Was Sie über NoSQL Datenbanken wissen sollten

Dr. Halil-Cem Gürsoy
adesso AG
Dortmund

Schlüsselworte
NoSQL, CAP-Theorem, BASE, MongoDB, Cassandra, Neo4J, Redis

Einleitung
NoSQL-Datenbanken haben in den letzten Jahren ihren festen Platz unter den in Individualprojekten eingesetzten Technologien erkämpft. Der Begriff „NoSQL“ steht aber dabei für ein sehr breites Spektrum verschiedener Technologien und unter Umständen auch unterschiedlicher Konzepte.
Daher ist das Spektrum der Datenbanken, die unter diesem Namen zusammengefasst werden, recht groß. Dies reicht von "einfachen" Key-Value-Datenbanken über dokumentenorientierte Datenbanken bis hin zu komplexen Graphendatenbanken, in denen ganze Netzstrukturen und Objektbäume abgelegt werden können.
Dabei unterscheiden sich diese Datenbanken teilweise stark von klassischen relationalen Datenbanken; Stichworte sind hier das CAP-Theorem sowie *BASE* auf die hier eingegangen werden soll.

„Nicht nur SQL“

Einführend soll erst einmal mit einem häufig anzutreffenden Missverständnis aufgeräumt werden. Denn „NoSQL“ steht nicht für „Kein SQL“ sondern „Not only SQL“, also „Nicht nur SQL“. Damit wird schon klar gemacht, dass wir es hier mit einem etwas diffusen Bild zu tun haben unter dem zuerst einmal nicht sehr viel verstanden werden kann.

Klassifizierung von NoSQL-Dtenbanken

Taucht man dagegen ein in diese Welt der Datenbanken wird man von der Diversität dieser überrascht. Diese Datenbanken lassen sich sehr grob in vier Kategorien einteilen:

- Key/Value-Datenbanken
- Column Family-Datenbanken
- Dokumentenorientirte Datenbanken
- Graphenorientierte Datenbanken

Die einfachste Form dieser Datenbanken stellen Key/Value-Datenbanken dar. Diese speichern „nur“ Schlüssel/Wert-Paare ab. Alleine daraus geht schon hervor, dass in diesen Datenbanken keine komplexen Datenstrukturen abgelegt werden können – ein Schema wie aus der relationalen Welt ist hier also nicht möglich und nicht vorhanden. Diese Datenbanken sind zudem häufig darauf spezialisiert mit sehr vielen Daten zu arbeiten und sehr hohe Lese- und Schreibgeschwindigkeiten zu bieten. Zudem sind sie häufig auf ein sehr hohes Grad an Verteilung ausgelegt. Klassische Anwendungsfälle für solche Datenbanken sind beispielsweise das Speichern von Massen-Messdaten oder auch Cachingsysteme. Bekannte Vertreter dieser Gattung wären unter anderem *Riak*, *Redis* oder auch *Amazon SimpleDB*.

Betrachten wir die nächste Gattung der NoSQL-Datenbanken, den Column Family-Datenbanken (auch häufig als *Wide Column* Datenbanken bezeichnet) so können schon etwas komplexere Strukturen in diesen abgespeichert werden. Die Daten werden auch als Schlüssel/Wert-Paare abgespeichert, aber mehrere dieser können zu Spalten kombiniert werden. Dabei ist es interessant dass in einer „Tabelle" Datensätze mit unterschiedlichen Spalten enthalten sein können – auch bei diesen Datenbaken wird kein Schema wie aus relationalen Datenbanken bekannt für eine Tabelle erzwungen. Einige Vertreter dieser Gattung legen zudem die Daten physikalisch spaltenorientiert ab. Gerade bei Anwendungen wie *Business Iteligence*, bei denen sehr große Datenmengen ausgewertet werden müssen, können solche Datenbanken ihre Vorteile ausspielen. Daher verwundert es nicht, dass Vertreter dieser Gattung wie *Apache Cassandra* häufig im *BI*-Umfeld anzutreffen sind. Auch wenn gegenüber Key/Value-Datenbanken komplexere Strukturen abgelegt werden können muss beachtet werden, dass zumeist im Gegensatz zu relationalen Datenbanken keine Relationen zwischen einzelnen Datensätzen angelegt werden können – solche Relationen müssen Applikationsseitig gepflegt werden. Damit ist auch klar, dass solche Datenbanken keine *Joins* unterstützen. Das „Warum" erschließt sich schnell durch die Tatsache, dass die Datenbanken auf eine starke horizontale Verteilung als auch hohe Performance ausgelegt sind.

