VSDB

Datenbanksynchronisation verteilter Datenbanken

Ausarbeitung

Alexander Rieppel

4. Juni 2014

5AHITT

Inhaltsverzeichnis

1 Einführung **3**
1.1 Umsetzung im Diplomprojekt 3
1.2 Datenbanksynchronisation 4
1.2.1 Synchronisationsprobleme 4
1.2.2 Konfliktbehebung 5
1.2.3 MySQL Cluster 9

2 Couchbase Lite **10**
2.1 Allgemein 10
2.2 Features 11
2.3 Synchronisation mit Couchbase Sync Gateway 12
2.4 Vorteile gegenüber SQLite 12

1 Einführung

1.1 Umsetzung im Diplomprojekt

Ziel des Diplomprojektes war es eine Applikation für ein Tablet zu entwickeln, dass die Mitarbeiter der Lebensmittelversuchsanstalt bei der Probennahme im Supermarkt unterstützen soll. Insgesamt umfasst das Diplomprojekt eine Android Applikation und eine Server-Applikation. Bei der Probennahme werden Lebensmittel eingekauft, die am Tablet entsprechend in eine Datenbank eingetragen und später in der LVA für die weiter Bearbeitung bereitgestellt werden. Die Daten werden anschließend in die interne Datenbank in der LVA synchronisiert und für die Verarbeitung der Daten durch das Laborteam bereitgestellt. Auf weitere Vorgänge innerhalb der Firma hat allerdings die Tablet-Applikation keinen Einfluss mehr. Deshalb wird hier lediglich auf den Teil der Speicherung in die Datenbank und vor allem die Synchronisation mit der LVA-Datenbank eingegangen.

Innerhalb der Tablet Applikation wird eine SQLite Datenbank verwendet, da diese nicht sonderlich viele Ressourcen des Tablets benötigt und so eine einfache und angenehme Verwendung durch den Benutzer ermöglicht. Die Datenbank erlaubt zudem nicht allzu viele Datenbanken und im Vergleich zur klassischen MySQL Lösung nur wenige Nutzer. Auf dem firmeneigenen Server läuft eine klassische Mysql Instanz. Der Auftraggeber hatte MySQL als Vorgabe deklariert, da bereits eine Lizenz für Mysql vorhanden ist. Weitere in Betracht gezogene Datenbanksysteme waren PostgreSQL und ebenfalls auch für das Serverprogramm SQLite.

Die Umsetzung sah so aus, dass Daten vom Tablet zunächst an die Server-Applikation über Java-Streams versandt und anschließend von der Server-Applikation entsprechend in die Datenbank eingepflegt werden. Während Daten von der Datenbank auf das Tablet synchronisiert

sind, dürfen diese auch von niemandem bearbeitet werden. So wird Konsistenz innerhalb des Systems gewährleistet.[RBDT]

1.2 Datenbanksynchronisation

Die sichere Synchronisation von Daten in verteilten Systemen ist ein wichtiges Anliegen, da der Prozess in erster Linie Konsistenz der Daten gewährleisten muss. Zu diesem Thema gibt es verschiedene Ansätze auf die hier, bezogen auf das oben beschriebene Diplomprojekt, näher eingegangen wird.[1]

1.2.1 Synchronisationsprobleme

Der Kernpunkt ist, dass ein zentrales Datenbanksystem und ein oder mehrere kleinere Systeme existieren. Während die zentrale Datenbank sämtliche Daten des Systems beinhaltet, verfügen die kleineren Datenbanken nur über einen Bruchteil dieser Daten. Diese Bruchteile können von den einzelnen Geräten natürlich jederzeit geändert werden. Die Aufgabe besteht nun, dass all diese Bruchteile von Daten, wieder zur Hauptdatenbank synchronisiert werden ohne, dass größere Konflikte entstehen.

Zusätzlich zu den bereits geschilderten Punkten, werden noch wichtige Nebenpunkte betrachtet.

Die Verbindung der verteilten Datenbanken zum Hauptsystem muss nicht immer vorhanden sein. Das zentrale System muss wissen wann eine Verbindung besteht.

