DatabaseHypervisor – ein Ansatz für relationale Datenbanken in der Cloud

Benjamin Schmidt

Zimory GmbH
Stralauer Platz 33-34
10243 Berlin
schmidt@zimory.com

Abstract: Aktuelle Cloud-Produkte und Infrastrukturen haben mittlerweile eine breite Akzeptanz gefunden, wenn es um den Präsentations- und Applikations-Layer einer typischen 3-gliedrigen Web-Anwendung geht. Für den dritten Layer, der Datenbank, haben sich NoSQL oder Key-Value-Datenbanken wie z.B. Googles BigTable oder Amazons SimpleDB etabliert. Diese bieten die in der Cloud-Umgebung geforderte horizontale Skalierbarkeit, wenn auch mit einem Haken – sie haben ein nichtrelationales Design und orientieren sich deshalb nicht am SQL-Standard.

Bei traditionellen Enterprise-Anwendungen sind jedoch diese nicht-relationalen, verteilten Data Stores keine Option. Die Akzeptanz von Cloud Computing in die IT-Infrastruktur eines Großunternehmens mit großen Datenbankservern hat deshalb noch keinen nennenswerten Zustrom erhalten. Relationale Datenbanken sind bisher aus sowohl Virtualisierten-, wie auch Cloud-Umgebungen gehalten worden und sind immer noch zu sehr an einen einzigen Server (oder Cluster) und Ort gebunden.

Dieser Beitrag vergleicht bisherige Cloud-Datenbank-Implementierungen und stellt einen neuen Ansatz zur Datenbankvirtualisierung vor. Dieser Ansatz wird anhand der Architektur des Prototyps "Spree-DatabaseHypervisor" veranschaulicht.

1 Skalierung von Datenbanken

Die Architektur traditioneller Unternehmensapplikationen sind in der Regel in drei logischen Schichten unterteilt: Präsentation, Anwendung (oder Business Logic) und Data.

Auf der Präsentations- und Applikationsschicht ist eine horizontale Skalierung eine weit verbreitete Praxis, um eine Lastverteilung, erhöhte Ausfallsicherheit und größere Leistung zu gewährleisten. In horizontal skalierten Architekturen, hat jedes System seine eigenen, unabhängigen Ressourcen wie Prozessoren, Arbeitsspeicher und Storage. Diese

"shared nothing"-Architektur bietet eine höhere Verfügbarkeit da es keinen Single-Point-of-Failure (SPOF) enthält. Zur Erhöhung der Leistung bei horizontalen Architekturen werden einfach neue Knoten dem Cluster hinzugefügt.

Im Gegensatz dazu, wird bei der unteren Schicht, der Datenschicht, eher vertikal skaliert. SMP (Symmetrisches Multiprozessorsystem) Maschinen dominieren auf der Datenebene, bei der die Ressourcen, wie Prozessor, Speicher und Storage, geteilt werden. Zur Erhöhung der Verfügbarkeit von Anwendungen und um die Gefahr des SPOFs unter Verwendung von SMP-Systemen bei der Datenschicht zu mildern, wird oft eine von mehreren Datenreplikationsmethoden angewandt, um ein zusätzliches System für Failover-Zwecke zu erhalten. Um die Gesamtleistung zu steigern, müssen zusätzliche Ressourcen dem System hinzugefügt werden (Upgrade); ist das System nicht mehr Upgrade-fähig, muss ein größeres (und meistens teureres) System beschafft werden.

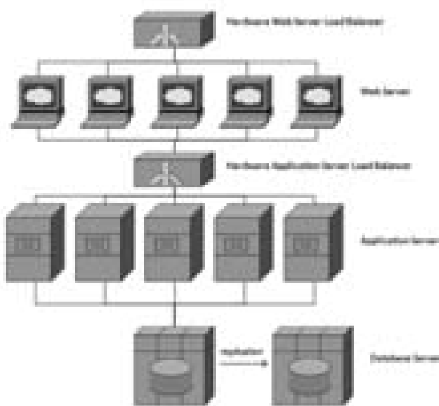

Abbildung 1: Schichten einer Unternehmensapplikation

Da beim Design heutiger Webapplikationen bereits auf eine Unterstützung von verteilten Systemen geachtet wird und somit eine horizontale Skalierung transparent ermöglicht, hat das Prinzip des Cloud Computing sehr schnell sehr großen Anklang gefunden, da es keine oder nur eine sehr geringe Adaptionszeit der Applikation benötigt. Beim Infrastructure-as-a-Service Cloud Computing wird dem Entwickler bzw. dem Applikationsmanager nicht nur die Verantwortung für die darunterliegende Hardwareinfrastruktur abgenommen, auch eine horizontale Skalierbarkeit, mit einer theoretisch unbegrenzten Anzahl von Knoten, ist ein Teil des Cloud Computing Angebots.

