SCHULD IST (NICHT) CASSANDRA!



Kein Kassandraruf kann die wahren Helden schrecken, auch wenn sie wissen, sie wird Recht behalten.[1]

| von LASZLO LÜCK

Cassandra[2], als Vertreter moderner, verteilter Datenbank-Architekturen, findet in immer mehr Projekten Beachtung. Ein Grund ist, dass aktuelle IT-Architekturen zunehmend auf hochskalierbare und verteilte Architekturen aufbauen, da die zu verarbeitenden Datenmengen immer größer werden. Typische Anwendungsfälle finden sich in polizeilichen Vorgangsbearbeitungssystemen, Quasi-Echtzeit-Systemen zur Personenkontrolle und vor allem in großen Registeranwendungen, bei denen Cassandra als Big-Data-Speicher zum Einsatz kommt.

Cassandra wurde als ein Vertreter einer neuen Generation moderner Datenbank-Architekturen auf Konferenzen und in der Fachpresse vielfach beachtet. Dies führt dazu, dass bei der Projektplanung versucht wird, die Datenschicht „optimaler" gestalten zu können als mit traditionellen Datenbanken wie Oracle oder MySQL. Diesem anfänglichen Enthusiasmus folgte jedoch oftmals die Ernüchterung – verbunden mit dem Wunsch, doch wieder auf bewährte Technologien in Form von relationalen Datenbanken setzen zu können. Und das, obwohl die Ursachen für diese Ernüchterung in den meisten Fällen gar nicht bei Cassandra lagen, sondern daran, das auf Cassandra basierende System mit dem Wissen und der Erfahrung aus klassischen SQL-Datenbanksystemen betriebstechnisch zu unterhalten und softwareseitig zu benutzen. Der vorliegende Artikel zeigt die bisher gewonnenen Erfahrungen aus etlichen Projekten und reflektiert sie kritisch.

WOHER KOMMT CASSANDRA?

Cassandra ist ein Gemeinschaftsprojekt zweier Facebook-Mitarbeiter, die sich davon herausgefordert sahen, Informationen über eine sehr große Menge von Facebook-Nachrichten zu speichern, während die Nutzer gerade mit ihren Freunden im Facebook-Netzwerk kommunizierten.[3] Die zu erwartende Speichermenge und die vorgegebenen Randbedingungen des Betriebs seitens Facebook machten es erforderlich, ein völlig neues Datenspeicherkonzept zu entwickeln. Es sollte leicht skalierbar, aber kostengünstig sein. Das Ziel war, eine generische Datenbanklösung zu entwickeln, die für viele Anwendungszwecke verwendet werden kann.

Nach der Freigabe des Source Codes im Jahr 2008 steuerten weitere Unternehmen (IBM, Twitter, Netflix und viele mehr) Codes zu Cassandra bei. Im Jahr 2009 wurde das Projekt bei der Apache Software Foundation als Unterprojekt in den Apache Incubator aufgenommen. Im Februar 2010 wurde Cassandra von der Apache Software Foundation zum einem der sogenannten Top-Level-Projekte erklärt.

WAS IST CASSANDRA?

Cassandra ist ein extrem schnelles, skalierbares und ausfallsicheres Datenbanksystem, das sich hervorragend für die Speicherung sehr großer Datenmengen (1.8446744[19] Datensätze) eignet.

