NoSQL - schöne neue Datenbankwelt oder vorübergehender Hype?

Bis vor kurzem war die Frage, welche Datenbank man in seiner Applikation verwenden soll, normalerweise kein heiß diskutiertes Thema. Wo aus Kostengründen keine kommerzielle Lösung wie **Oracle**, **SQL Server** oder **DB2** in Frage kam, gab es immer genügend Open Source Lösungen wie **MySQL** oder **PostgreSQL**. Allenfalls kam noch eine genügsame Variante wie **SQLite** in Betracht. Für die Anwendungsentwicklung war das meist ziemlich belanglos, weil alle Datenbanken entweder über standardisiertes SQL angesprochen wurden oder eine Persistenzschicht (wie etwa Hibernate) die letzten feinen Unterschiede gänzlich wegabstrahierte.

Mit dieser Eintracht in der Datenbankwelt ist es vorerst vorbei. Plötzlich werden mehr und mehr Lösungen angeboten, die vom relationalen Datenbankschema abweichen aber dafür bisher ungeahnte Möglichkeiten anbieten. Dabei bringen sie allerdings auch Nachteile mit sich, welche in der Aufbruchsstimmung gerne einmal übersehen oder unterschätzt werden. Jedenfalls ist die Frage nach dem bestgeeigneten Datenbanksystem wieder sehr spannend geworden.

Alles Neu

Das Konzept der relationalen Datenbanksysteme stammt aus der Zeit bevor MS-DOS, C++ oder gar Java entwickelt wurden. Seither hat sich offensichtlich einiges in der IT-Welt verändert.

Relationale Datenbanken arbeiten alle im Wesentlichen nach einem einheitlichen und bewährten Prinzip, das für viele Anforderungen auch sehr gut geeignet ist. Wenn man sich allerdings einmal auf einem Gebiet bewegt, das bezüglich Performanz, Skalierbarkeit, Verfügbarkeit oder Flexibilität besondere Ansprüche stellt, kann diese „one size fits all“ Lösung an ihre Grenzen kommen und, wenn überhaupt, nur mit unverhältnismäßig großem finanziellen Aufwand eingesetzt werden.

So findet Google aus Milliarden von Webseiten in Sekundenbruchteilen die relevantesten Übereinstimmungen mit einem Suchbegriff, bewertet nach Dutzenden von Kriterien. Das alles für tausende Abfragen pro Sekunde, wobei die Verfügbarkeit der Google Suchseite geradezu sprichwörtlich ist. Mit herkömmlicher Datenbanktechnologie sind derartige Anforderungen nicht realisierbar. Nun gibt es wohl wenige Applikationen, die Ansprüche in der quantitativen Größenordnung von Google haben, aber das bedeutet noch lange nicht, dass man von den Innovationen der Google Ingenieure nicht auch in kleinerem Rahmen profitieren könnte.

Was ist NoSQL

NoSQL ist ein Überbegriff für alternative, nicht-relationale Datenbanksysteme mit dem Anspruch hohe Performanz, Skalierbarkeit oder Ausfallssicherheit zu bieten. Aus technischer Sicht haben sie aber wenig gemeinsam, und sie sind jeweils für spezielle Einsatzszenarien optimiert.
Einen wahren Boom erleben diese Systeme erst in den letzten Jahren durch entsprechende Veröffentlichungen der großen Player, wie beispielsweise Google oder Amazon und durch zahlreiche

dadurch inspirierte Open Source Projekte. Einige der Lösungen waren aber schon lange etabliert bevor der Begriff NoSQL erfunden wurde.

Entsprechend ihrer Grundfunktionalitäten können die meisten NoSQL Systeme den Hauptgruppen Key-Value Stores, Wide-Column Stores oder Document Stores zugeordnet werden.

Key-Value Stores

Key-Value Stores speichern in ihrer einfachsten Form zu einem Schlüssel jeweils einen zugehörigen Wert. Durch dieses simple Datenmodell können solche Systeme sehr hohe Performanz erreichen, und auch fortgeschrittenere Konzepte, wie die Verteilung der Daten über mehrere Knoten hinweg, sind bei manchen Systemen relativ einfach zu realisieren.

