Für Daten ohne Grenzen

Apache Cassandra

Phillip Ghadir, Marc Jansing

In dieser Kolumne stellen wir Cassandra vor und zeigen, warum die Datenbank quasi beliebig skaliert und höchst performant ist. Cassandra wirkt für Kenner von relationalen Datenbanken auf den erster Blick recht vertraut, weil hier auch von Tabellen und Spalten die Rede ist und die Abfragen einfachen SQL-Abfragen ähnlich sehen. Dennoch unterscheidet sich Cassandra konzeptionell von relationalen Datenbank-Management-Systemen (RDBMS).

Bei Cassandra handelt es sich um eine spaltenorientierte Datenbank, die auf die Verteilung von Daten in großen Umgebungen zugeschnitten ist. Die Konzepte von Cassandra ähneln zwar relationalen Konzepten, unterscheiden sich aber dennoch. Tabelle 1 stellt die Cassandra-Konzepte ihren relationalen Pendants gegenüber.

Relationales DBMS	Cassandra
Datenbankinstanz	Cluster
Datenbank	Keyspace
Tabelle	Tabelle / Column Family
Zeile	Zeile
Spalte (für alle Zeilen gleich)	Spalte (kann je Zeile unterschiedlich sein)

Tabelle 1: Gegenüberstellung der Datenbankkonzepte von RDBMS und Cassandra

Eine Spalte ist die kleinste logische Einheit innerhalb von Cassandra. Jede Spalte besteht aus Name, Wert und einem Zeitstempel. Spalten mit ähnlichem Inhalt werden in Tabellen (auch *Column Families* genannt) gruppiert. Jede Zeile einer Tabelle ist eindeutig identifizierbar und kann auch Spalten enthalten, die aus einer Liste von weiteren Spalten besteht und *Super Columns* genannt werden.

Cassandra unterstützt Schemamigrationen, ohne die generelle Struktur einer Column Family zu verändern. Im Unterschied zur relationalen Welt muss eine Zeile nicht aus den gleichen Columns bestehen.

Cassandra legt einem Anwendungsentwickler eine Menge an Beschränkungen auf, die vielleicht auf den ersten Blick ungewohnt sind. Die Beschränkungen ermöglichen letztlich eine skalierbare und effiziente Datenhaltung.

Wer zusammengehörige Daten bereits beim Speichern gruppieren oder aggregieren kann, erhält eine Datenbasis, die an sich in wahnsinnige Dimensionen wachsen und dennoch effizient abgefragt werden kann.

Den Cluster im Blut

Cassandra ist darauf ausgelegt, im Cluster zu laufen. Solch ein Cluster kann dabei einfach über Datacenter, Racks und Hosts hinweg aufgespannt werden. Beim Hinzufügen zum oder Entfernen von Cassandra-Instanzen aus dem Cluster sorgt Cassandra automatisch für eine angemessene Replikation und Verteilung der Daten (siehe Kasten „Intelligent verteilen“).

Für den Anwendungsentwickler ist es unerheblich, ob die Datenbank als Cluster oder einzelne Instanz realisiert ist. Die Abfragen stellt man stets an eine Instanz, die am besten im eigenen Datacenter liegt. Diese Instanz beantwortet alle Anfragen des Clients entweder direkt oder indirekt, indem die Antwort von einer anderen Instanz beschafft wird.

Die Daten werden je nach Strategie von dem am besten passenden Server geliefert beziehungsweise abhängig von den Replikationseinstellungen auf den oder die nächsten Server gespeichert.

Der Primärschlüssel

Cassandra verwendet Primärschlüssel zum Verwalten der Ablageorte von Daten. Aufgrund der möglichen Replikation gibt es potenziell mehrere. Der Primärschlüssel einer Tabelle kann entweder einfach oder aber aus mehreren Spalten zusammengesetzt sein.

Bei einem zusammengesetzten Primärschlüssel nutzt Cassandra die erste Spalte stets als sogenannten Partitionierungsschlüssel (*partition key*), über den die Daten redundant auf einzelne Cluster-Instanzen verteilt werden. Die zweite Spalte des Primärschlüssels ist der sogenannte *Clustering-Key*, mit dem Cassandra Daten vorgruppiert, um diese effektiv mit einer Anfrage zu ermitteln.