Ein Praxisbeispiel für die Resilienz und Ausfallsicherheit von Cassandra ist ein Vorfall aus dem Jahr 2014 bei der Firma Netflix: Durch das Einspielen eines Notfall-Security-Patches im Amazon-AWS-Stack mussten 218 der insgesamt über 2.700 Cassandra Nodes nahezu gleichzeitig neu gestartet werden. Nach der Wiederinbetriebnahme des betroffenen AWS-Netzes wurden alle Cassandra Nodes ohne Probleme wieder in die bestehende Landschaft integriert, ohne dass ein Kunde etwas vom Ausfall bemerkte.[4] Cassandra zeichnet sich außerdem als ein sehr schnelles Datenspeichersystem aus, sowohl beim Schreiben als auch Lesen von Daten.[5]

Sollten sich Engpässe beim Speicherplatz oder der Geschwindigkeit bemerkbar machen, ist es für einen IT-Betrieb mit entsprechendem Fachwissen sehr einfach möglich, weitere Cassandra Nodes dem Gesamtverbund (Node, Cluster – siehe Glossar) hinzuzufügen. Um die Neuverteilung der bereits im Cluster befindlichen Daten kümmert sich Cassandra automatisch. Sollten manuelle Eingriffe notwendig sein, bietet Cassandra hier Tools und Mechanismen, um Optimierungen zu ermöglichen. Dazu zählt beispielsweise die manuelle Verteilung der Daten oder Token auf den jeweiligen Nodes, wenn diese nicht mit dem gleichen Datenspeicherplatz ausgestattet sind.

Die Grenzen sind hier nur die Mauern des Rechenzentrums, in dem der Verbund betrieben wird. Aber selbst wenn das eine Rechenzentrum zu klein würde: Über Geo-Replikation ist Cassandra geradezu für den verteilten Einsatz über mehrere Standorte prädestiniert. Diese Eigenschaften machen Cassandra zu einem sehr ausfallsicheren Datenbanksystem.

Die Einfachheit beim Betrieb und die leichte Skalierbarkeit eines Cassandra-Clusters auf der einen Seite setzen fundierte Kenntnisse und Wissen als Softwarearchitekt und -entwickler auf der anderen Seite voraus. Der Ansatz des DevOps[6] (Developer und Operator) gewinnt beim Einsatz von Cassandra sehr große Bedeutung, denn nur mit dem technischen Wissen über Aufbau und Funktionsweise dieses Datenspeichers lässt sich Cassandra aus Sicht des Entwicklers korrekt nutzen.

WAS IST CASSANDRA NICHT?

Cassandra ist kein klassisches relationales Datenbanksystem (RDBMS). Weder erlaubt es die Speicherung und Ausführung von Geschäftslogik innerhalb der Datenbank (Stored-Procedures, Trigger) noch bietet es Relationen (Verbindungen/Beziehungen) zwischen Tabellen an. Außerdem ist Cassandra ein schemaloses Datenbanksystem (mittels eines Schemas legt man in einem klassischen RDBMS die Speichertechnik oder das Datenbankdesign fest).

Die fachliche Einordnung von Daten in Cassandra erfolgt mittels Keyspaces. Verfolgt man für die Modellierung der Software beispielsweise den Ansatz des Domain-Driven-Designs[7], kann man diesen bis in die Datenbank konsequent fortsetzen, da ein Keyspace für die Datenhaltung einer fachlichen Domäne verwendet wird. Technisch definiert man in einem Keyspace den Replikationsfaktor für die enthaltenen Daten.

Weniger gut ist Cassandra geeignet, wenn Datensätze häufig verändert werden. Ebenso erzeugt das häufige Löschen von Datensätzen eine höhere Last im Cassandra-Cluster, als man es von klassischen SQL-Servern gewohnt ist. Der Grund: Ein gelöschter Datensatz wird in Cassandra immer erst „zur Löschung markiert" und erst zu einem späteren Zeitpunkt aus dem Cluster physikalisch entfernt.

Ebenfalls eine Herausforderung für Cassandra ist die Implementierung zum Erzeugen und zur Persistenz von Nummernsequenzen. Diese benötigt man zum Beispiel, um applikationsweit eineindeutige Nummern/IDs zu generieren. Ist dies in einem RDBMS ein Standard, lässt sich diese Funktionalität mit Cassandra aufgrund der Verteilung der Daten (Partitionstoleranz) und des Konsistenzlevels (Eventual Consistency) nur sehr schwer umsetzen beziehungsweise bei hohem Konsistenzlevel gar nicht abbilden. Für diese Art von Operationen sind klassische SQL-Datenbanken klar im Vorteil.