Eine der altbewährten Lösungen ist **Memcached**, welches sehr häufig auf Websites zusammen mit einer relationalen Datenbank wie MySQL eingesetzt wird. Dabei werden zum Beispiel die einmal mittels MySQL Abfragen generierten Webseiten im Hauptspeicher zusammen mit der URL zwischengespeichert, und können von dort sehr rasch wiedergefunden werden. Memcached arbeitet im Client-Server Modus und ist, obwohl oft als reine Caching Lösung eingesetzt, ein Repräsentant dieser einfachsten Klasse von NoSQL Datenbanken. Das Beispiel zeigt auch klar, wie eine Kombination von mehreren Technologien die Gesamtapplikation optimieren kann.

Memcached ist aber nur das eine Ende des Spektrums. Es gibt auch Key-Value Stores mit reicherer Funktionalität, wie beispielsweise **Redis**, das komplexere Datenstrukturen wie Mengen oder sortierte Listen als Werte zulässt, und Funktionen wie server-seitige Scripts und Message Queues anbietet. Damit kommt es schon für eine wesentlich weitere Menge von Applikationen als hoch-performante Datenbanklösung in Frage. Wie Memcached ist auch Redis eine Im-Memory Datenbank. Das bedeutet allerdings nicht unbedingt, dass Daten nur flüchtig vorhanden sind, und etwa bei einem Systemabsturz verloren gehen. Moderne In-Memory Datenbanken bieten mittels Snapshots oder Change Logs Möglichkeiten der Datenpersistenz, die vom Standpunkt der Dauerhaftigkeit einem klassischen RDBMS durchaus gleichwertig sind. In-Memory bedeutet dann lediglich, dass immer alle Daten gleichzeitig im Speicher gehalten werden. Damit ist natürlich die Datenmenge pro Server limitiert, es ermöglicht aber andererseits alle Algorithmen auf diese Gegebenheit hin zu optimieren, so dass auch ein RDBMS mit noch so gutem Cache in der Regel performancemäßig nicht mithalten kann.

Manche Key-Value Stores - wie etwa **Riak -** bieten die Möglichkeit die Daten mit relativ einfachen Mitteln auf viele unabhängige Server zu verteilen. Das ermöglicht nicht nur die Verwaltung enormer Datenmengen, sondern bringt auch Lastverteilung bei vielen gleichzeitigen Zugriffen, und nicht zuletzt kann die Ausfallssicherheit durch redundante Speicherung erhöht werden. Aufgrund der nach wie vor einfachen Datenstrukturen bleibt dabei die Gesamtkomplexität trotzdem in überschaubarem Rahmen, der auch von Startups oder kleinen IT-Abteilungen zu bewältigen ist. Dagegen ist beim Versuch ein RDBMS mit Datenmengen zu befüllen, die über die Kapazität eines einzelnen Servers hinausgehen und gleichzeitig mittels redundanter Kopien gegen Serverausfälle zu schützen, schon so manches Projektbudget gesprengt worden.

Wide Column Stores

Wenn die Datenkomplexität die Möglichkeiten von Key-Value Stores übersteigen, kommen Wide Column Stores ins Spiel. Diese Klasse von NoSQL Datenbanksystemen basiert auf Konzepten, die Google mit **BigTable** publik gemacht hat. Die einzelnen Datenelemente sind hier nicht nur einem Schlüssel zugeordnet (wie bei Key-Value Stores), sondern stehen in einer zweidimensionalen Anordnung wie bei RDBMS. Allerdings sind diese zwei Dimensionen beliebig gestaltbar und nicht wie bei RDBMS einem fixen Schema unterworfen. Ein Datensatz kann praktisch beliebig viele Spalten haben (daher der Name Wide Column Store), und zwei Datensätze können auch unterschiedliche Spalten haben. Wide Column Stores unterstützen die relativ einfache Verteilung der Daten auf große, auch geographisch getrennte Serverfarmen, welche sehr flexibel bei sich änderndem Bedarf umkonfiguriert werden können. Google beweist, dass man mit Wide Column Stores auch sehr komplexe Applikationen implementieren kann, schließlich verwenden zahlreiche Google Produkte wie Web Indexing, Gmail, Google Maps oder YouTube diese Plattform.