Alle weiteren Spalten des Primärschlüssels können ebenfalls zum Clustern und Selektieren von Teilmengen genutzt werden.

Abfragesprache CQL

Cassandra bietet eine an SQL angelehnte Abfragesprache: die Cassandra Query Language (CQL). CQL ist heute das Standard-Interface. CQL und SQL teilen sich das Modell von Tabellen, welche sich aus Spalten und Zeilen zusammensetzen. Mit CQL können Datenstrukturen angelegt (`CREATE TABLE`), abgefragt (`SELECT`) oder Daten manipuliert werden (`INSERT`, `UPDATE`, `DELETE`). CQL unterstützt keine JOIN-, Sub-Query- oder GROUP-BY-Operationen.

Die Einschränkung der Ergebnismenge mittels `WHERE` setzt stets einen Index voraus und kann nicht mit OR verknüpft werden. Der Zugriff mit CQL erfolgt über die von Cassandra mitgelieferte CQL-Shell cqlsh.

Für den programmatischen Zugriff stehen Treiber [Drivers] für diverse Programmiersprachen bereit. Die Treiber unterstützen CQL-Abfragen, Prepared Statements sowie die automatische Paginierung der Ergebnismenge.

Außerdem existieren Client-Bibliotheken mit einer höheren Abstraktionsschicht wie zum Beispiel [Hector]. Hector bietet

Intelligent verteilen

Cassandra ist per Design auf Hochverfügbarkeit ausgelegt. Mehrere Cassandra-Instanzen (Nodes) werden zu einem Ring zusammengeschlossen, bei dem jede Instanz jede Anfrage bearbeiten kann. Ein Cluster kann auch über Ortsgrenzen hinweg aufgespannt werden. Die Nodes innerhalb eines Datacenters sollten jedoch örtlich nah beieinander liegen, um die Latenz bei Abfragen möglichst gering zu halten. Es existiert keine kritische Komponente (Single Point of Failure) innerhalb eines Clusters, da jeder Node gleichwertig behandelt wird.

Um eine hohe Zuverlässigkeit und Fehlertoleranz zu gewährleisten, werden Kopien der Daten (Repliken) auf anderen Nodes abgelegt. Sollte ein Node, ob temporär oder dauerhaft aufgrund eines Defekts aus dem Cluster ausscheiden, so ist die Verfügbarkeit des Clusters dank der auf den anderen Nodes abgelegten Repliken nicht gefährdet. Die Anzahl der Repliken ist konfigurierbar und wird allgemein als Replikationsfaktor (N) bezeichnet. Ein Replikationsfaktor von N=3 besagt demnach, dass insgesamt drei Kopien der Daten innerhalb eines Clusters zur Verfügung stehen.

Die Art und Weise, wie diese Kopien verteilt werden, wird über die Replikationsstrategie je Keyspace festgelegt. Cassandra bietet hierfür zwei Strategien: die SimpleStrategy und die NetworkTopologyStrategy.

Die *SimpleStrategy* ist für den Betrieb eines einzelnen Datacenters optimiert. Hierbei wird die erste Kopie auf Grundlage des Partitionierungsschlüssels auf einem anderen Node repliziert. Alle weiteren Repliken werden anschließend im Uhrzeigersinn entlang des Rings verteilt.

Die *NetworkTopologyStrategy* hingegen berücksichtigt die zugrunde liegende Netzwerktopologie. Hierbei werden die Repliken auf verschiedenen Racks verteilt, da diese aufgrund ihrer geografischen Nähe durch Strom- oder Netzwerkprobleme besonders gefährdet sind und oft gemeinsam ausfallen. Diese Strategie wird für die meisten Installationen empfohlen und erleichtert die zukünftige Erweiterung des Clusters.