Müssen noch komplexere Datenstrukturen abgelegt werden bieten sich die sogenannten Dokumentenorientierten Datenbanken an. Diese sind darauf spezialisiert sogenannte Dokumente zu speichern und werden in den meisten Vertretern dieser Gattung im JSON-Format (*JavaScript Object Notation*) abgelegt und dargestellt. Auch diese Datenbanken sind „Schemafrei", das heisst, in einer „Tabelle" (bei einem der prominentesten dokumentenorientierten Datenbanken, der *MongoDB*, sprechen wir von „*Collections*") können verschiedene Strukturen abgelegt werden. Auch diese Datenbanken sind zumeist auf eine horizontale Skalierung ausgelegt. Eine Funktion, die bei dem Namen naheliegt, wird aber bei den meisten dieser Datenbanken vermisst: die Volltextsuche. Bei MongoDB wurde erst kürzlich eine rudimentäre Volltextsuche hinzugefügt. Dagegen sticht *ElasticSearch* aus der Menge hervor, denn hier liegt ein Zwitter aus Dokumentenorientierter Datenbank und einer *Apache Lucene*-basierter Suchengine vor.

Zu guter Letzt gibt es noch die Gruppe der graphenorientierten Datenbanken. Wie der Name schon nahelegt werden in diesen Datenbanken Graphen, also Knoten und deren Beziehungen untereinander, abgespeichert, wobei sowohl die Knoten als auch die Beziehungen Attribute aufweisen können. Bestes Beispiel für solche Graphen sind Soziale Netzwerke oder auch Infrastrukturen von z.B. Telekommunikations-Dienstanbietern. Ein bekannter Vertreter dieser Gattung ist *Neo4J*. Diese Datenbank sticht zudem aus der Menge der NoSQL-Datenbanken durch die Unterstützung von JTA (*Java Transaction API*) und ACID auf – dazu etwas später mehr.

Das CAP-Theorem

Beschäftigt man sich mit NoSQL-Datenbanken kommt man unweigerlich mit dem CAP-Theorem, welches von Andrew Brewer im Jahr 2000 im Rahmen seiner PODC Keynote dargestellt wurde, in Berührung. Das Akronym steht dabei für die Eigenschaften Konsistenz (*Consistency*), Verfügbarkeit (*Availability*) und Partitionierungstoleranz (*Partitioning*). Das Theorem besagt, dass in einem verteilten System maximal nur zwei dieser Eigenschaften vollständig erfüllt sein können. Relationale Datenbanken bilden zumeist klassische *CA*-Systeme. Durch Transkationen (ggfs durch Anwendung von Verfahren wie *Two Phase Commit*) wird eine starke Konsistenz erreicht, die Systeme selbst stehen, Beispielsweise durch Verwendung von hochausfallsicherer Hardware, immer zur Verfügung. Fällt aber nun die Netzwerkverbindung zwischen zwei Knoten aus, so kann das Gesamtsystem die Konsistent der Daten nicht mehr gewährleisten – das System ist also gegenüber einer Netzwerkpartitionierung nicht tolerant. Betrachten wir dagegen das andere Extrem bezüglich der Eigenschaft „P": Das allgegenwärtige DNS. Hier verzichtet das Gesamtsystem bewusst auf starke

Konsistenz. Dies bedeutet, es kann eine gewisse Zeit verstreichen, bis auf allen Knoten ein Datensatz auf dem aktuellen Stand ist. Allerdings ist dieses System immer Verfügbar (sehr viele Knoten) und tolerant gegenüber Netzwerkausfällen: die Daten werden halt dann weiter repliziert wenn das Netzwerk wieder funktionsfähig sind. Das DNS ist damit ein *AP*-System. Weiterhin haben wir es auch mit sogenannten *CP*-Systemen zu tun: dies sind Datenbanken, die Wert auf eine starke Konsistenz und Partitionierungstoleranz legen, aber dafür müssen diese auf Verfügbarkeit verzichten.

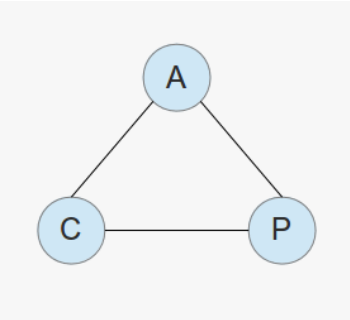

Abb. 1: Das CAP-Dreieck nach Andrew Brewer

BASE anstellen von ACID und Konsistenz

Viele NoSQL-Datenbanken lassen sich in die Gruppe der *AP*-Systeme einordnen, wobei bei vielen die Möglichkeit besteht, dies durch Konfiguration zu beeinflussen. Die Tatsache, dass für Verfügbarkeit (*A*) und Partitionstoleranz (P) auf Konsistenz (C) verzichtet wird führte zu dem logischen Gegenstück von ACID: *BASE*. BASE steht dabei für ***B**asically **A**vailable, **S**oft-state, **E**ventually consistent.* „Eventually consistent“ wird leider im deutschsprachigen Raum immer noch häufig mit „eventuell Konsistent“ übersetzt was grundlegend falsch ist. Vielmehr bedeutet dies, dass solche Systeme eine *schlussendliche Konsistenz* aufweisen. Konsistenz im Sinne von BASE bedeutet, dass auf allen Knoten respektive in Replikaten eines Datensatzes die gleichen Daten vorhanden sind. Dies darf nicht mit referentieller Integrität verwechselt werden.