Während BigTable ein reines Google-internes Produkt ist, gibt es inzwischen etliche davon inspirierte Open Source Implementierungen, etwa **Cassandra**, das ursprünglich von Facebook entwickelt wurde, oder **HBase**, ebenso wie Cassandra jetzt ein Apache Top Level Projekt.

Document Stores

Einen etwas anderen Ansatz verfolgen Document Stores. Die Grundidee dabei ist, dass Daten, welche man in einem Datenbestand erfassen will, in Wirklichkeit oft keine einheitliche Struktur haben. Beispielsweise haben Artikel in einem Warenhaus ganz unterschiedliche Attribute, etwa ein Notebook und ein Mikrowellenherd. Üblicherweise müssen diese Daten aber in ein durch ein RDBMS erzwungenes Schema gepresst werden. Das ist zwar möglich, und Generationen von Entwicklern haben diese als Datenmodellierung bezeichnete Herangehensweise so gelernt, trotzdem ist das häufig keine natürliche, und damit auch keine effiziente Abbildung. Wenn die damit verbundenen Nachteile als notwendige Tatsache hingenommen werden, hat das eher mit eingefahrenen Denkmustern zu tun als mit technischen Notwendigkeiten.

Document Stores erlauben nicht nur, ähnlich wie Wide Column Stores, eine schemafreie Modellierung, und damit unterschiedliche Spalten pro Datensatz. Sie unterstützen auch weitergehende Freiheiten wie Arrays oder verschachtelte Objekte. Der bekannteste Document Store, **MongoDB**, akzeptiert etwa beliebige JSON Strukturen als Datensätze. Im Idealfall werden solche JSON-Objekte dann auch in der Applikation als solche behandelt, so dass von der Datenbank bis zum User Interface ein einheitlicher Mechanismus für den Umgang mit äußerst flexiblen Strukturen verwendet werden kann.

Weitere NoSQL Varianten

Die oben beschriebenen drei Varianten sind diejenigen, die am häufigsten anzutreffen sind, und am meisten diskutiert werden. Es gibt jedoch noch zahlreiche weitere Typen wie Graphdatenbanken, RDF Stores oder Native XML Stores. Manche Systeme können auch nicht immer eindeutig klassifiziert werden.

NewSQL

Wer sich vom relationalen Modell doch nicht ganz trennen kann oder will, findet trotzdem neue, innovative Ansätze. Die Vertreter dieser Klasse sehen sich nun nicht als NoSQL Systeme, um sich aber dennoch begrifflich von den klassischen Produkten abzugrenzen, haben sie den Namen NewSQL geprägt. Ein interessanter Vertreter ist **VoltDB**. VoltDB ist eine In-Memory Datenbank, die ähnlich wie Redis bei Bedarf auch volle Persistenz anbietet. Der innovative Ansatz ist aber ein anderer: ausgehend von Messungen, die zeigen, dass herkömmliche RDBMS einen guten Teil der Laufzeit mit Tätigkeiten zum Absichern von Transaktionen benötigen anstatt mit der eigentlichen Datenverwaltung, hat VoltDB die Implementierung von Transaktionen radikal geändert. In herkömmlichen RDBMS werden die einzelnen Verarbeitungsschritte einer Transaktion vom Client gesteuert. Dabei werden vom Server immer wieder Daten angefordert beziehungsweise Datenänderungen zum Server geschickt, währen die eigentliche Logik client-seitig abläuft. Der Datenbankserver muss aber, um die Transaktionssicherheit entsprechend der ACID Kriterien zu erfüllen, diese Schritte vor anderen, parallel ablaufenden Applikationen vorerst verstecken und mit aufwändigen Mechanismen sicherstellen, dass jeder Client einen konsistenten Stand von Daten sieht, der zwischenzeitlich aber für jeden Client unterschiedlich sein kann. Im Falle von Transaktionsabbrüchen oder Systemabstürzen muss wieder ein früherer konsistenter Datenstand hergestellt werden. Es ist leicht einzusehen dass all dies einiges an Overhead erfordert. VoltDB fordert stattdessen, dass alle Transaktionen am Server in einer Stored Procedure ablaufen. Der Server arbeitet single-threaded, es gibt keine parallel durchgeführten Transaktionen. Eine Sperrverwaltung ist damit nicht notwendig, und die Mechanismen zur Erhaltung der Konsistenz sind wesentlich einfacher. Am Ende einer Stored Procedure werden geänderte Daten nach einem Alles-oder-Nichts Prinzip entweder als Gesamtheit abgespeichert oder eben nicht. Das Datenbankmanagementsystem kann dadurch wesentlich kleiner und einfacher werden, und dies bringt wie so oft eine höhere Performanz. Die Einschränkung des Ablaufes von Transaktionen am Server ist natürlich gravierend. VoltDB ist somit auch kein genereller Ersatz für klassische RDBMS. In typischen OLTP Anwendungen mit kurzen und wenig komplexen Verarbeitungsschritten kann aber durchaus ein Vielfaches von Datenbankoperationen pro Sekunde erreicht werden, und das ohne auf SQL oder ACID verzichten zu müssen.

