

Datenflut bereitet NoSQL den Weg

Unter dem Begriff Not only SQL entsteht eine neue Generation von Datenbanken, die sich dem relationalen Modell entgegenstellen. Hier eine Beschreibung der wichtigsten Vertreter dieser Technik.

Von Kai Spichale, Thomas Westphal und Eberhard Wolff*

Genau genommen führt die Bezeichnung NoSQL als Abgrenzung von SQL zu Missverständnissen. NoSQL steht nicht für „kein SQL“, sondern für „Not only SQL“, also nicht nur SQL. Dadurch ist der Begriff zwar einprägsam, aber unklar. Unter „Not only SQL“ fällt wohl letztlich jeder Mechanismus für die Verwaltung von Daten.

Die NoSQL-Bewegung ist eine Reaktion auf die Herausforderungen, vor denen das Daten-Management in den nächsten Jahren steht:

- Die Datenmengen wachsen exponentiell.
- Daten sind immer stärker vernetzt. Dazu gehören der Hypertext im Web, Blog-Inhalte oder User-generated Content. Da viele Anwender immer und überall online sind, produzieren sie, beispielsweise in Social Networks, immer mehr Daten selbst, statt sie nur abzurufen.
- Die Daten sind immer weniger strukturiert. Konnten sie früher in Tabellen repräsentiert werden, fällt dies heute zum Beispiel bei User-generated Content zunehmend schwerer. Auch bei der Suche nutzt man jetzt Text-Indizes und nicht mehr bestimmte Werte in Tabellenspalten.
- Die Integration durch eine gemeinsame Datenbank nimmt ab. Stattdessen setzen sich Techniken wie Web-Services für die

Integration durch. Jedes System hat seine eigene Datenbank.

- Durch die Cloud werden noch mehr Daten zentral gehalten, Systeme müssen daher stärker skalieren. Dafür sollten primär viele weniger leistungsstarke Rechner genutzt werden, die in der Cloud typischerweise zur Verfügung stehen. Klassisch nutzt man aber gerade für relationale Datenbanken eher wenige und dafür leistungsstarke Maschinen.
- Durch die Verfügbarkeit von kostengünstigen Hochgeschwindigkeits- sowie mobilen Internet-Zugängen werden zunehmend hohe Anforderungen in Bezug auf Datendurchsatz und Hochverfügbarkeit an Internet-Plattformen gestellt.

Keine vollständige Ablösung

Aufgrund dieser Herausforderungen gibt es genügend Bereiche, die durch neue Techniken abgedeckt werden können, ohne dass relationale Datenbanken dort unbedingt abgelöst werden müssten. Da häufig strategische Entscheidungen zugunsten bestimmter relationaler Datenbanken getroffen werden, ist ihre vollständige Ablösung ohnehin eher unwahrscheinlich.

Es gibt völlig unterschiedliche Konzepte, die unter der Bezeichnung NoSQL beworben und vermarktet werden. Die wichtigsten sind:

- „Key/Value Stores“ speichern unter einem Schlüssel einen einzigen Wert ab. Dieses Modell ist ausgesprochen einfach. Es hat den Vorteil, dass die Daten sehr leicht auf mehrere Rechner verteilt werden können. Der Nachteil ist allerdings, dass man komplexere Datenstrukturen kaum sinnvoll abbilden kann.
 - „Document Stores“ können mit Dokumenten umgehen. Die Daten müssen keinem vordefinierten Schema entsprechen, aber Anfragen nach Datensätzen mit bestimmten Eigenschaften werden unterstützt. Diese Datenbanken sind vor allem für wenig strukturierte Daten nutzbar und von einem objektorientierten System aus auch einfach zu verwenden.
 - „Wide-Column“-Ansätze bieten die Möglichkeit, in einer Zeile viele Daten zu speichern. Dabei können auch neue Spalten eingeführt werden.
 - „Graphendatenbanken“ speichern Knoten und Beziehungen zwischen Knoten. Sie sind nützlich, um beispielsweise die Beziehungen in einem sozialen Netzwerk abzuspeichern, und können Anfragen wie „Wie viele Bekannte zweiten Grades hat der Nutzer?“ leicht bearbeiten.
- Für diese verschiedenen Typen von NoSQL-Datenbanken sollen nun einige Beispiele genauer betrachtet werden. ▶

Redis

► Redis ist ein in C geschriebenes Open-Source-Projekt mit einer liberalen Lizenz und unterstützt die meisten gängigen Plattformen. Es läuft als Server-Prozess mit einem sehr einfachen Protokoll. Bibliotheken zum Ansprechen des Servers gibt es für zahlreiche Programmiersprachen. Redis fällt in die Kategorie der Key-Value-Stores, wobei die Werte auch komplexere Typen wie Listen, sortierte oder unsortierte Mengen oder auch Maps sein können. Dadurch sind auch komplexere Strukturen abbildbar.

