL-Datenbank unterscheided sich folgendermaßen von relationalen Datenbankmanagementsystemen (RDBMS):

- **keine Joins:** Entweder werden die Daten client-seitig gejoint oder gejointe Daten werden als neue *Table* abgelegt

- **keine referentielle Integrität:** Es existieren keine Fremdschlüssel-Beziehungen. Diese müsste man selbst durch ID's zu anderen Entitäten erstellen

- **Denormalisierung:** Relationale Datenbanken werden manchmal aus Performance-Gründen denormalisiert. Cassandra lebt von Denormalisierung, es können je nach Anwendungsfall andere *Tables* definiert werden.

- **„Query first design“:** In der relationalen Welt werden zuerst die Entitäten mit ihren Beziehungen zueinander definiert. Danach werden Queries geschrieben, um ein bestimmtes Ergebnis aus diesem Modell zu liefern. In Cassandra werden sich erst die Queries überlegt bzw. welche Daten erfragt werden müssen. Danach wird die dazu passende *Table* angelegt.

- **Sortierung in Schema festgelegt:** Bei relationalen Datenbanken lassen sich die Ergebnisse bei der Abfrage nach einer bestimmten Spalte sortieren. Bei Cassandra werden die zu sortierenden Daten bzw. die Reihenfolge durch den *Clustering Key* schon bei der Schemadefinition festgelegt.

4 Datenmodell

Die kleinste Einheit des Datenmodells von Cassandra bildet die *Column*. Sie ist eine Map von einem Namen auf einen Wert mit einem bestimmten Datentyp. Eine *Row* besteht aus mehreren *Columns* und ist über einen eindeutigen Schlüssel identifizierbar. *Rows* werden in einer *Table* (früher *Column Family*) zusammengefasst. Wenn der *Primary Key* einer *Row* aus nur einer *Column* besteht wird die *Table* als *Skinny Row* bezeichnet. Dann ist die Länge einer *Row* gleich mit der Anzahl der *Columns* einer *Row*.

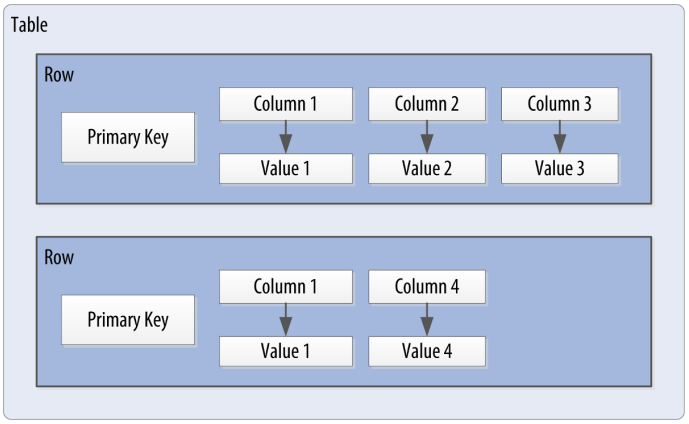

Abbildung 4.1: Skinny Row [4]

Wenn der Schlüssel einer *Row* aus mehreren *Columns* zusammengesetzt ist wird dies als *Wide Row* bezeichnet. Die erste *Column* des Schlüssels bildet den eindeutigen Schlüssel für die jeweilige *Row*. Die anderen, nachfolgenden *Columns* bilden die *Clustering Keys*. Sie bestimmen die Sortierreihenfolge und sind der Schlüssel für „Blöcke“ von *Columns*. Da jetzt unter einem *Row-Key* viele *Clustering-Keys* abgespeichert werden können, kann eine einzelne *Row* sehr breit werden.

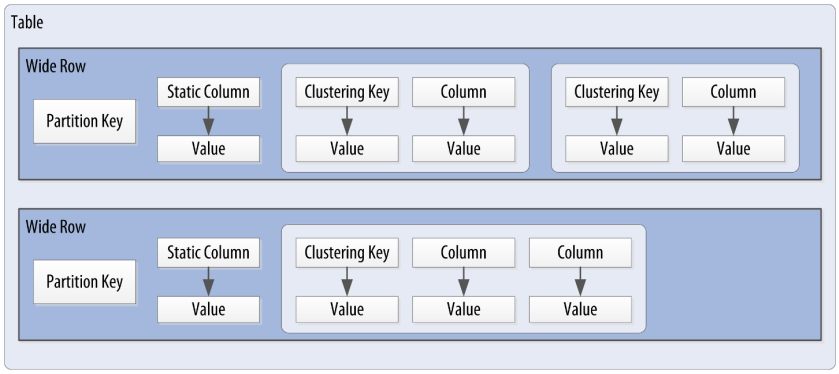

Abbildung 4.2: Wide Row [5]

[4] vgl. Carpenter und Hewitt, *Cassandra: The Definitive Guide: Distributed Data at Web Scale*, S. 60

Die *Tables* werden in einem *Keyspace* zusammengefasst, der vergleichbar mit einer *Database* in der relationalen Welt ist. In einem *Cluster* können mehrere *Keyspaces* existieren.