Systemauswahl

Durch die Vielfalt an Datenbanksystemen ist die Systemauswahl wesentlich schwieriger geworden. Die Entscheidung sollte eher für ein klassisches relationales DBMS erfolgen, wenn die folgenden Kriterien sehr wesentlich sind:

- **Reife des Systems:** Die klassischen Datenbankprodukte sind seit über drei Jahrzehnten am Markt und laufen dementsprechend stabil. Der Umgang mit ihnen ist sowohl in den Entwicklungsabteilungen als auch in den Betriebsabteilungen gelebte Praxis. Die großen Hersteller dieser Produkte garantieren auch eine gewisse Investitionssicherheit. Obwohl es auch für die NoSQL- und NewSQL-Produkte entsprechende Referenzen gibt, ist deren Reifegrad im Allgemeinen noch wesentlich geringer und der langfristige Support dieser Systeme unsicherer. Auch gibt es für Betrieb und Administration der klassischen Systeme eine Vielzahl unterschiedlicher Tools, die nur in Ausnahmefällen auch NoSQL-Produkte unterstützen.

- **Kompatibilität:** Durch Verwendung von SQL und anderer standardisierter Schnittstellen und vor allem durch das einheitliche relationale Datenmodell ist es für Anwendungen nahezu transparent, welches der relationalen Datenbanksysteme letztlich verwendet wird. Bei den NoSQL-Systemen hat jedes Produkt seine eigenen Schnittstellen, sodass ein Produktwechsel bei den versorgten Anwendungen zu einem großen Umstellungsaufwand führen kann.
- **Konsistenzerfordernisse**. Beim Einsatz eines RDBMS wird das Absichern der Datenkonsistenz, wie etwa Transaktionssicherheit oder Fremdschlüsselbeziehungen, in einem hohen Ausmaß von der Datenbank selbst unterstützt. Dadurch wird der Applikationscode von so mancher Bürde befreit und die Anwendungen werden robuster. Diese Möglichkeiten stehen bei NoSQL Systemen nicht oder nur eingeschränkt zur Verfügung, und die Verantwortung für die Konsistenz fällt verstärkt in die Applikationen. Dies erhöht einerseits die Komplexität der Anwendungen, kann aber andererseits, bei weniger strikten Konsistenzerfordernissen, zu großen Performanzgewinnen führen.
- **Mächtigkeit der Abfragen**. SQL Programmierer sind sehr verwöhnt. Für komplexe Abfragen fasst man in SQL verschiedene Tabellen zusammen, dabei filtert, gruppiert und sortiert man fast nach Belieben. Der Applikationscode wird dann oft sehr einfach, z.B. wird das Ergebnis nur noch für die Anzeige formatiert. Bei NoSQL Datenbanken müssen viele dieser Datenmanipulationen in der Applikation ausprogrammiert werden.

