Gefangen im CAP Theorem
Warum Datenbanken nicht gut skalieren

Thomas Klughardt
Quest Software
Köln

Schlüsselworte
Skalierbarkeit, Performance, Konsistenz, Verfügbarkeit, ACID, Transaktionen, NoSQL, BASE

Einleitung
In der Applikationslandschaft haben es die relationalen Datenbanken nicht leicht. Sie sind verantwortlich dafür, den wichtigsten Bestandteil für geschäftskritische Anwendung zu verwalten, die Daten. Dabei reicht es nicht die Daten abzulegen und bei Bedarf wieder zur Verfügung zu stellen, die Datenbank hat auch ACID konform zu arbeiten. Das heißt, unter anderem, dass die Daten in sich stimmig und zueinander Konsistent sind, dass es immer nur einen zentralen Bestand an Daten gibt und somit keine Konflikte und Widersprüche auftreten. Dazu kommen einige andere Anforderungen, auf die in dem Vortrag eingegangen wird.

ACID Konform zu arbeiten ist aber nicht alles, zusätzlich wird erwartet, dass die Daten ständig verfügbar sind und schnell zur Verfügung gestellt werden können. Und dann soll die Datenbank auch noch gut skalieren, steigt das Datenvolumen oder das Transaktionsvolumen an, sollen die Antwortzeiten trotzdem gleich bleiben oder nur linear ansteigen. Diese Anforderungen sind ein Widerspruch; das besagt Eric Brewster‘s CAP Theorem, um das es in diesem Vortrag gehen wird.

Insgesamt geht es in diesem Vortrag also um teilweise unmöglich zu erfüllende Anforderungen an relationale Datenbanken, um Alternativen wie NoSQL (Not only SQL) Datenbanken und Möglichkeiten, diese einzusetzen. Dazu wird erklärt, warum von allen Anforderungen gerade Skalierbarkeit die unwichtigste ist.

Definition: Von welchen Datenbanken reden wir hier?
Grundsätzlich bezeichnet der Begriff Datenbank nur ein Programm, das in der Lage ist, Daten abzulegen und bereitzustellen. Wenn wir über eine Datenbank reden, meinen wir aber üblicherweise ein transaktionsbasierendes Relationales Datenbank Management System (RDBMS), das für die Transaktionen bestimmte Anforderungen erfüllt. In den meisten Fällen (es gibt Ausnahmen wie zum Beispiel andere Isolation Level) entspricht das den ACID (oder deutsch AKID) Eigenschaften, das steht für:

1. *Atomarität*
   Jede Transaktion wird mit allen Datenänderungen ganz oder gar nicht durchgeführt. Erst wenn jede Operation erfolgreich war, war die Transaktion erfolgreich, sonst wird sie komplett zurückgerollt.

2. *Konsistenz*
   Alle Daten entsprechen vor und nach einer Transaktion definierten Integritätsbedingungen, sie passen also zueinander und es gibt keine Widersprüche. Die Integritätsbedingungen drücken dabei die Abhängigkeiten im Datenmodell aus.

3. *Isolation*
   Erst wenn die Transaktion erfolgreich war und damit die Daten konsistent sind, werden die Daten für andere Transaktionen sichtbar. Es ist von anderen Sessions aus abhängig vom Start einer Abfrage der Zustand vor oder nach einer anderen Transaktion sichtbar, aber nie der Zustand einer nicht abgeschlossenen Transaktion.

4. *Durability / Persistenz*
   Eine abgeschlossene Transaktion wird dauerhaft in der Datenbank gespeichert und geht danach nicht verloren. Sobald die Rückmeldung kommt, dass die Transaktion erfolgreich war, sind alle Änderungen persistent gespeichert.

Neben den relationalen Datenbanken gibt es noch andere Datenbanktypen, die andere Eigenschaften aber auch einen anderen Fokus haben. Dazu gehören zum Beispiel die NoSQL Datenbanken, die die ACID Eigenschaften nicht erfüllen und meist auch nicht mit Transaktionen arbeiten. ACID Konformität und Transaktionen bedeuten für die Datenbank Engine einen großen Overhead, werden sie nicht benötigt, dann sind NoSQL Datenbanken eine echte Alternative. Dass die Datenbank diese Arbeit dann nicht machen muss macht sich dann natürlich in schnelleren Operationen, besserer Skalierbarkeit und geringerem Ressourcenbedarf bemerkbar. Ein weiterer Pluspunkt sind die finanziellen Aspekte, die meisten NoSQL Datenbanken sind frei verfügbar, auf der anderen Seite ist vom Hersteller meist kein Support erhältlich.