[5]vgl. Carpenter und Hewitt, *Cassandra: The Definitive Guide: Distributed Data at Web Scale*, S. 60

5 Anfrageschnittstellen

Als Abfragesprache dient die *Cassandra Query Language*. Mithilfe von ihr lässt sich das Schema der *Tables* definieren und es können Werte eingefügt, abgefragt und gelöscht werden. Sie ist sehr ähnlich zu SQL, wahrscheinlich auch, damit der Umstieg von einer relationalen Datenbank erleichtert wird. Es existiert auch eine Vielzahl aktueller Client-Treiber, die größtenteils durch *DataStax* weiterentwickelt werden. Unterstützte Programmiersprachen sind unter anderem Java, Python, NodeJS, Ruby, C++ und PHP. Innerhalb von Cassandra werden folgende Datentypen unterstützt:

- **numerische Datentypen:** int, bigint, smallint, tinyint. varint, float, double, decimal
- **Textdatentypen:** varchar, text
- **Zeit- und Identitätsdatentypen:** timestamp, date, time, uuid, timeuuid
- **Collection-Datentypen:** list, set, map
- **andere Datentypen:** boolean, blob, inet, counter
- **benutzerdefinierte Datentypen:** zusammengesetzte Datentypen

6 Einsatz

6.1 Use-Cases

Folgende Use-Cases sind geeignet um Cassandra als Datenbank-Lösung zu benutzen:

- **große Umgebungen:** Bereiche wo große Datenmengen anfallen bzw. abgefragt werden müssen und horizontal skaliert wird.
- **geografische Verteilung:** Erstellung von Clustern über Rechenzentren hinweg möglich.
- **viele Schreibzugriffe:** Cassandra ist auch bei hoher Schreiblast performant
- **schnell veränderliche Software**: das flexible Schema ist passend für eine veränderliche Software-Architektur

6.2 Wer benutzt Cassandra ?

Cassandra hat sich seit der Entwicklung von Facebook in der Datenbank etabliert und wird von bekannten Unternehmen für unterschiedliche Zwecke eingesetzt:

- **Apple** soll ein Cluster von ca. 100.000 Knoten besitzen. Der Anwendungszweck ist unbekannt
- **Netflix** benutzt Cassandra als Backend-Database für den Streaming-Service
- **Reddit** migrierte 2010 von MemcacheDB zu Cassandra
- **Soundcloud** benutzt Cassandra als Speicher für User-Account Daten
- **Twitter** setzt Cassandra für das Speichern von Tracking-Daten ein

7 Demo-Beispiel

Als Demo wird per Docker ein Cassandra-Cluster bestehend aus drei Knoten hochgefahren. In Einem der verfügbaren Docker-Container werden die Befehle innerhalb der *Cassandra Query Language Shell* (cqlsh) ausgeführt.

Ausgabe der Cassandra-Version

```
1 SHOW version
```

Ausgabe der verfügbaren Keyspaces

```
1 DESCRIBE keyspaces
```

Anlegen eines Keyspaces

Beim Anlegen eines *Keyspaces* kann man den Replikationsfaktor bzw. die Replikationsstrategie festlegen.

```
1 CREATE KEYSPACE htwk
2 WITH replication = {'class':'SimpleStrategy', 'replication_factor': 2};
```

Anlegen einer Table

Beim Anlegen einer *Table* müssen die Datentypen der Felder angegeben und der Primärschlüssel definiert werden. Außerdem kann ein *Clustering Key* hinzugefügt werden.

```
1 CREATE TABLE marks(
2   student_name text,
3   exam_date timestamp,
4   mark float,
5   exam_name text,
6   PRIMARY KEY (student_name,exam_name)
7
8 ) WITH CLUSTERING ORDER BY (exam_name DESC);
```

Einfügen von Daten in Table

```
1 INSERT INTO marks(student_name ,exam_date , mark ,exam_name) VALUES ('Jan
      ','2016-12-10',76 ,'Oberseminar');
2
3 INSERT INTO marks(student_name ,exam_date ,mark ,exam_name) VALUES ('Jan
      ','2016-11-11',90 ,'Computergrafik');
4
5 INSERT INTO marks(student_name ,exam_date ,mark ,exam_name) VALUES ('Jan
      ','2016-11-12',68 ,'Aufbaukurs Datenbanken');
```

Löschen von Daten aus Table

```
1 DELETE FROM marks WHERE student_name='Jan' AND exam_name='Oberseminar';
```

Hinzufügen einer Spalte zu einer bestehenden Table

```
1 ALTER TABLE marks ADD notes text;
```

Setzen der TTL einer Column

Für eine *Column* ist es möglich eine *Time-To-Live* (TTL) zu setzen. Nach Ablauf dieser Zeit wird der Eintrag bzw. die Column gelöscht.

```
1 UPDATE marks USING TTL 20
2 SET notes ='Asking some questions'
3 WHERE student_name = 'Jan'
4 AND exam_name = 'Aufbaukurs Datenbanken' ;
```

Literatur

Carpenter, J. und E. Hewitt. *Cassandra: The Definitive Guide: Distributed Data at Web Scale*. O'Reilly Media, 2016.

Hohpe, Gregor. *Starbucks Does Not Use Two-Phase Commit*. 11. Mai 2019. URL: www.enterpriseintegrationpatterns.com/ramblings/18%20starbucks.html.

IT gmbh, solid. *DB-Engines Ranking*. 9. Mai 2019. URL: db-engines.com/de/ranking.

Lakshman, Avinash und Prashant Malik. *Cassandra - A Decentralized Structured Storage System*. 8. Mai 2019. URL: www.cs.cornell.edu/projects/ladis2009/papers/lakshman-%20ladis2009.pdf.

Luber, Stefan. *Was ist NoSQL?* 10. Mai 2019. URL: www.bigdata-insider.de/was-ist-nosql-a-615718/.

Wikipedia. *Mooresches Gesetz*. 7. Mai 2019. URL: en.wikipedia.org/wiki/Moore.

– *Wirthsches Gesetz*. 7. Mai 2019. URL: de.wikipedia.org/wiki/Wirthsches%20Gesetz.