Wie wird nun in solchen Datenbanken eine hohe Verfügbarkeit erreicht? Schlicht durch Replikation der Daten auf mehrere Knoten. Die meisten Datenbanken verfügen über Mechanismen um schnell einen Replikations-Cluster aufzubauen und untereinander eine Koordination durchzuführen. Fallen nun Knoten aus, so sind die überlebenden Knoten in der Lage Anfragen zu beantworten oder auch neue Daten abzuspeichern. Und hier kommt nun der Begriff „Schlussendliche Konsistenz“ wieder ins Spiel – das System benötigt dann halt eine gewisse Zeit bis auf allen Knoten die aktuellsten Daten gelandet sind. Der Applikationsentwickler muss nun entscheiden ob er die aktuellsten Daten benötigt oder mit gegebenenfalls etwas veralteten Daten leben kann. Diese Entscheidung ist immer Abhängig vom Anwendungsfall.

In diesem Zusammenhang wird auch immer die Frage nach Transaktionen und atomaren Änderungen aufgeworfen: die meisten NoSQL-Datenbanken unterstützen nun einmal keine ACID-Transaktionen (eine Ausnahme bildet wie schon erwähnt die Graphendatenbank *Neo4J*). Die meisten NoSQL-Datenbanken garantieren hier Atomizität auf der Ebene eines Datensatzes. MongoDB als Beispiel betrachtet garantiert, dass eine Änderung an einem Dokument atomar abläuft. Mehrere aufeinanderfolgende Änderungen an einem Dokument oder Änderungen über mehrere Dokumente hinweg werden nicht atomar behandelt. Hier muss der Applikationsentwickler selber Sorge dafür tragen und z.B. bereits bekannte Konzepte im Zusammenhang mit verteilten Systemen wie „*Compensation*“ anwenden. Der Grund, warum auf solche Aspekte verzichtet wird ist recht schlicht: Knotenübergreifende Koordination von atomaren Vorgängen kostet einfach zu viel Ressourcen –

Systeme wie die MongoDB, die auf hohe horizontale Skalierbarkeit und Performance ausgelegt sind nehmen dafür dann gerne diesen Verzicht an.

Suchen mit Map/Reduce

Im Zusammenhang mit NoSQL-Datenbanken wird auch sehr häufig der Begriff *Map/Reduce* genannt. Hierbei handelt es sich um ein von Google beschriebenes Verfahren um in einem verteilten System schnell zu suchen. Das besondere ist, dass mit Hilfe dieses Verfahrens bei steigender Datenmenge die Suchzeit dennoch beinahe konstant gehalten werden kann wenn die Daten horizontal verteilt werden. Dabei besteht dieses Verfahren aus zwei Schritten. Im ersten Schritt, dem *Map*, werden alle Knoten im System nach bestimmten Daten gefragt, Beispielsweise die Menge bestimmter Artikel in einem Warenwirtschaftssystem. Im nächsten Schritt, dem *Reduce*, werden alle Antworten gesammelt und abhängig vom Anwendungsfall aggregiert, gruppiert usw. Dieses Verfahren garantiert schnelle Antworten, hat aber einen ganz offensichtlichen Nachteil: die Ressourcen-Effizienz ist nicht gegeben da alle Knoten gefragt werden. Zudem ist die Antwortzeit abhängig vom langsamsten Knoten des Systems. Dann kommen Verfahren wie *Harvest/Yield* diesem Zusammenhang zum Tragen, mit deren Hilfe die nichtfunktionalen Anforderungen an solche Anfragen definiert werden können.

Zusammenfassung

Unter dem Begriff NoSQL-Datenbanken werden ganz unterschiedliche Datenbanksysteme zusammengefasst. Diese lassen sich sehr grob in vier Hauptkategorien untergliedern. Vielen dieser Datenbanken ist gemein dass sie auf hohe Performance und hohe horizontale Skalierbarkeit ausgelegt sind. ACID-Transaktionen werden selten unterstützt, vielmehr sind diese Datenbanken meistens BASE-basiert. Das CAP-Theorem hat einen entscheidenden Einfluss auf die Architektur dieser Systeme.

Kontaktadresse:
Dr. Halil-Cem Gürsoy
adesso AG
Stockholmer Allee 24
D-44269 Dortmund

Telefon:	+49 231 930-9330
Mobil:	M +49 178 2808148
E-Mail	halil-cem.guersoy@adesso.de
Internet:	www.adesso.de
Twitter:	@hgutwit
G+:	http://goo.gl/hljRS