Aktuelle Entwicklungen im Bereich post-relationaler Open-Source-Datenbanken

Freischwebend

Eine Reihe neuer Produkte schickt sich an, nicht nur den relationalen Datenbanken sondern auch den Cloud-Vorreitern Google und Amazon Marktanteile streitig zu machen. **Mathias Meyer**

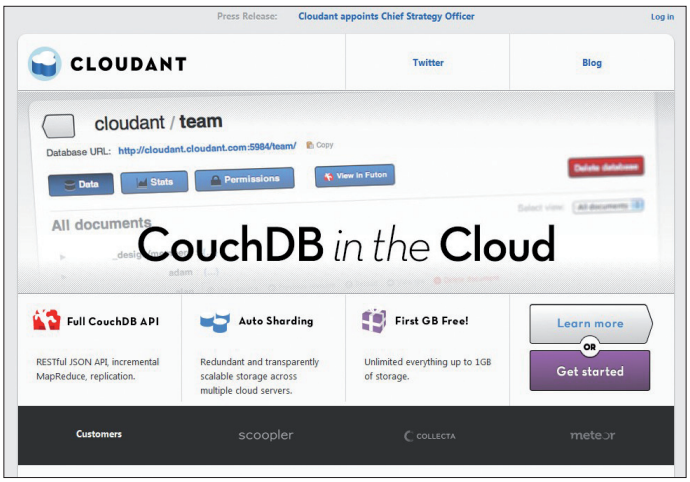
Auf einen Blick

Google und Amazon sind nicht nur Vorreiter in der Cloud, sie sind Inspiration für eine ganze Reihe neuer Datenbanken, die den Skalierungsansprüchen der Cloud auf unterschiedlichste Weise gerecht werden wollen und dabei die Offenheit der Daten bewusst im Vordergrund lassen..

Plattform
unabhängig

Autor
Mathias Meyer arbeitet bei Peritor in Berlin, wo er sich Consulting und Entwicklung mit Ruby on Rails und Amazon Web Services widmet. Er probiert gern neue Technologien in den Bereichen von Web, post-relationaler Datenbanken und Cloud-Computing aus, und feilt an Scalarium (http://scalarium.com), einer eleganten Lösung für Cloud-Management und -Deployment.

Die Cloud bringt nicht nur neue Herausforderungen bei der Applikationsentwicklung mit sich, sie erfordert Umdenken in der Art wie Daten behandelt und gespeichert werden. Traditionelle, relationale Datenbanken wollen nicht so richtig in das Modell einer sich stetig im Fluss befindenden Wolkenlandschaft passen. Sobald neue Computing-Ressourcen verfügbar werden, sollte sich die existierende Datenbank-Landschaft möglichst nahtlos und selbständig ausbalancieren und integrieren. Daten sollten so effizient wie möglich auf den neuen Instanzen verfügbar sein, am besten ohne große Interaktion durch den Betreiber.



Cloudant bietet Hosting-Angebote für CouchDB – das erste GByte Datenspeicher ist kostenlos (Bild 1)

Google und Amazon haben mit ihren Tools eine Steilvorlage für eine völlig neue Generation post-relationaler Datenbanken gegeben, die nicht nur den Ballast eines festen Schemas ablegen, sondern auch neue Wege beschreiten um Skalierbarkeit und Verteilung sowohl horizontal als auch vertikal zu ermöglichen.

Als kommerzielle Anbieter betreiben Google und Amazon ihre interessantesten Tools größtenteils intern. Googles BigTable [1] wird zum Beispiel in Produkten wie Google Reader, Google Earth, Google Code und noch einigen weiteren Anwendungen eingesetzt, steht aber auch zur Benutzung über AppEngine zur Verfügung. Amazons Dynamo [2] kommt zum Beispiel dann zum Zuge, wenn man ein Produkt in den Warenkorb legt.

+1

CouchDB – Bewährtes in neuem Gewand

Sowohl das Modell von Google als auch das von Amazon sind seit Jahren im Einsatz und gehen auf bereits länger bewährte Prinzipien und Algorithmen zurück. Nicht minder alt ist das Prinzip auf dem CouchDB [3] basiert, es reicht zurück bis in die Zeiten von Lotus Notes.