1.1 Konsistenzbetrachtung

Da es bisher unmöglich war, Datenkonsistenz über mehrere Server hinweg zu gewährleisten, ohne dabei Abstriche bei der Leistung zu machen, wurde die Brauchbarkeit einer horizontalen Architektur in der Datenschicht eingeschränkt.

Die Gültigkeit und Bedeutung von Brewers CAP Theorem [Br00] in Bezug auf Datenbanksysteme in der Cloud wurde sowohl von Prof. Michael Stonebraker [St10], dem ersten Entwickler von PostgreSQL, wie auch von Amazons CTO, Dr. Werner Vogels hervorgehoben.

Kurz zusammengefasst, besagt das CAP Theorem, dass es beim Entwerfen und der Implementierung von Applikationen in verteilten Umgebungen, drei bedeutende Anforderungen gibt, deren gleichzeitige Einhaltung unmöglich ist.

C: **Consistency** – Hierbei ist das Ziel, auf bei Transaktionen über mehrere Knoten hinweg, eine „all-or-nothing“ Sematik einzuhalten. Multiple Werte für das gleiche Datenstück sind verboten. Traditionelle relationale Datenbank fokussieren auf ACID Eigenschaften (atomicity, consistency, isolation, durability) und gewährleisten Konsistenz z.B. mit Isolationsmethoden. Wird Replikation eingesetzt, müssen auch alle Replikate einen konsistenten Status besitzen.

A: **Availability** – Hier ist das Ziel einer permanenten Verfügbarkeit des Systems. Sollte ein Knoten ausfallen, kommt es zum Failover auf einen anderen Knoten ohne die Gesamtverfügbarkeit zu beeinflussen. Im Fall von DBMS kann Replikation eingesetzt werden.

P: **Partition Tolerance** – Im Falle einer Netzwerkunterbrechung können zwei oder mehrere Knotengruppen entstehen, wobei Knotengruppe A nicht mehr mit Knotengruppe B kommunizieren kann.

Das CAP Theorem besagt nun, das im Fehlerfall nicht alle drei Ziele zu verwirklichen sind. Eine Eigenschaft muss zwangsläufig kompromittiert werden.

Zum Leben wird das CAP Theorem erst bei der Skalierung von Applikationen erweckt. Bei einem niedrigen Transaktionsvolumen haben kurze Verzögerungen, verursacht durch die Konsistenzwahrung der Datenbank, keinen merklichen Einfluss auf die Gesamtleistung des Systems. Laut Vogels [Vo08] treten diese Herausforderungen erst bei einer Skalierung von verteilten Systemen auf weltweiter Ebene hervor. Ereignisse, die in der Regel eine geringe Eintrittswahrscheinlichkeit haben, treten bei Milliarden von Anfragen garantiert ein und müssen schon im Voraus beim Entwurf des Systems und in der Architektur berücksichtigt werden.

Vogels argumentiert, dass Dateninkonsistenz in einer großen verteilten Umgebung aus zwei Gründen toleriert werden muss, da:

1. Die Schreib- und Leseperformance unter hoch nebenläufigen Bedingungen verbessert werden muss

2. Partitionen aufgrund von Netzwerkproblemen nicht zu einem Ausfall eines Teilsystems führen dürfen obwohl die Knoten noch aktiv sind.

C wird also für A und P geopfert. Die Konsistenzverantwortung wird der Applikation übertragen. Im Gegensatz zum “strong consistency” Modell, bei dem nach einem erfolgreichen Update *jeder* Zugriff bereits den neuen Wert liefert, und dem „weak consistency“ Modell, bei dem das System *keine* Garantie übernimmt bzw. erst nach einem *„inconsistency window“* garantiert, dass nachfolgende Zugriffe den neuen Wert liefern, befürwortet Vogels das “eventually consistent” Modell. Diese spezielle Form des weak consistency Modells garantiert, dass letztendlich (irgendwann) alle Zugriffe den neuen Wert liefern solange das Objekt keine neuen Updates erhält. Das inconsistency window wird beeinflusst von Faktoren wie Kommunikationsverzögerung (Latenzzeit), der Systemlast und die Anzahl der involvierten Replikate[1] – aber auch nur solange währenddessen keine Fehler auftreten!