Was besagt jetzt das CAP Theorem?
Im Wesentlichen besagt das CAP Theorem, dass es keine Möglichkeit gibt, die Daten zur gleichen Zeit Konsistent, jederzeit (schnell) Verfügbar sein und gut skalieren können. Will man eine bestimmte Eigenschaft erreichen, muss man Einschränkungen bei den anderen Eigenschaften hinnehmen.

1. *Konsistenz*
   Wenn die Daten immer konsistent sein müssen, dann muss die Konsistenz sichergestellt sein, bevor die Daten wieder verfügbar werden. Das heißt, dass es eine zentrale Instanz der Daten, den Master, geben muss, bei dem die Konsistenz sichergestellt wird.

2. *Verfügbarkeit*
   Wenn die Daten jederzeit verfügbar sein müssen, dann kann man das Zurückgeben der Daten nicht verzögern. Das bedeutet dann auch, dass bei redundanter Datenhaltung alle Datenbestände jederzeit abgefragt werden können und das Ergebnis sofort geliefert werden muss.

3. *Skalierbarkeit*
   Wenn der Datenbestand gut skalierbar sein soll, muss das System unabhängig von Datenmenge und Last gleiche oder ähnliche Antwortzeiten liefern. Das funktioniert nur dann, wenn die Daten redundant vorgehalten werden und die Einzelbestände unabhängig voneinander abgefragt werden können.

Wenn man sich die einzelnen Anforderungen für die Eigenschaften ansieht, wird schnell klar, dass sie sich teilweise widersprechen und es deshalb kein System geben kann, das in der Lage ist, alle Anforderungen zu erfüllen. Das ist die Aussage des CAP Theorems, das bisher nicht widerlegt wurde.

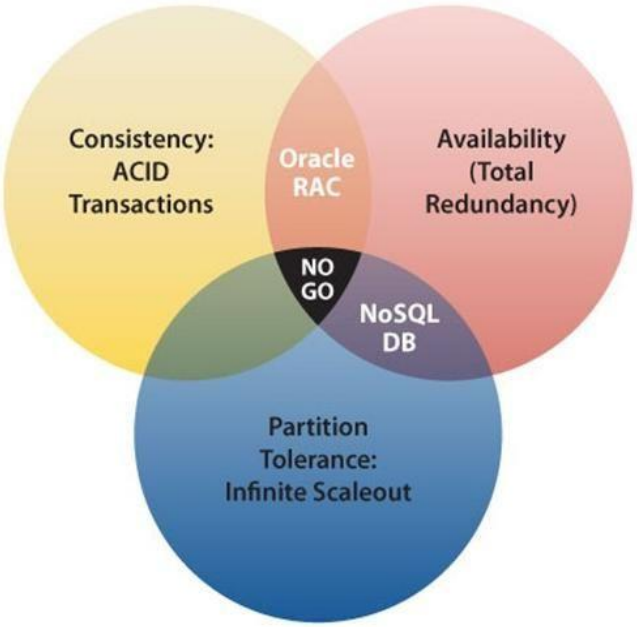

Abb. 1: Das CAP Theorem als Venn Diagramm (Quelle: DBPedias.com)

Will man die Daten jederzeit konsistent halten, dann benötigt man eben diesen einen Master, gegen den ständig abgeglichen wird. Dafür muss man eine der anderen Eigenschaften opfern. Entweder sind die Daten jederzeit konsistent und verfügbar, dann muss man sie aber zentral an einer Stelle halten, so dass sie während der Eingabe auf Konsistenz geprüft werden können. Darunter leidet natürlich die Skalierbarkeit.
Will man die Daten dagegen gut skalieren lassen, muss man mehrere Bestände an verschiedenen Stellen halten und über Replikation pflegen. Sollen sie dazu konsistent sein, dann darf man sie nicht herausgeben, muss sie also sperren, bis die Konsistenz sichergestellt ist. Darunter leidet dann natürlich die Verfügbarkeit.

Natürlich könnte man die Daten auch an mehreren Stellen jederzeit verfügbar machen. In dem Fall skaliert das System gut und die Daten können jederzeit geändert und abgefragt werden, es gibt aber keine Möglichkeit sicherzustellen, dass sie auch konsistent sind. Für diesen Anwendungsfall hat sich in Abgrenzung zu den ACID Eigenschaften der Begriff BASE durchgesetzt. BASE steht für **B**asically **A**vailable, **S**oft State, **E**ventual Consistency. Grundsätzlich sind die Daten also immer verfügbar, allerdings ist der Zustand der Daten nicht bekannt, also ist nicht bekannt, ob sie konsistent sind. Über die Zeit werden die Daten aber doch aufbereitet und irgendwann auch konsistent, solange sich nicht in der Zwischenzeit wieder etwas geändert hat. Für ein transaktionsbasierendes System wie eine Auftragsverwaltung sind diese Eigenschaften natürlich unbrauchbar, aber wenn wir zum Beispiel eine Websuche machen, ist uns egal, ob das Suchergebnis aktuell, eine halbe Stunde alt oder eine Mischung aus beidem ist. In solchen Szenarien funktioniert ein derartiges System wunderbar.