Die Bestimmung des Replikationsfaktors und der jeweiligen Strategie zur Verteilung führt unweigerlich zu Fragen bezüglich der Datenkonsistenz. Beim Einsatz von Cassandra ist es üblich, Verfügbarkeit und Partitionstoleranz höher als die Datenkonsistenz zu priorisieren. Schreibvorgänge erfolgen in Cassandra zeitnah (aber potenziell zeitverzögert) sichtbar (eventual consistency).

Die Konsistenz kann für Lese- und Schreibvorgänge separat sowohl global als auch clientseitig pro Operation konfiguriert werden, was sich dann wiederum auf die Verfügbarkeit auswirkt. Die gängigsten Konsistenzniveaus zeigt die folgende Tabelle.

Konsistenzniveau	Bedeutung
ONE, TWO, THREE	1, 2 bzw. 3 Nodes müssen Lese-/Schreiboperation bestätigen
QUORUM	Mehrheit (>51%) der Nodes muss Operation bestätigen
LOCAL_QUORUM	Mehrheit (>51%) der Nodes im DC muss Operation bestätigen
ALL	alle Nodes müssen Operation bestätigen, volle Konsistenz

In einem 3-Node-Cluster mit QUORUM-Konsistenzniveau für Lese-/Schreiboperationen müssen jeweils zwei Nodes eine Operation bestätigen. Bei Ausfall eines Nodes verbleiben weiterhin zwei Nodes, um die Verfügbarkeit des Clusters aufrecht zu erhalten. Ein Konsistenzniveau von ALL würde zwar volle Datenkonsistenz gewährleisten, dafür jedoch die Fehlertoleranz des Clustern verringern. Bei Ausfall eines Nodes stehen nicht mehr alle Nodes zur Bestätigung der Operation zur Verfügung. Der Cluster verliert damit seine Lese- und Schreibfähigkeit. Bei unterschiedlichen konfigurierten Konsistenzniveaus für Lese- und Schreiboperationen kann der Cluster auch nur eine dieser beiden Eigenschaften verlieren.

ein objektorientiertes Interface, clientseitige Fehlerbehandlung, Connection Pooling und weitere Features.

Korrespondenzen finden

Eine Tabelle kann man in Cassandra so ähnlich anlegen wie in einer relationalen Datenbank. Wollen wir für ein System alle Korrespondenzen finden, die von einer Person zu einem Vorgang angefallen sind, könnten wir in der Cassandra-Shell (cqlsh) eine Tabelle so anlegen:

```
CREATE TABLE CORRESPONDENCE (
  customer_no int,
  policy_id int,
  time timestamp,
  message_id int,
  direction varchar,
  subject text,
  PRIMARY KEY (customer_no, time, policy_id, message_id)
) WITH CLUSTERING ORDER BY ( time desc );
```

Dabei wird eine Tabelle definiert, die neben den Spalten zur Selektion und Identifikation der Korrespondenz den Betreff und die Nachricht enthält. In dem Szenario hier gehen wir davon aus, dass ein Kunde nicht unzählige Nachrichten schickt, sondern eher sporadisch, wodurch die Granularität von Timestamps ausreichen sollte, um die Kommunikation auf sämtlichen Kommunikationskanälen eindeutig zu erkennen.

Auch wenn die Tabellendefinition ein wenig wie SQL aussieht, fallen ein paar kleine Unterschiede auf: Für `varchar` benötigt Cassandra keine Angabe, wie breit eine Spalte werden kann. Darüber hinaus ist die Angabe der Clustering-Strategie erwähnenswert.

Aufgrund der Annahme, dass ein Kunde sich auf Anliegen bezieht, die irgendwann abgeschlossen sind, wird direkt bei dem Anlegen der Tabelle `CORRESPONDENCE` eine Clustering-Strategie für die Sortierung nach Zeit der Korrespondenz (absteigend) angegeben. Damit werden bei einer Abfrage die jüngsten Einträge zuerst zurückgeliefert.

Abfragen brauchen Index

Kennen Sie erfahrene Entwickler, Architekten oder Admins, die bei der Einführung oder bei produktionsnahen Tests darauf achten, dass nirgends aus Versehen Full-Table-Scans ver-

ursacht werden? Bei einer relationalen Datenbank kann ein Full-Table-Scan potenziell verheerend für die Antwortzeit sein. Wenn beispielsweise eine Abfrage zwei Tabellen miteinander verknüpft und dabei für jede Zeile der ersten Tabelle einmal über jede Zeile der zweiten Tabelle iteriert, kann das dauern.