Die Anwendungen, bei denen hingegen NoSQL Systeme oft besser geeignet sind, werden mit den „3 Vs“ charakterisiert:

- **Volume**, also Anwendungen wo große Datenmengen eine Rolle spielen,
- **Velocity**, hohe Verarbeitungsgeschwindigkeiten, und
- **Variety**, hohe Vielfalt der Datenstrukturen.

Durch ihre Einfachheit tendieren NoSQL Systeme auch dazu, weniger Aufwand im Betrieb der Datenbanken zu verursachen, wobei das manchmal proklamierte „Ende des Datenbank-administrators“ wohl noch ein unerreichtes Wunschdenken bleibt.

Das Beste aus beiden Welten erreicht man oft mit einem **hybriden Ansatz**, bei dem zwei oder mehr Systeme zum Einsatz kommen und die Daten geeignet aufgeteilt werden. Z.B. könnte man bei einem Online Shop die Daten, bei denen strikte Konsistenz erforderlich ist, also Daten über Kunden, Bestellungen und Lieferungen, in einem RDBMS gespeichert werden, während Daten, bei denen hohen Datenmengen rasch verarbeitet werden müssen, etwa Informationen über Clicks auf einzelne Links bei Besuchen auf der Shop-Site, in einer NoSQL Datenbank landen. Auch bei den meisten NoSQL Pionieren wie Amazon, Facebook oder Google findet man nach wie vor auch viele Bereiche in denen RDBMS eingesetzt werden.

Neben den technischen Argumenten darf man bei der Systemauswahl aber auch nie betriebswirtschaftliche Aspekte vernachlässigen. Gerade bei einem sich rasch ändernden technischen Umfeld entstehen oft hohe zusätzliche Initialkosten, etwa für Mitarbeiterschulungen und Pilotprojekte. Wenn dann das ausgewählte Produkt sich am Markt nicht durchsetzt, dann sind nicht nur diese Kosten verloren, es droht auch die Notwendigkeit eines aufwändigen Systemumstieges.

Fazit

Der Datenbankmarkt ist im Umbruch, die Meinungen über die NoSQL Bewegung gehen auseinander und werden oft emotional diskutiert. Dabei sollte man neue Angebote einfach als Bereicherung sehen, die ihre Stärken und Schwächen haben. Die kontrahierenden Lager bewegen sich auch durchaus aufeinander zu. So hat etwa Google, das mit der Publikation des intern verwendeten Datenbanksystems **BigTable** die NoSQL Bewegung maßgeblich beeinflusst hat, kürzlich mit **Spanner** eine neues internes System präsentiert, welches versucht einige Konzepte der relationalen Modelle auf die Big Data Anforderungen zu transponieren. Umgekehrt hat Oracle, der wohl bedeutendste RDBMS Hersteller, noch Anfang 2011 in einem vielbeachteten Bericht NoSQL Systeme gegenüber klassischen RDBMS in einem sehr schlechten Licht dargestellt. Mittlerweile bietet allerdings Oracle selbst mit **Oracle NoSQL** einen eigenen Key-Value Store an, und der Bericht ist plötzlich von der Oracle Website verschwunden.

Das Popularitätsranking von DB-Engines (http://db-engines.com/de/ranking) zeigt, dass nach wie vor die wichtigsten relationalen Datenbanksysteme die größte Verbreitung haben. Dahinter aber drängen eine Unzahl an alternativen Systemen nach vorne, welche zum Teil durchaus schon eine beachtliche Anerkennung erreicht haben.

Wer mit seinen Datenbanktechnologien kein Problem hat und rundum glücklich ist, für den spricht wenig für eine Änderung. Wer hingegen den Eindruck hat, dass er relativ viel Geld für die Erstellung, die Wartung und den Betrieb von Datenbankapplikationen ausgibt, oder wer immer wieder Projektideen verwerfen muss, weil die Entwicklungs- und Betriebskosten zu hoch wären, der sollte sein Augenmerk auf Alternativen werfen, bevor er mit der Produktivität der Konkurrenz nicht mehr mithalten kann.

Autoren

Paul Andlinger und Matthias Gelbmann sind die Geschäftsführer der solid IT gmbh, einem auf Datenbanken spezialisierten IT Beratungsunternehmen in Wien.

consulting & software development gmbh