Wie kann ich mit dem CAP Theorem umgehen?
Eine Möglichkeit haben wir gerade kennengelernt. Wenn ich eine der Eigenschaften nicht erfüllen muss, muss ich mich am CAP Theorem nicht stören.

Konsistenz
Benötige ich keine Konsistenz in den Daten, dann kann ich ein beliebig skalierbares Datenbanksystem schaffen. Es ist in dem Fall dann eben keine relationale Datenbank, sondern eine Datenhalde. Oft

haben diese Datenhalden keine Schemastrukturen und man kann keine Join Operationen durchführen. Die würden auch nichts bringen, weil die Foreign Key Constraints, die man aus relationalen Datenbanken kennt, hier sowieso nicht zum Tragen kommen. Diese Datenbanken verstehen deshalb oft keine SQL Syntax oder bilden sie auf eigene Analyse Jobs ab, deshalb werden sie auch als Not only SQL (NoSQL) Datenbanken bezeichnet.
Gerade für Datenbankadministratoren und Entwickler, die sich gut mit relationalen Datenbanken auskennen, ist der Umgang mit diesen Systemen nicht einfach, weil man nicht in relationalen Schemata denken darf. Statt oder zusätzlich zu SQL kommen dann oft exotische funktionale Programmiersprachen wie Erlang zum Einsatz, mit denen man über Map-Reduce Algorithmen mächtige Datenanalysejobs laufen lässt
Es gibt aber Tools wie das Freeware Produkt Toad for Cloud Databases, mit dem über einen sogenannten Data Hub die schemalosen Daten auf relationale Strukturen abbilden kann, so dass sie sich wie in den gewohnten Datenbanksystemen manipulieren und abfragen lassen.

Verfügbarkeit
Ständige Verfügbarkeit ist genau wie Konsistenz eine der Haupteigenschaften für Datenbanken und kaum ein Anwender wird hier Abstriche zulassen. Sollte darauf aber doch irgendwo zugunsten von Skalierbarkeit verzichtet werden können, so empfehlen sich synchrone Replikationsverfahren, um hohe Skalierbarkeit und Konsistenz zu erreichen, eben mit dem Nachteil, dass Anwender oft lange auf Ergebnisse oder auf die Manipulation von Daten warten müssen.

Skalierbarkeit
In den meisten Fällen wird man Konsistenz und Verfügbarkeit voraussetzen. Bei der Skalierbarkeit werden die Abstriche gemacht und daher kommt auch der Titel. Darauf sollte man sich aber nicht ausruhen, trotz der Abstriche lässt sich bei der Skalierbarkeit über ein gutes Anwendungsdesign und Optimierungen eine Menge erreichen. Irgendwann wird aber das Ende der Fahnenstange erreicht sein, beliebig skaliert kein RDBMS und nach allen anderen Optimierungen helfen am Ende dann nur stärkere Maschinen gegen hohe Antwortzeiten. Auch ein Oracle RAC sorgt nicht dafür, dass eine Datenbank besser skaliert, denn auch hier haben wir immer einen Masterdatensatz, gegen den alles abgeglichen werden muss, um die Daten konsistent zu halten.

Fazit
Das Fazit ist diesmal recht kurz. Man könnte sagen: Datenbanken würden besser skalieren, wenn die Daten nicht immer verfügbar und konsistent sein müssten. Das sind aber Anforderungen, die aus der Anwendung und den Fachbereichen kommen. Ein gutes Anwendungsdesign hilft und wenn die Datenbank nicht skalieren will, ist selten der Datenbankadministrator Schuld, auch wenn er fast immer die Schelte einstecken muss.
Es lohnt sich, sich mit Themen wie NoSQL Datenbanken zu beschäftigen, sie können überall da zum Einsatz kommen, wo ständige Konsistenz nicht notwendig ist. Für die meisten Anwendungen in normalen Unternehmen werden sich NoSQL Datenbanken aber nicht eignen.

Kontaktadresse:
Thomas Klughardt
Quest Software
Im Mediapark, 4e
D-50670 Köln

Telefon:	+49 (0) 221-5777 4114
Fax:	+49 (0) 221-5777 40
E-Mail	thomas.klughardt@quest.com
Internet:	www.questsoftware.de