Mit Cassandra kann so etwas nicht passieren. Es können nur unverknüpfte Tabellen abgefragt werden. Soll die Treffermenge eingeschränkt und so nur bestimmte Daten selektiert werden, funktioniert dies nur über Spalten, auf denen ein Index definiert wurde. Ohne Index auf den entsprechenden Feldern gibt es keine Ergebnismenge, sondern eine Fehlermeldung.

Kunde ruft an

Bei einem Anruf nun die entsprechende Korrespondenz zu ermitteln, ist mit obiger Tabelle **CORRESPONDENCE** leicht. Ruft der Kunde an, wird mit seiner Kundennummer nach der Korrespondenz gesucht:

```
SELECT *
  FROM CORRESPONDENCE
WHERE customer_no = 4711;
```

Seine Daten werden sortiert von ganz neu bis uralt zurückgeliefert, weil die Tabelle mit dem Ordnungskriterium angelegt wurde. Der Mitarbeiter am Helpdesk kann sich somit leicht einen Überblick verschaffen.

Verwendung in der Java-Anwendung

Es gibt verschiedene Client-Bibliotheken für verschiedene Programmiersprachen. Im Folgenden verwenden wir die Cassandra-Client-Bibliothek von DataStax. Im Build-File fügen wir die Dependency zu **com.datastax.cassandra:cassandra-driver-core:2.1.4** hinzu.

Die folgenden Code-Fragmente dienen dazu, eine Client-Session für eine Cassandra-Instanz (optional innerhalb eines Clusters) zu initialisieren. Für die lokale Instanz muss der String-Parameter node mit "127.0.0.1" initialisiert sein:

```
// irgendwo importieren wir den folgenden Package-Pfad
import com.datastax.driver.core.*;

// dann können wir den Cluster initialisieren
// und uns mit dem Keyspace "demoKeyspace" verbinden.
Cluster cluster;
Session session;

cluster = Cluster.builder()
  .addContactPoint(node)
  .build();
session = cluster.connect( "demoKeyspace" );
```

Sowohl die Session als auch die Repräsentation des Clusters auf der Client-Seite müssen am Ende mit **close()** geschlossen werden. Wir haben diese Initialisierung daher in einer eigenen Klasse gekapselt, die **AutoCloseable** implementiert. Dort haben wir auch eine Methode **showCorrespondence** mit der oben genannten Abfrage implementiert:

```
public void showCorrespondence(String customerNo) {
  final ResultSet resultSet = session.execute(
    "SELECT * FROM CORRESPONDENCE where CUSTOMER_NO = ?;",
    customerNo);

  for ( Row row : resultSet ) {
    System.out.println( row );
  }
}
```

Vorbereitung ist alles

Beim Einsatz von Cassandra ist es üblich, Daten bereits beim Speichern aufzubereiten, damit sie danach effektiv gelesen werden können. Das Zählen von Werten in einer Datenbanktabelle wie mit der folgenden Abfrage ist in Cassandra unüblich:

```
SELECT count(*)
  FROM CORRESPONDENCE
WHERE CUSTOMER_NO = 4711;
```

Cassandra-typisch ist eher, einen entsprechenden Counter anzulegen und den bei jedem Speichern und Löschen von neuen Daten zu aktualisieren. Für die Korrespondenzen können wir einen Counter folgendermaßen anlegen:

```
CREATE TABLE CORRESPONDENCE_PER_CUSTOMER
  (counter_value counter,
  customer_no varchar,
  PRIMARY KEY (customer_no)
);
```

Einen solchen Counter müssen wir in der Anwendung selbst pflegen. Beispielsweise könnten wir in einer Methode zum Anlegen eines neuen Kontaktes (einer neuen Korrespondenz) zusätzlich den Counter erhöhen. Das Erhöhen eines Counters erfolgt per CQL in einem Update-Statement so:

```
UPDATE CORRESPONDENCE_PER_CUSTOMER
  SET counter_value = counter_value + 1
WHERE customer_no = 4711;
```

Das Vermindern eines Counters funktioniert analog, indem man den Counter herunterzählt.