Das eventual consistency Modell hat eine Paar Variationen zu bieten, wie z.B. „read-your-writes consistency“ oder „monotonic read/write consistency” die versuchen, einige der Schwächen des Modells, zu kompensieren. Trotzdem ist es offensichtlich, dass Unternehmensapplikationen, die heute beispielsweise mit einem hochverfügbaren Oracle Database Cluster auf der Datenschicht arbeiten, mit dieser „irgendwann Philosophie“ nicht zufrieden sein können. Eine berechtigte Frage auf Vogels Blogpost lautet daher: „Wie wird die New York Stock Exchange beispielsweise, konsistente Kurse an alle Investoren liefern können? Ist ‚irgendwann‘ gut genug?“

1.2 NoSQL Datenbanken

Amazons SimpleDB, ein verteiltes Datenbankmanagementsystem auf Basis von einfachen Key-Value Stores, basiert auf dem eventual consistency Modell. Da man sich aber bei Amazon der Schwächen bewusst ist, wurde Anfang 2010 eine „strong consistency“ Option eingeführt – „consistent read“. Bei dieser Option reflektieren Leseoperationen immer das vorherige Update. Damit will man bei Amazon speziell Applikationen adressieren, die ursprünglich eine klassische relationale Datenbank benützen, um so den Umstieg auf SimpleDB zu erleichtern.[2]

Alternativen zum bisherigen, fast vierzig Jahre altem Konzept einer relationalen Datenbank werden unter der Bezeichnung „NoSQL Datenbanken“ vereint. Dabei umschließt dieser Begriff eine Reihe von Datenbankkonzepten und Technologien, viele davon zur Lösung eines speziellen Bedürfnisses. Gemein haben NoSQL Datenbanken, dass sie Konsistenz für Skalierbarkeit (Größen- wie auch Komplexitätsskalierung) opfern und keine relationale Datenbank sind.

---

[1] Als Beispiel für das „eventually consistent“ Modell dient DNS (Domain Name System), bei dem Updates nach einem bestimmten Schema und in Verbindung mit einem zeitgesteuertem Cache propagiert werden; letztendlich werden alle Clients die Updates sehen.

[2] Auch diese Option wird aber von Vogels relativiert: “Even under extreme failure scenarios, such as complete datacenter failures, SimpleDB is architected to continue to operate reliably. However when one of these extreme failure conditions occurs it may be that the stronger consistency options are briefly not available while the software reorganizes itself to ensure that it can provide strong consistency. Under those conditions the default, eventually consistent read will remain available to use.” [Vo10]

Generell können NoSQL Datenbanken in vier Klassen (mit einigen Beispielen) eingeteilt werden:

1. Document-oriented Databases: MongoDB, CouchDB
2. Graph Databases: Neo4j, AllegroGraph
3. Tabular/Column-oriented Databases: Googles BigTable
4. Key-Value Stores: Amazons Dynamo und SimpleDB, Cassandra (auch column-oriented), Voldemort

Außer dem weicheren Konsistenzmodell und dem Fehlen einer einheitlichen Datenbanksprache, gibt es für NoSQL Datenbanken auch keine einheitlichen Abstraktionsbibliotheken wie z.B. JDBC oder ODBC. In eine Entscheidungsmatrix für einen IaaS- oder PaaS-Provider muss demnach auch immer der Vendor Lock-in für die Datenschicht betrachtet werden. Dies ist ein weiterer Grund dafür, warum viele Unternehmensapplikationen nicht in eine kommerzielle Cloud ausgelagert werden.