WIE FUNKTIONIERT CASSANDRA?

Apache Cassandra legt die Daten in sogenannten Key Value Stores (Schlüssel-Wert-Speicher) ab. Der Schlüssel wird einmalig beim Erstellen einer Tabelle angelegt und kann nicht mehr ge-

ändert werden. Mit diesem Schlüssel ist definiert, wie Cassandra die Daten innerhalb des Clusters beim Schreiben verteilt (Partitionierung) und beim Lesen über den Schlüssel wieder auffinden kann. Der Primary Key besteht aus zwei Teilen:

1. Dem Partition Key, der festlegt, auf welcher Node die Zeile im Cassandra-Verbund geschrieben wird,
2. Dem Cluster Key, über dessen Attribute weitere Einschränkungen (beispielsweise Von-bis-Abfragen) durchgeführt werden können.

Der Partition Key muss beim Lesen von Daten immer komplett angegeben werden, damit die entsprechenden Datensätze innerhalb des Clusters gefunden werden kann (partitionsweise Selektion).

Beide Schlüsselteile (Partitition- und Cluster Key) können sich dabei jedoch aus mehreren Attributen (Feldern) einer Tabelle zusammensetzen (Compound Key). Ein selektierbares Attribut kann dabei entweder nur Teil des Partition Keys oder nur Teil des Cluster Keys sein. Gibt es mehrere Partition Key-Bestandteile, werden diese durch eine Klammer verbunden.

Beispiel:
PRIMARY KEY((column1, column2), column3, column4)
Beim Lesen von Datensätzen sind ausschließlich folgende Kombinationen möglich:

column1 (Partition Key)	column2 (Partition Key)	column3 (Cluster Key)	column4 (Cluster Key)
■	■		
■	■	■	
■	■	■	■

Das Weglassen aller Schlüssel kommt einer Selektion über alle Datensätze gleich. Das sorgt jedoch auf der Client-Seite für eine unter Umständen extrem hohe Speicherauslastung bei Empfang der Datensätze und auf der Server-Seite (beim Coordinator) ebenfalls für eine sehr hohe Speicherauslastung und eine hohe CPU-Last. Bei Tabellen mit vielen Einträgen wird diese Abfrage in den meisten Fällen zu einem Time-out und damit zu einem Fehler führen.

Während sich die selektierbaren Kriterien in jeweils eigenen Spalten befinden müssen, wird der abzurufende Datensatz (Value) in eine einzige Spalte geschrieben. Dadurch ist eine extrem hohe Geschwindigkeit beim Schreiben und Auffinden von Datensätzen im Cluster gewährleistet, jedoch muss der Schlüssel für das Auffinden von Datensätzen immer exakt angegeben werden.

Beim Lesen und Schreiben von Daten kann jeweils festgelegt werden, ob für den jeweiligen Vorgang eher die Konsistenz der Daten im Cluster oder die Verfügbarkeit im Vordergrund steht. Für diesen Zweck gibt es in Cassandra die Angabe des Consistency-Levels.[8]

CONSISTENCY LEVEL[9]

The Cassandra consistency level is defined as the minimum number of Cassandra nodes that must acknowledge a read or write operation before the operation can be considered successful.

Der Konsistenz-Level von Cassandra definiert, wie hoch die Mindestanzahl von Cassandra Nodes ist, die einen Lese- bzw. Schreibvorgang bestätigen müssen, damit dieser Vorgang als erfolgreich durchgeführt gilt.