CouchDB ist eine dokumentenorientierte Datenbank, inspiriert von Lotus Notes. Schemafreiheit ist nur ein Grundprinzip was sich auch heute noch in Form des kompakten Text-Datenformats JSON (JavaScript Object Notation) widerspiegelt. CouchDB setzt durchweg auf Webtechnologien, verwendet HTTP als Protokoll, JSON für Dokumente und JavaScript zur Abfrage und Aggregation von Daten über MapReduce.

Die Details von CouchDB wurden bereits ausführlich in einem früheren Artikel behandelt [4], das für Cloud-Computing interessante Feature ist aber die Replikation, die in CouchDB einzigartig umgesetzt und ohne Zweifel eines der

Column-Stores

Google BigTable [1] beschreitet den Weg eines Column-Stores. Hier werden Daten nicht in einzelnen Zeilen abgelegt, sondern nach Spalten. Das Konzept basiert auf der Annahme, dass zusammengehörende Daten auch so nah wie möglich beieinander gespeichert werden sollten. Das Nachschlagen von Daten (Lookup) erfolgt so nur noch mit Hilfe eines Schlüssels, der aus der ID des Datensatzes und dem Namen des Attributes besteht. Zusätzlich wird jede Spalte mit einem Zeitstempel versehen, um Änderungen verfolgen und über verteilte Knoten (Nodes) abgleichen zu können. Daten lassen sich so nicht nur besser über viele Knoten verteilen, der Zugriff auf bestimmte Attribute erfolgt direkt, ohne den Umweg über Identifier, Zeile und dann Spalte.

Amazons Dynamo

Auch wenn Amazon mit SimpleDB eine interessante Cloud-Datenbank anbietet, versteckt sich das eigentliche Filetstück doch intern. Nach außen hin bekannt ist der Name Dynamo und ein sehr umfassendes Dokument zu dessen Funktionsweise [2]. Auch wenn ein simpler Key-Value-Store dahinter steckt, behandelt der interessanteste Teil des Dokuments die effiziente Verfügbarkeit, Partitionierung und Replikation von Daten und wurde so zur Vorlage für einige Nachahmer wie Riak, Cassandra und Dynomite.

wichtigsten Features ist [5].

Grundlage für Umsetzung der Replikation in CouchDB ist die Annahme, dass Knoten (Nodes) in einem Netzwerk von Datenbankservern kommen und gehen beziehungsweise zeitweilig voneinander getrennt sein können (zum Beispiel durch Netzwerkausfall). Während einige andere Tools den Weg von Dynamo beschreiten, um Ausfälle automatisch zu kompensieren, kann auch eine alleinstehende CouchDB-Instanz weiterarbeiten und seine Änderungen später an die anderen Knoten übertragen. Das Konzept heißt „Offline by Default“ und hat seinen Ursprung wie Dokumente in Lotus Notes. Ein Client konnte hier problemlos für unbestimmte Zeit offline sein und sich später nach einem ähnlichen Prinzip wieder mit dem Server synchronisieren.

Replikation Erster Klasse

Jede CouchDB-Datenbank kann sich mit jeder anderen replizieren. Es spielt keine Rolle, ob beide Instanzen völlig unterschiedliche, nur ähnliche oder gleiche Daten haben. Basis hierfür ist eine Revision für jedes Dokument, die sowohl beim Update von Dokumenten als auch bei der Replikation eine wichtige Rolle spielt.

Ein Dokument in CouchDB ist nicht mehr als eine längere JSON-Zeichenkette, ein Beispiel dafür ist in Listing 1 sehen.

Attribute, die mit Unterstrich beginnen, sind reserviert. Neben *_id* ist *_rev* interessant, denn hier wird die Revision eines Dokumentes gespeichert, die aus einem Inkrement und einer langen Zeichenfolge, dem Unique-Identifier bestehen.

Wird ein Dokument aktualisiert, muss immer die Revision angegeben werden, auf die sich das Update bezieht. Existiert bereits eine neuere Version, wird das Update mit einem Fehler abgebrochen. Ansonsten wird der erste Teil der Revision inkrementiert, und der zweite Teil mit einem neuen Hash versehen. Bei normalen Updates spielt also nur das Inkrement eine Rolle.