In unserer Implementierung speichern wir die Korrespondenz und erhöhen den Counter in einem Rutsch. Dazu führen wir in der Methode **createIncomingCall()** zwei Anweisungen aus. Sollte dabei etwas schief gehen, kehrt **session.execute()** mit einer Exception zurück. Das **ResultSet** ist bei Anweisungen, die keine Abfragen sind, stets leer:

```
public void createIncomingCall(String customerNo) {
  session.execute(
    "INSERT INTO CORRESPONDENCE " +
     "(customer_no, time, policy_id, message_id, direction, subject) "
     + "VALUES (?,now(),?,?,?,?);",
    createMetaDataForIncomingCall(customerNo));

  session.execute(
    "UPDATE CORRESPONDENCE_PER_CUSTOMER " +
      " SET counter_value = counter_value + 1 " +
      " WHERE customer_no = ?;",
    customerNo);
}
```

Ein so stets aktuell gehaltener Zähler lässt sich effizienter abfragen, als über eine Menge von Daten zu iterieren und die Anzahl zusammenzuzählen.

Collection-Felder für Daten mit mehreren Details

Wenn wir Daten mit zugehörigen Details speichern und laden wollen, bietet Cassandra dafür Collection-Felder. Tabellen können Collection-Felder vom Typ Liste, Set oder auch Map enthalten, die sich genauso verhalten, wie man es erwarten würde.

Bei einer 1-zu-n-Beziehung mit n im kleinen zweistelligen Bereich ist eine Collection sicherlich in Ordnung. Wer die aber für sehr große n verwenden will, verstößt definitiv gegen die ursprüngliche Design-Idee. Das wird mit langsameren Antwortzeiten und irgendwann auch mit einer Exception quittiert.

Cassandra-Werkzeuge

Zusätzlich zu den mitgelieferten Tools existieren Werkzeuge, welche die Arbeit mit Cassandra weiter erleichtern. Hierzu zählen die kostenfrei erhältlichen Werkzeuge [OpsCenter] und [DevCenter].

DataStax OpsCenter ist eine Webanwendung zur Verwaltung von Cassandra-Clustern. Es unterstützt bei der Administration eines Clusters, bietet Realtime-Monitoring und erstellt per Knopfdruck Berichte und Metriken über den Cluster. Für die Integration in andere Anwendungen stellt es außerdem eine Programmierschnittstelle bereit.

Das auf Eclipse RCP basierende *DevCenter* hingegen richtet sich an den Anwendungsentwickler. Hierbei handelt es sich um ein visuelles Abfragewerkzeug, das diverse Verbindungen verwalten kann und bei der Formulierung sowie der Analyse von CQL-Befehlen unterstützt.

Die *DataStax Sandbox* [Sandbox] bietet für Interessierte einen guten Einstieg. Hierbei handelt es sich um eine Cassandra-Distribution inklusive der vorgestellten Werkzeuge, die in Form einer virtuellen Maschine für VMWare oder VirtualBox bereitgestellt wird.

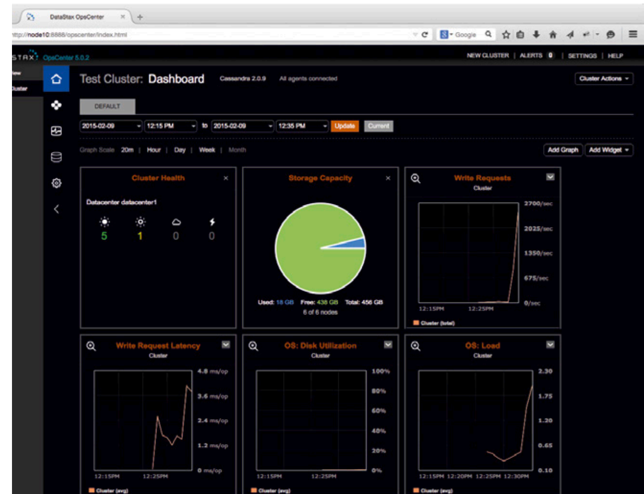

Abb. 1: OpsCenter Dashboard eines 6-Node Clusters

Daten nach Ablauf vergessen (TTL)