Die technische Einfachheit von Cassandra schränkt die Benutzbarkeit für Entwickler mit der Erfahrung klassischer Datenbanksysteme erst einmal ein. Einerseits muss man die Verteilung der Daten innerhalb des Clusters im Auge behalten (Wahl des Partition Keys), andererseits muss das Tabellen- und Schlüsseldesign so gewählt sein, dass die fachlichen Anforderungen der Anwendung umgesetzt werden können. Bedenkt man, dass mit Cassandra exakte Abfragen ausschließlich auf dem oder den Schlüssel(n) möglich sind, ist es notwendig, hier neue Wege zu gehen.

Die Selektion von Datensätzen über mehrere Datenfelder erreicht man unter anderem mit:

1. Metatabellen
Mittels Metatabellen kann man die Selektionskriterien so gestalten, dass die fachlichen Anforderungen erreicht werden können. Metatabellen funktionieren in diesem Falle wie ein Zeiger auf den Hauptdatensatz. So kann mit verschiedenen wählbaren Schlüsseln jeweils der Schlüssel für den Hauptdatensatz selektiert werden, um letztendlich auf den Hauptdatensatz zugreifen zu können.

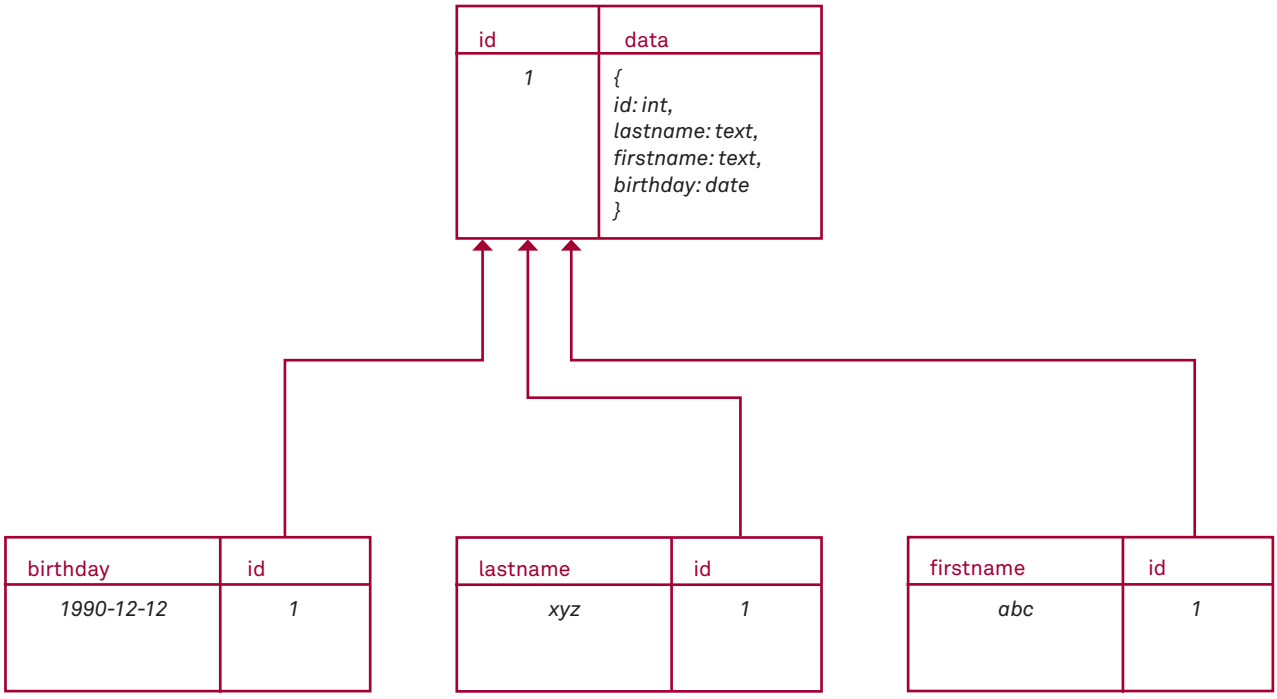


Abbildung 1: Beispieldarstellung Selektierbarkeit mittels Metatabellen

Wählt man dieses Vorgehen, sollte man immer beachten, dass a) beim nachträglichen Hinzufügen weiterer Meta-/Suchtabellen immer ein vollständiger Migrations-Job durchgeführt werden muss, um die neue Tabelle zu füllen. In Cassandra gibt es einen solchen Auto-Mechanismus nicht.
b) je nach gewählter Komplexität der Fachlichkeit immer mehrere Selektierungen notwendig sind, um die Ergebnisse anzeigen zu lassen oder verarbeiten zu können.

2. Suchindex