Beide Teile der Revision werden bei der Replikation relevant. Wurde das gleiche Dokument basierend auf der gleichen Revision auf zwei Nodes geändert, ist ein Konflikt die Folge, und das Dokument wird mit einem Flag markiert.

Trotzdem muss sichergestellt sein, dass eine der beiden Versionen als aktuellste angesehen wird. Basis für diesen Algorithmus, der auf allen Instanzen zum gleichen Ergebnis führt, ist zum einen das Inkrement, und zum anderen der Hash. Ist das Inkrement von einer Seite höher, gewinnt das entsprechende Dokument. Sind die Inkrements gleich, entscheidet welche Revision alphanumerisch größer ist. *2-f5b508964242c82d169f15faa68acb4d* ist demnach höher als *2-9eb9b90274d81d0516280d66fc1f8666*.

Warum ist gerade dieses Feature von CouchDB so interessant für die Cloud? Die Replikation ermöglicht es, einfach und schnell neue Ressourcen zu bereits existierenden hinzuzufügen. Eine neue Instanz muss nur einmal von einer anderen replizieren und kann danach sofort Applikationen bedienen. Sollte eine Reihe von Instanzen zwischenzeitlich physikalisch oder durch Netzwerkprobleme zwischen verteilten Datacentern nicht verfügbar sein, können alle Instanzen unabhängig voneinander weiterarbeiten. CouchDBs Replikation kann die Daten effizient wieder zusammenführen.

Ein weiterer interessanter Aspekt an CouchDB im Zusammenhang mit Cloud-Computing ist der eingebaute Support, komplette Applikationen in der Datenbank zu betreiben. Mit sogenannten CouchApps ist es möglich Frondend und Backend komplett in JavaScript zu implementieren und direkt aus CouchDB auszuliefern. Muss eine Applikation auf Lastspitzen schnell reagieren können, ist es nicht mehr notwendig neue Application-Server und neue Datenbank-Server hinzuzufügen. Es reicht eine neue CouchDB-Instanz.

MongoDB – Das Beste aus zwei Welten

Während CouchDB auf hohe lokale Datenkonsistenz setzt, geht MongoDB [6] einen etwas anderen Weg, um sich als Cloud-Datenbank anzubieten. Es ist vor allem auf eines getrimmt: sehr hohen Schreib- und Lesedurchsatz. ▶

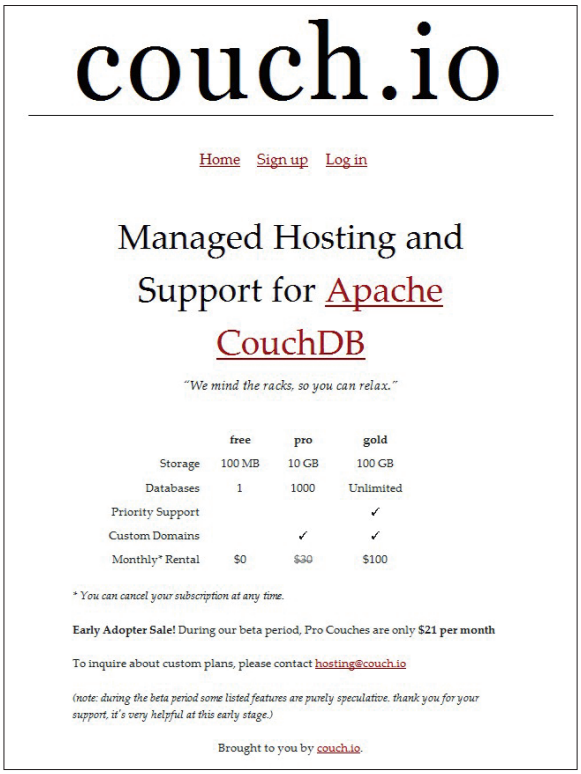
Listing 1: CouchDB-Beispieldokument.

```
{
  „_id“ :
    „533690773f350b7e1297a2579e058365“,
  „_rev“ :
    „1-9cb8de24b7b0caccd01ebb7cfbf39bc8“,
  „title“: „Dokument 1“,
  „body“ :
    „Sehr wichtige Informationen...“
}
```

Die Grundlagen sind denen von CouchDB ähnlich, Daten werden in Form von Dokumenten abgespeichert in einem JSON-ähnlichen Format namens BSON (binäres JSON). Es erweitert die Standard-Features von JSON um einige Datentypen wie zum Beispiel Datum und reguläre Ausdrücke. Auch das Protokoll von MongoDB folgt anderen Philosophien als CouchDB, es ist durchweg binär und von seinen Entwicklern konsequent auf Effizienz und hohen Durchsatz getrimmt.