Ein weiteres bemerkenswertes Feature von Cassandra ist, dass es für einzelne Daten einen Gültigkeitszeitraum (Time-To-Live, TTL) speichern kann. Dazu muss man beim Einfügen eines Datensatzes angeben, wie viele Sekunden die Daten gespeichert werden sollen.

Dazu wird an die Insert-Anweisung zusätzlich eine "Using TTL"-Klausel angefügt. Hier speichern wir beispielsweise die Angebotsaufforderung von Interessenten für ein Jahr:

```
INSERT INTO request_for_quote (
  name, address, date, content
) VALUES (?,?,?,?)
  USING TTL 31536000;
```

Das funktioniert genau so für einzelne Werte in einer Collection-Spalte oder auch für einen Eintrag in einer Map.

Fazit

In der Vergangenheit sind hier bereits verschiedene NoSQL-Datenbanken vorgestellt worden. Mit dem spaltenorientierten Datenbank-Management-Systemen Cassandra haben wir einen weiteren Vertreter dieser Art vorgestellt, der sich durch hohe Performance auszeichnet und für den Betrieb im Cluster (auch über Rechenzentrumsgrenzen hinweg) geschaffen ist.

Die Daten werden dabei im Cluster nach anpassbaren Strategien über Instanzen hinweg verteilt und geeignet gruppiert. Der Betrieb eines Cassandra-Clusters erfordert weder besondere Administrationsfähigkeiten noch spezielle Aufmerksamkeit. Cassandra-Instanzen können einfach zum Cluster hinzugefügt oder aus ihm entfernt werden. Dabei werden die Daten effizient neu verteilt.

Ein wesentlicher Teil von Cassandras Effizienz liegt in deren Architektur und der ausgefeilten Implementierung. Dennoch erlaubt erst das Befolgen von typischen Empfehlungen für die Implementierung effizienter Datenmodelle und Abfragen, wirklich performante und skalierbare Systeme zu bauen.

Daher hilft das Beschäftigen mit Cassandra auch jedem, der Datenbanken einsetzt, sich noch einmal die Tugenden effizienter Datenbankabfragen vor Augen zu führen: lokale Abfragen mit möglichst geringem Datenzugriff und Einhaltung der Reihenfolge, in der das Datenbank-Management-Systemen die Daten eh verwaltet.

Links

[CassandraScalability] A. Cockcroft, D. Sheahan, Benchmarking Cassandra Scalability on AWS – Over a million wites per second, 2.11.2011, `http://techblog.netflix.com/2011/11/benchmarking-cassandra-scalability-on.html`

[DevCenter] `http://www.datastax.com/what-we-offer/products-services/devcenter`

[Drivers] `http://www.datastax.com/download-drivers`

[Hector] `http://hector-client.github.io/hector/build/html/index.html`

[OpsCenter] `http://www.datastax.com/what-we-offer/products-services/datastax-opscenter`

[Sandbox] `http://www.datastax.com/what-we-offer/products-services/sandbox`

Phillip Ghadir baut am liebsten tragfähige, langlebige Softwaresysteme. Er ist Mitglied der Geschäftsleitung bei innoQ und hat sich früh auf Architekturen für verteilte, unternehmenskritische Systeme spezialisiert. Darüber hinaus ist er Mitbegründer und aktives Mitglied des iSAQB, des International Software Architecture Qualification Board.
E-Mail: phillip.ghadir@innoq.com

Marc Jansing arbeitet als Consultant bei innoQ und entwickelt seit mehreren Jahren Webanwendungen mit leichtgewichtigen Frameworks. Sein Schwerpunkt liegt in der Implementierung von verteilten Systemen und ergonomischen Anwendungen auf REST-Basis.
E-Mail: marc.jansing@innoq.com