Ist absehbar, dass die Durchsuchbarkeit des gespeicherten Datenmodells gegeben sein muss, lässt sich das auch über Metatabellen nicht mit adäquatem Aufwand abbilden. Für diesen Anwendungsfall empfiehlt sich der Einsatz einer Suchindex-Komponente. Ein solcher Suchindex bietet von Haus aus viele Möglichkeiten der Filterung und Suche inklusive Relevanz der Ergebnisse. Als Beispiele seien hier Elasticsearch[10] oder SOLR[11] genannt. Hat man Datastax Enterprise[12] (kommerzielle Variante von Cassandra) im Einsatz, wird hier der SOLR-Suchindex mitgeliefert.

Weitere Möglichkeiten eines angepassten Zugriffs auf die Daten sind die Verwendung von Materialized Views oder der Einsatz eines sogenannten (Secondary) Index in Cassandra. Materialized Views befinden sich jedoch noch in der Betaphase und sind nicht für den produktiven Einsatz empfohlen. Gegen die Verwendung von Indizes sprechen technische, durch Cassandra bedingte, Einschränkungen.

CASSANDRA ALS VERTEILTES SYSTEM (EVENTUAL CONSISTENCY)

Ein wichtiger Grundsatz beim Einsatz von verteilten Systemen ist das sogenannte „CAP-Theorem“[13], das besagt, dass nie alle drei Eigenschaften, also **C**onsistency (Konsistenz), **A**vailability (Verfügbarkeit) und **P**artition Tolerance (Ausfallsicherheit), in einem verteilten System gleichzeitig garantiert werden können. Bei Apache Cassandra wird das A und das P garantiert, sofern die Anzahl Nodes im Cluster > 1 und der Replikationsfaktor entsprechend größer 1 gewählt ist. Cassandra verfolgt dabei den Ansatz der Eventual Consistency (keine Konsistenzgarantie im Cluster), was vereinfacht bedeutet, dass ein Datensatz irgendwann konsistent (das heißt über alle Replikas gleich) sein wird, sofern nur eine hinreichend lange Zeit ohne Schreibvorgänge und ohne Fehler vorausgesetzt werden kann.[14]