CouchDB ist eine völlige Abkehr von relationalen Prinzipien, MongoDB stellt dagegen einen Kompromiss aus beiden Welten dar. Ähnliche Daten werden in Collections gespeichert, die sehr nah am Prinzip der relationalen Tabellen sind, und die Daten müssen für den schnellen Zugriff über Queries indiziert werden.

Diese Konstellation eröffnet zwar interessante Möglichkeiten, bringt aber durchaus auch spürbare Nachteile mit sich. MongoDB unterstützt atomare Operationen auf den Datensätzen und somit partielle Updates von einzelnen Attributen. Somit ist es einfach und sinnvoll, zusammengehörige Daten in Dokumente einzubetten anstatt sie in einer gesonderten Collection abzulegen und zu referenzieren. Erhält zum Beispiel ein Blog-Eintrag einen neuen Kommentar, kann dieser einfach auf die Liste der existierenden Kommentare geschoben werden, ohne dabei das gesamte Dokument zu aktualisieren. Noch dazu ist diese Operation dann atomar.



CouchDB-Hosting bei CouchIO (Bild 2)

Geschindigkeit vs. hohe Konsistenz

MongoDB verzichtet dabei auf ständig konsistente Daten. Es schreibt Änderungen aus dem Speicher nur alle 60 Sekunden auf die Festplatte. Da es sich um Memory-Mapped-Files handelt, ist auch das Schreiben der Daten nicht zu einhundert Prozent transaktional, es besteht ein gewisses Risiko, dass ein Crash während des Schreibvorgangs korrupte Daten hinterlässt.

Die fehlende Atomarität ist durchaus ein Ausschlusskriterium für so einige Anwendungsfälle, allerdings gibt es gerade im Web und in der Cloud genug Einsatzmöglichkeiten, bei denen hundertprozentige Datenkonsistenz nicht die höchste Priorität hat, hohe Geschwindigkeit allerdings sehr gefragt ist. Wenn es darum geht, Massendaten, wie Statistiken, Logging-Daten und dergleichen zu speichern, ist ein Verlustmoment von 60 Sekunden durchaus verträglich.

Die Speicherfrequenz ist konfigurierbar und hat einen klaren Vorteil: Lesen und Schreiben von Daten in und aus MongoDB ist extrem schnell. Schreiben neuer Daten bedeutet einfach nur Speicher-Allokierung für ein entsprechendes Objekt, Daten zu lesen ist nicht mehr als ein Ablaufen der Dokumente in einer Collection.

Konsistenz durch Replikation

MongoDB bietet keine Garantie für lokale Konsistenz. Als Alternative bietet sich Replikation an, hier unterstützt MongoDB das klassische Master-Slave-Prinzip. Slaves erhalten neue Daten vom Master potentiell mit leichter Verzögerung, können dafür aber für Backup-Snapshots verwendet werden, ohne den Master zu beeinträchtigen.

Ein interessantes Add-on zur Replikation sind Replica-Pairs, zwei Instanzen die zusammen ein Master-Slave-Paar bilden. Fällt der Master aus,

Warum eignet sich MongoDB so gut für die Cloud?

Diese Frage beantwortet Eliot Horowitz, CTO von 10gen, der Firma hinter MongoDB, wie folgt:

„Mit Cloud Computing denken Leute anders über Skalierung. Selbst wenn man will, ist es schwer in der Cloud vertikal zu skalieren, da die Möglichkeiten beschränkt sind. Horizontal zu skalieren ist die einzige Option, und MongoDBs Dokumenten-Modell bietet sich hier sehr gut an. Da man umfangreiche Dokumente erstellen kann, sind Joins nicht notwendig. Weil Joins überflüssig sind ist Sharding mit MongoDB möglich. Auto-Sharding unterstützt fast die gesamte Funktionalität von MongoDB, inklusive Sekundär-Indizes für Range-Queries und Sortierung, MapReduce, et cetera."