Auf Schnelligkeit getrimmt

Außerdem eröffnen sich weitere Möglichkeiten für die Lösung typischer Probleme bei der Implementierung großer Systeme: So kann eine Liste auch sehr einfach eine Warteschlange implementieren. Man legt die Aufgaben in die Liste, und die Bearbeiter holen sich jeweils das oberste Element.

Redis ist vor allem auf hohe Geschwindigkeit ausgerichtet: Die Daten werden meistens im Speicher gehalten und asynchron auf die Platte geschrieben oder auch

Redis

Vorteile:

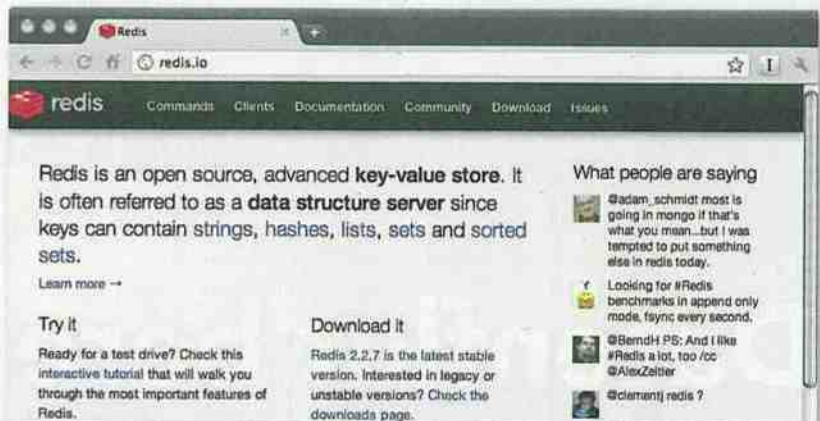
- Hohe Performance durch Halten der Daten im Speicher und asynchrones Schreiben auf die Festplatte;
- nützliche Datentypen für verteilte Algorithmen.

Nachteile:

- Keine Unterstützung für komplexe Datentypen;
- ohne Clustering eher für kleine Datenmengen geeignet.

auf andere Server repliziert. Dadurch erreicht Redis eine sehr hohe Performance: Beim Lesen lassen sich die Daten direkt aus dem RAM auslesen, und beim Schreiben blockiert der Zugriff auf die Festplatte das System nicht, weil er asynchron im Hintergrund ausgeführt wird. Durch Replikation lässt sich der Lese-Durchsatz weiter erhöhen, indem zusätzliche Server für Lese-Operationen zur Verfügung gestellt werden.

Wenn so viele Daten zu verwalten sind, dass sie nicht mehr in den Speicher passen, kann eine Redis-eigene Virtual-Memory-Verwaltung die Daten auch auf die Platte auslagern. Darüber hinaus ist es möglich, ähnlich wie in einem Cache, die Daten mit einer Lebensdauer (Time to Live = TTL) auszustatten, nach der sie aus der Datenbank verschwinden.



Die Redis-Website: Das in C geschriebene Open-Source-Projekt verwendet als NoSQL-Verfahren einen Key-Value-Store und ist auf hohe Geschwindigkeit eingerichtet.

Redis arbeitet mit Master-Slave-Replikation. Das heißt, es gibt einen Master-Server für Lese- und Schreiboperationen und beliebig viele Slave-Server. Diese können lediglich Leseoperationen abarbeiten und werden ständig mit dem Master synchronisiert. Bei einem Failover übernimmt einer der Slave-Server die Rolle des ausgefallenen Masters. Aufgrund dieser Master-Slave-Rollenaufteilung kann Redis starke Konsistenz garantieren.