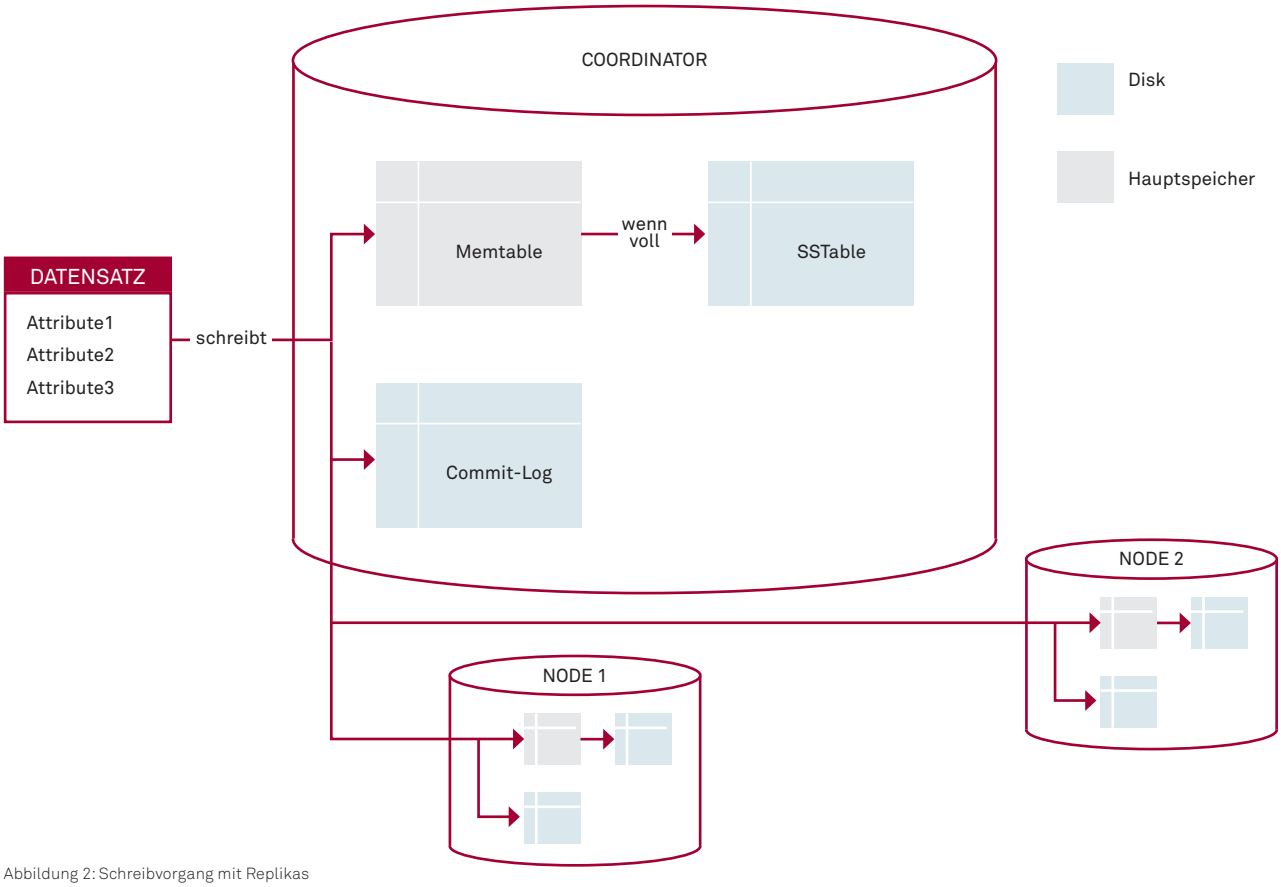


Abbildung 2: Schreibvorgang mit Replikas

EVENTUAL CONSISTENCY IN DER PRAXIS

Wird ein Datensatz geschrieben, speichert der Coordinator (siehe Glossar) den Datensatz im Commit-Log. Beim unerwarteten Ausfall einer Node können die Daten aus dem Commit-Log in die Memtable (siehe Glossar) zurückgeschrieben werden, um hier größtmögliche Konsistenz der Daten zu wahren. Bei der Memtable handelt es sich um das Teilabbild der Daten im Hauptspeicher, da der immer noch um Faktoren schneller ist als die schnellsten Solid State Disks. Gleichzeitig mit dem Schreiben in das Commit-Log verteilt der Coordinator den Datensatz auf (je nach Replikationsfaktor) den oder die Replika-Nodes. Ist die Memtable voll, werden die Daten in die SSTable (String Sorted Table) auf Disk übertragen. Während der Verteilung der Daten auf die verschiedenen Nodes ist es somit möglich, dass bei einer Abfrage und verschiedenen technischen Parametern (Konsistenzlevel beim Lesen, unterschiedliche Koordinatoren pro Verbindung, Anzahl der Replikas) für einen Zeitraum (bis alle Nodes denselben Datenstand haben) unterschiedliche Ergebnisse aus Cassandra zurückgeliefert werden. Diesen Zustand nennt man „eventual consistent“.