übernimmt der Slave dessen Rolle und gibt sie wieder ab, sobald der alte Master wieder online bereit steht.

Replica-Pairs sind eine wichtige Basis für MongoDBs Sharding [7], wobei es auch möglich ist, Shards aus einzelnen Nodes aufzubauen. Allerdings riskiert hier man hier Inkonsistenz der Daten, sollte ein Shard ausfallen.

Was macht MongoDB zur Cloud-Datenbank? Davon abgesehen, dass MongoDB ursprünglich Teil eines Webframeworks für Cloud-Comptuing war, ist MongoDBs große Stärke ohne Zweifel die hohe Lese- und Schreibgeschwindigkeit. Gerade in der virtuellen Welt von Cloud-Instanzen wie Amazons EC2 kann es durchaus von Vorteil sein, dass Daten nicht sofort geschrieben und Disk-I/O auf feste Abstände reduziert wird. Replica-Pairs können zusätzlich sicher stellen, dass Ausfälle automatisch abgefangen werden.

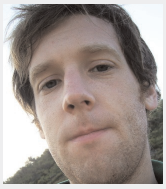
MongoDBs Sharding-Support befindet sich noch in der Entwicklung, macht es aber sehr interessant für umfangreichere Setups in der Cloud. Auch wenn sich die reduzierte Konsistenzgarantie abschreckend anhört, ist sie doch in vielen Situationen ertragbar, wobei der Abstand zwischen den einzelnen Schreibzugriffen konfigurierbar ist.

MongoDB bietet sich im Gegensatz zu CouchDB eher für lokale und nicht geografisch verteilte Anwendungen an, da es kein zuverlässiges Multi-Master-Setup unterstützt. Sollen Daten geografisch verteilt werden, ist CouchDB die bessere Wahl. Beide sind auf ihre Weise gut für die Cloud geeignet, weswegen sich bereits ein Ökosystem entwickelt hat, um sowohl CouchDB als auch MongoDB als Service verfügbar zu machen. Für CouchDB schicken sich Couchio [8] von den Machern von CouchDB und Cloudant [9] an, für MongoDB geht MongoHQ [10] ins Rennen (Bild 1 – 3).

Selbst ohne Nutzung eines Service ist ein Blick auf beide Datenbanken ganz sicher eine Bereicherung nicht nur des eigenen Toolkits, sondern auch weil sie Umdenken erfordern und andere beziehungsweise völlig neue Wege beschreiten.

[bl]

Warum eignet sich CouchDB so gut für die Cloud?



Jan Lehnardt, Apache CouchDB Entwickler und Mitgründer von Couchio beantwortet diese Frage so:
„Die Vorteile von Cloud-Umgebungen sind offensichtlich, doch gibt es eine Achilles-Ferse: Die Internetverbindung vom Benutzer zur Cloud. DSL ist zu Stoßzeiten oft langsam, Mobilfunkverbindungen sind langsam und instabil. Mit CouchDBs Replikation kann man Benutzern ihre Daten lokal und ohne Verzögerung verfügbar machen und so die Vorteile einer Cloud-Umgebung nutzen, ganz ohne von der „letzten Meile“ abhängig zu sein.

[1] Google BigTable, labs.google.com/papers/bigtable.html
[2] Amazon Dynamo, www.allthingsdistributed.com/2007/10/amazons_dynamo.html
[3] CouchDB, couchdb.apache.org
[4] Frank Plentka, Eine Datenbank für das Web, Einführung in die CouchDB-Datenbank, databasepro 5/2009, Seite 29 ff.
[5] CouchDB Wiki - Replication, wiki.apache.org/couchdb/Replication
[6] MongoDB, mongodb.com
[7] MongoDB Sharding Design, http://www.mongodb.org/display/DOCS/Sharding+Design
[8] couch.io Hosting, hosting.couch.io
[9] Cloudant, cloudant.com
[10] MongoHQ, mongohq.com

MonoHQ hostet MongoDB-Datenbanken (Bild 3)