Sharding soll folgen

Eine Cluster-Funktion mit Sharding ist für Redis geplant. Bei Sharding handelt es sich um ein Verfahren zur Aufteilung von großen Datenmengen auf mehrere Datenbank-Server. Ein Beispiel sind Kunden, die je nach Region auf einem anderen Server gespeichert oder nach Kundennummer aufgeteilt werden. Durch Sharding kann es mehrere Master geben, die für jeweils einen Teil der Daten verantwortlich sind. Außerdem wird an der Möglichkeit gearbeitet, in der Datenbank Skripte in der Sprache Lua auf die Daten anzuwenden. Unter <http://try.redis-db.com> besteht die Möglichkeit, eigene Erfahrungen mit der Datenbank ohne Installation zu sammeln.

Apache Cassandra

2007 begann Facebook mit der Entwicklung einer proprietären Lösung zum Speichern und Suchen von Benutzernachrichten für die Inbox-Suche. Das war die Geburtsstunde von Cassandra, das inzwischen zu den Top-Level-Projekten der Apache Software Foundation gehört. Die neue Datenbank sollte auf kostengünstigen Commodity-Servern laufen und inkrementell skalierbar sein, um dem wachsenden User-generated Content langfristig Herr zu werden.

Cassandra gehört zur Familie der Wide Column Stores und lässt sich als hochskalierbarer, schlussendlich konsistenter, verteilter und strukturierter Schlüssel-Wert-Speicher beschreiben. Hochskalierbar bedeutet, dass ein Cassandra-Cluster durch Hinzufügen weiterer Server horizontal skaliert werden kann. Auf diese Weise sind riesige Datenmengen auch mit Commodity-Hardware verwaltbar. Hochverfügbarkeit wird durch Replikation und Verteilung der Daten auf mehreren Servern garantiert. Dabei kann der Ort für die Replikate so gewählt werden, dass diese sich in verschiedenen Rechenzentren befinden. Schlussendlich konsistent (eventually consistent) bedeutet, dass innerhalb eines begrenzten Zeitfensters nicht immer alle User die gleiche Sicht auf die Daten haben. Dadurch kann eine bessere Performance gewährleistet werden.

Cassandra unterstützt verschiedene Konsistenzstufen für Lese- und Schreibopera-

Apache Cassandra

Vorteile:

- Gute Performance durch horizontale Skalierbarkeit und Verwendung von kostengünstiger Commodity-Hardware;
- garantierte Hochverfügbarkeit durch Verwaltung von Replikaten auf verschiedenen Servern und Rechenzentren;
- flexibles schemaloses Datenmodell.

Nachteile:

- Unterstützt verschiedene Konsistenzstufen, aber keine Transaktionen mit ACID-Eigenschaften;
- eingeschränkte Datenabfrage soll erst ab Version 0.8 durch die Cassandra Query Language verbessert werden.

tionen, um den Spagat zwischen Konsistenz und Performance für den jeweiligen Anwendungsfall zu meistern. Das System wurde in Java entwickelt und kann daher auf praktisch allen Plattformen eingesetzt werden. Client-Bibliotheken auch für andere Programmiersprachen wie Python, PHP und C# sind vorhanden. Die Daten werden in Form von mehrfach ineinandergeschichteten Schlüssel-Wert-Paaren gespeichert. Das Datenmodell ist strukturiert, aber schemalos und daher sehr flexibel. Durch die Integration der Apache-Hadoop-Technik unterstützt Cassandra ebenfalls Map-Reduce zur nebenläufigen Verarbeitung großer Datenmengen im Computer-Cluster.

Mehrfach praxiserprobt

Cassandra verwendet wie Redis Sharding und Replikate, aber es gibt keine Unterscheidung zwischen Master- und Slave-Servern. Mit diesem Ansatz wird die Schreib-Performance verbessert, starke Konsistenz kann jedoch nicht garantiert werden. Während Redis auf Konfliktvermeidung ausgelegt ist, arbeitet Cassandra mit Mechanismen zur Konfliktauflösung.

Cassandra wird von Unternehmen wie Facebook, Twitter, Cloudflick und Rackspace erfolgreich eingesetzt und stellt damit seine Zuverlässigkeit und Reife unter Beweis. Weitere Informationen zu Cassandra gibt es unter <http://cassandra.apache.org>.

CouchDB

CouchDB steht für „Cluster of unreliable Commodity Hardware Database“. Bei diesem seit 2005 laufenden Projekt handelt es sich um einen Document Store. Der Vorteil dieser Datenbank ist, dass Dokumente sowohl dem natürlichen Objekt als auch dessen Repräsentation in einer objektorientierten Sprache entsprechen können. Ferner muss der Aufbau der Dokumente vor der Speicherung nicht bekannt sein, denn es handelt sich bei CouchDB um eine schemalose Datenbank.