GLOSSAR

Cluster	Verbund von einzelnen Cassandra Nodes
Node	Eine Cassandra-Service-Instanz
Coordinator	Erster Kontaktpunkt (Verbindungsschnittstelle) einer Applikation mit Cassandra; jeder Node eines Clusters kann Coordinator sein.
Replikations-faktor	Wert, der angibt, über wie viele Nodes ein Datensatz im Cluster verteilt wird
Partition	Speicherplatz einer Tabelle auf einer Node
Keyspace	Fachliche Einordnung von Tabellen in einem Cassandra-Verbund
Replika	Gespeicherte Kopien eines Datensatzes im Cluster (für Ausfallsicherheit)
Commit-Log	Zwischenspeicher (auf Disk) einer Cassandra Node, der für Back-up-Zwecke verwendet wird
Memtable	Zwischenspeicher (im RAM) einer Cassandra Node, der auf Disk (SSTable) übertragen wird, wenn er voll ist
SSTable	String Sorted Table; Ablageort der Daten auf Disk einer Cassandra Node
Wide-Row-Datenspeicher	Cassandra zählt zu dieser Kategorie. Hier werden Daten in Spalten abgelegt (wie in klassischen SQL-Datenbanken). Das Gegenstück hierzu sind beispielsweise dokumentenbasierte Datenspeicher (Elasticsearch, MongoDB).

CASSANDRA-VERSIONEN

Zurzeit existieren zwei Lizenzmodelle für Cassandra-Datenbanken.

Apache Cassandra (Open-Source-Version im Apache-License-Modell)
Mit dieser Version ist es möglich, eine vollumfängliche, über Rechenzentrumsgrenzen hinweg verteilte Datenspeicherlösung zu implementieren. Ebenso bietet diese Version für Softwareentwickler die Möglichkeit, während der Entwicklung verschiedene Implementierungen mit Cassandra zu testen, ohne dass Lizenzkosten anfallen. Apache Cassandra steht als Docker-Container[15] zur Verfügung, wodurch es beispielsweise für Softwareentwickler oder Mitarbeiter des Betriebs sehr einfach möglich ist, einen oder mehrere Cassandra Nodes im Verbund zu betreiben, um eine Systemlandschaft zu simulieren.

DSE (Datastax Enterprise, lizenzpflichtige, nicht kostenlose Version)[16]
Die Basis-Software entspricht Apache Cassandra, jedoch erhält der Kunde erweiterten Support durch die Firma Datastax. Man erhält optimierte Treiber, erweiterte Konnektivitäten und zusätzliche Softwarepakete. Gerade beim Betrieb größerer und großer Datenbanklandschaften über Rechenzentrumsgrenzen hinweg sollte der Einsatz der lizenzpflichtigen Software erwogen werden.

FAZIT

Werden Softwaresysteme entworfen, stellt sich unweigerlich die Frage, welche Persistenz man wählt. Wenn im Vorfeld darüber Klarheit herrscht, dass große Datenmengen anfallen werden, sich der genaue Umfang jedoch nicht abschätzen lässt, sollte man den Einsatz des verteilten Datenspeichersystems Cassandra in Betracht ziehen. Durch die einfache Skalierbarkeit, hohe Ausfallsicherheit und sehr hohe Geschwindigkeit ist es bestens für den Einsatz im Big-Data-Umfeld geeignet.

Gleichzeitig verlangt Cassandra sowohl vom Betrieb als auch von der Softwareentwicklung ein hohes Maß an Lernbereitschaft. Gerade wenn bei Entwicklung und Betrieb der Fokus bislang auf klassischen RDBMS lag, müssen jetzt beide Seiten voneinander lernen, um das System so optimal wie möglich nutzen zu können und Fehler zu vermeiden.