Das Problem der fehlenden Clienttreiber hat sich Xkoto (aufgekauft von Teradata) mit dem Produkt Gridscale angenommen. Gridscale besteht aus einer Reihe von Clienttreibern (ODBC, JDBC, CLI), einem (oder aus Redundanzgründen mehreren) Gridscale Server, die direkt zwischen den Applikationsserver und den Datenbankserver platziert werden, sowie aus DB Connectoren zu jedem Datenbankserver. Somit stellt sich Gridscale den Anwendungen gegenüber als eine einzelne, voll funktionsfähige Datenbank dar. [Xk08]

2 Spree DatabaseHypervisor Prototype

Das Konzept vom Zimory Projekt „Spree“ geht noch ein Stück weiter – selbst die Clienttreiber werden nicht modifiziert, an der Applikation ändert sich somit überhaupt nichts, und auch an den Datenbankservern müssen keine Modifikationen vorgenommen werden. Die Kernkomponenten des Systems ist der „DatabaseHypervisor“, einer neue Schicht zwischen dem JDBC Treiber und des Datenbankservers, der als Interceptor dient. Zusätzlich zur ursprünglichen „Master Datenbank“ können eine beliebige Anzahl von „Satelliten“ an den DatabaseHypervisor angeschlossen werden. Die Satelliten übernehmen alle Leseabfragen, somit wird die Master-Datenbank nur bei Schreiboperationen belastet.

Um die Konsistenz des Systems auch über mehrere Satelliten zu garantieren, werden verschiedene Strategien der Replikation verwendet. Details hierzu und zu den unten abgebildeten Benchmarking-Ergebnissen bleiben künftigen Veröffentlichungen überlassen.

Der erste Prototyp fokussierte auf eine Oracle 11g Datenbank als Master sowie auf mehrere Oracle 11g Datenbanken als Satelliten. Zur Performancemessung wurde der TPC-W Benchmark verwendet. Abbildung 2 zeigt den Leistungsgewinn

(Transaktionen/s) beim Hinzufügen von Satelliten mit dem TPC-W Benchmark (konfiguriert mit 5% Updates, 95% Selects). In der ersten Phase wird die reine Leistung des Masters gemessen. Nach dem Hinzufügen des ersten Satelliten, steigt die Gesamtleistung (schwarze Linie) nur minimal. Jedoch ist zu sehen, dass der Master nur noch Schreiboperationen (rote Linie) erhält, während der Satellit die Leseoperationen (grüne Linie) erhält. Ein deutlicher Gewinn wird ab dem zweiten Satelliten erzielt.

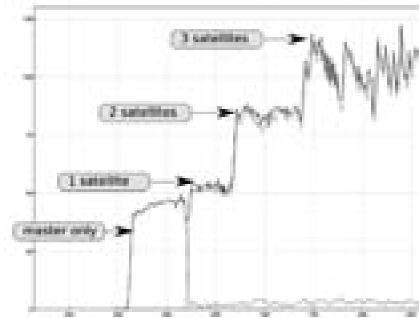

Abbildung 2: TPC-W Benchmark (5% Updates); Leistungsgewinn bei bis zu drei Satelliten

Da sowohl das Hinzufügen von Satelliten wie auch das Entfernen sehr schnell, transparent und on-the-fly passiert, lassen sich kurzfristige Lastspitzen auf der Datenschicht sehr gut bewältigen. Ein mögliches Szenario, wie es momentan auch bei Zimory entwickelt wird, ist der Einsatz der Satelliten auf einer pay-on-demand Basis als virtuelle Maschinen in der Cloud. Dabei können Resources eines externen Providers verwendet werden oder auch eigene Resourcen als „Enterprise Cloud" bereitgestellt werden.

Im Gegensatz zur NoSQL Initiative, bietet Zimory Spree somit eine horizontale Skalierbarkeit für Applikationen, die relationale Datenbanken benötigen und nicht auf Konsistenz verzichten möchten. Entwickler können damit weiterhin SQL für neue Applikationen benützen und an existierenden Systemen müssen keinerlei Änderungen vorgenommen werden, um sie in der Cloud zu betreiben, weder auf der Applikationsseite noch auf der Datenbank.

Literaturverzeichnis

[Br00] Brewer, E. A.: Towards Robust Distributed Systems, Proceedings of the Annual ACM Symposium on Principles of Distributed Computing, 2000

[St10] Stonebraker, M.: Errors in Database Systems, Eventual Consistency, and the CAP Theorem, http://cacm.acm.org/blogs/blog-cacm/83396-errors-in-database-systems-eventual-consistency-and-the-cap-theorem

[Vo08] Vogels, W.: Eventually Consistent – Revisited, http://www.allthingsdistributed.com/2008/12/eventually_consistent.html

[Vo10] Vogels, W.: Choosing Consistency, http://www.allthingsdistributed.com/2010/02/strong_consistency_simpledb.html

[Xk08] Technical Whitepaper Gridscale ® Database Virtualization Software;