Keine Schreib-Lese-Blockaden

Die Dokumente werden im JSON-Format verwaltet. Dabei versieht die Datenbank jedes JSON-Objekt mit einem eindeutigen Schlüssel und mit einer Versionsnummer. Mit Hilfe der Versionsnummer verwaltet CouchDB die konkurrierenden Zugriffe. Dazu wird die Versionsnummer bei jeder Änderung eines Dokuments erhöht. CouchDB erkennt beim Schreiben die Konflikte und meldet sie dem jeweiligen Client. Die bei anderen Systemen durch Locking entstehenden Schreib-Lese-Blockaden treten somit nicht auf. Dieses System,



Die Futon-Web-Oberfläche dient zur Verwaltung von CouchDB-Datenbanken – das Logo verspricht eine entspannte Datenverwaltung.

Multi-Version Concurrency Control genannt, wird auch bei der Replikation verwendet, um Konflikte zu entdecken und zu beheben.

Um auf Dokumente nicht nur anhand des Schlüssels zuzugreifen, verbindet CouchDB den dokumentenorientierten Ansatz mit der von Apache Hadoop bekannten MapReduce-Technik. Diese Abfragen werden bei CouchDB als „View“ bezeichnet. Eine Abfrage stellt dabei eine Sicht auf die Daten dar und verändert diese nicht. Sie besteht aus einer Map- und aus einer optionalen Reduce-Funktion. Beide werden in Javascript beschrieben und können in CouchDB hinterlegt werden. Permanente Views werden, im Gegensatz zu temporären Views, als Design-Dokumente bezeichnet, die von CouchDB indiziert werden. Ein performanter Zugriff ist somit gewährleistet. Das Ergebnis der Views sind Schlüssel-Werte-Paare. Dabei besteht sowohl der Schlüssel als auch der Wert aus einem JSON-Dokument.

CouchDB

Vorteile:

- Flexibel, da die Dokumente keinem Schema entsprechen müssen;
- Dokumente entsprechen sehr gut natürlichen Objekten: einfaches Design, kein objektrelationales Mapping und keine Normalisierung;
- einfache Bedienung: einfache Web-Schnittstellen und bekannte Techniken wie HTTP-REST, JSON und Javascript;
- einfacher bidirektionaler Replikationsmechanismus;
- skaliert sowohl nach oben (Cluster) als auch nach unten (Smartphone).

Nachteile:

- Genügt keinen hohen Konsistenzanforderungen.

Der Zugriff auf Daten erfolgt über eine HTTP-REST-Schnittstelle. Für viele Programmiersprachen sind zusätzlich Bibliotheken verfügbar. Die Administration erfolgt per HTTP-REST oder über die Web-Oberfläche Futon. Mit dem in CouchDB integrierten Javascript-Web-Framework CouchApp lassen sich Web-Anwendungen erstellen, die direkt in der CouchDB laufen. Ein zusätzlicher Web-Server entfällt dabei.

Schwach ausgeprägte Konsistenz

CouchDB verfügt über einen inkrementellen Replikationsmechanismus mit schwacher Konsistenz und bidirektionaler Konfliktbehandlung: Konflikte durch Änderungen an einem Objekt auf verschiedenen Servern werden erkannt und behandelt. Dabei bleiben alle Versionen erhalten. Die Replikation lässt sich sehr einfach für fortlaufende Replikation einrichten oder ad hoc verwenden. Horizontale Skalierung durch Sharding beherrscht CouchDB nicht. Nur durch zusätzliche Produkte, die als Proxy zwischen Client und Datenbank arbeiten, kann Sharding realisiert werden. Das Open-Source-Projekt BigCouch bietet ebenfalls eine skalierbare Lösung für CouchDB an.

Die Datenbank ist zum größten Teil in der Programmiersprache Erlang geschrieben und für die meisten Plattformen verfügbar. Sogar auf einem Android-Smartphone oder auf einem iPhone kann eine CouchDB-Instanz betrieben werden. Auch Cloud-Angebote wie Cloudant und Iris Couch lassen sich nutzen.

CouchDB als Download sowie weitere Informationen zum Thema bieten die Apache Software Foundation (<http://couchdb.apache.org>) und die Firma Couchbase Inc. (<http://www.couchbase.com>). (ue)

*Kai Spichale und Thomas Westphal sind Senior Software Engineers, Eberhard Wolff ist Architecture and Technology Manager bei der Adesso AG in Dortmund.