Eine falsch genutzte Cassandra-Datenbank kann auf allen Seiten für sehr viel Frustration sorgen und das Beheben handwerklicher Fehler kann gerade am Anfang sehr zeit- und kostenintensiv beziehungsweise im späteren laufenden Betrieb nahezu unmöglich werden.

Cassandra ist auch nicht für alle Einsatzszenarien gedacht. Steht eine hohe transaktionale Sicherheit beziehungsweise Konsistenz der Daten im Vordergrund, ist man bei einem klassischen RDBMS besser aufgehoben. Dies trifft auch dann zu, wenn sich gleiche Datensätze oft ändern oder Datensätze häufig gelöscht werden.

Für den Betrieb als verteilter Datenspeicher im eigenen Rechenzentrum ist Cassandra derzeit alternativlos. Es existiert zwar mit SkyllaDB[17] eine von Cassandra abgewandelte Version, allerdings ohne professionellen Support. Grundsätzlich andere Datenspeichermethoden, wie beispielsweise MongoDb[18], die Daten als Dokumente speichert, oder Redis[19], bei der es sich um eine In-Memory-Datenbank handelt, sollten zumindest als Vertreter der NoSQL-Datenbanken auch für das Einsatzszenario wie bei Cassandra in Erwägung gezogen werden.

Wird der Einsatz in der Public Cloud erwogen, stehen hier zum Beispiel bei Amazons Cloud AWS die DynamoDB[20] oder bei Microsoft Azure die CosmosDB[21] beziehungsweise bei Google BigTable[22] als Vertreter der NoSQL-Wide-Row-Datenspeicher zur Verfügung. ●

1 Süddeutsche Zeitung, 23.09.1995.
2 http://cassandra.apache.org/ (abgerufen am 28.02.2020).
3 https://www.facebook.com/note.php?note_id=24413138919 (abgerufen am 28.02.2020).
4 https://www.infoq.com/news/2014/10/netflix-cassandra/ (abgerufen am 28.02.2020).
5 https://academy.datastax.com/planet-cassandra/nosql-performance-benchmarks (abgerufen am 28.02.2020).
6 https://de.wikipedia.org/wiki/DevOps (abgerufen am 28.02.2020).
7 https://de.wikipedia.org/wiki/Domain-driven_Design (abgerufen am 28.02.2020).
8 https://docs.datastax.com/en/ddac/doc/datastax_enterprise/dbInternals/dbIntConfigConsistency.html (abgerufen am 28.02.2020).
9 https://docs.apigee.com/private-cloud/v4.17.09/about-cassandra-replication-factor-and-consistency-level#aboutthecassandraconsistencylevel (abgerufen am 28.02.2020).
10 https://www.elastic.co/de/ (abgerufen am 28.02.2020).
11 https://lucene.apache.org/solr/ (abgerufen am 28.02.2020).
12 https://www.datastax.com/products/datastax-enterprise (abgerufen am 28.02.2020).
13 https://de.wikipedia.org/wiki/CAP-Theorem (abgerufen am 28.02.2020).
14 https://de.wikipedia.org/wiki/Konsistenz_(Datenspeicherung)#Verteilte_Systeme (abgerufen am 28.02.2020).
15 https://hub.docker.com/_/cassandra (abgerufen am 28.02.2020).
16 https://db-engines.com/de/system/Cassandra%3BDatastax+Enterprise (abgerufen am 28.02.2020).
17 https://www.scylladb.com/ (abgerufen am 28.02.2020).
18 https://www.mongodb.com/de (abgerufen am 28.02.2020).
19 https://redis.io/ (abgerufen am 28.02.2020).
20 https://aws.amazon.com/de/dynamodb/ (abgerufen am 28.02.2020).
21 https://azure.microsoft.com/de-de/services/cosmos-db/ (abgerufen am 28.02.2020).
22 https://cloud.google.com/bigtable/ (abgerufen am 28.02.2020).