Die verteilten Datenbanken besitzen nur einen Teil der in der Hauptdatenbank gespeicherten Daten, wobei allerdings keiner der Datenbestände mit einem anderen der verteilten Datenbanken überlappt. Nur die zentrale Datenbank besitzt den selben Datenbestand wie eine der

[1] Nachfolgend geschilderte Ansätze und Lösungen dienen lediglich als Unterlage für eine mögliche Implementierung einer solchen Situation. Das beschriebene Diplomprojekt wurde ausschließlich so implementiert wie im entsprechenden Punkt angegeben und steht, abgesehen von der Gemeinsamkeit der Verwendung von verteilten Datenbanken, in keiner direkten Beziehung zu den geschilderten Ansätzen und Lösungen.

verteilten Datenbanken.

Ein Problem stellt hierbei ein Fall dar, wenn eine der verteilten Datenbanken vorübergehend offline geht und währenddessen Daten in der Hauptdatenbank geändert werden. Eine Methode um dies zu beheben wäre das simple überschreiben der Daten auf der entsprechenden Datenbank, falls einem der beiden Datenbanken eine höhere Priorität zugesprochen wird. Wenn zum Beispiel die verteilte Datenbank eine höhere Priorität besitzt als die zentrale Datenbank, kann die verteilte Datenbank die Daten der Hauptdatenbank einfach überschreiben.

Die Synchronisation der einzelnen Daten findet stets nur zwischen einer verteilten Datenbank und der zentralen Datenbank statt und keinesfalls unter zwei verteilten Datenbanken. Zusätzlich sind die Systemzeiten der einzelnen verteilten Datenbanken nicht synchronisiert. In einem zusammengefasst kommt man zu folgenden Punkten:

- Die Daten sollten konsistent bleiben, auch wenn die Synchronisation einmal fehlschlägt oder keine Verbindung besteht.
- Konflikthandhabung hängt von den einzelnen Tabellen ab
- Verteilte Datenbanken werden hier nur mit der zentralen Datenbank synchronisiert und nicht untereinander
- Datenbankzeiten werden nicht synchronisiert

[WFKM]

1.2.2 Konfliktbehebung

Besonders wichtig bei der Synchronisation von verteilten Datenbanken ist die Behebung von entstandenen Konflikten. Im Idealfall sollten diese natürlich gar nicht erst auftreten, doch muss auch eine passende Strategie für den Fall der Fälle vorhanden sein. Im Allgemeinen besteht die Behebung von Konflikten aus zwei Teilen: die Erkennung und die eigentliche Behebung. Die Erkennung ist hierbei ebenfalls ein wichtiger Faktor, da Fehler und Konflikte so früh wie möglich erkannt werden müssen um Konsistenz sicherzustellen und Datenverlust zu vermeiden. Unter anderem gibt es bei der Konfliktbehebung zwei Aspekte die ausschlaggebend sind.

In Konflikt stehende Teilnehmer

Zum einen ist es wichtig zu wissen, welche Teilnehmer den Konflikt verursacht haben und in welcher Beziehung diese Teilnehmer zu einander stehen. Zu beachten ist, dass mehr als zwei Teilnehmer in einen Konflikt verwickelt sein können.

In Konflikt stehende Daten

Der zweite Aspekt sind die eigentlichen in Konflikt stehenden Daten welche wiederum eigene Kriterien beinhalten die dabei beachtet werden müssen. Unter anderem wo die Daten gespeichert wurden in der Datenbank, welche Datentypen sie haben und welche Werte, bezogen auf die in Konflikt stehenden Teilnehmer, tatsächlich im Konflikt stehen.

Ein Beispiel für die Notwendigkeit des Ortes der Daten für die Konfliktbehebung ist, wenn Daten lediglich in einer Tabelle gespeichert wurden, welche für die Speicherung von temporären Daten zuständig ist. Hier hängt es nun davon ab wie wichtig diese Daten für andere Teilnehmer sind. Hier könnte als Problemlösungsansatz einfach immer der neueste Wert in die Tabelle gespeichert werden. Sinnvoll wäre es unter Umständen ebenfalls die Tabelle komplett aus dem Synchronisationsprozess zu entfernen und eine Ausnahme zu setzen, falls es nicht notwendig ist diese mit anderen Teilnehmern zu synchronisieren.

Behebungsstrategien

1. Zeitbasiert

   Die erste nennt sich zeit-basierte Strategie, bei der schlicht so vorgegangen wird, dass immer der aktuellste Wert übernommen wird. Der offensichtliche Nachteil dieser einfach zu realisierenden Strategie ist, dass sämtliche Updates die kurz danach unternommen wurden verloren gehen. Dies macht diese Strategie in seiner natürlichen Form nicht wirklich nutzbar. Auch wenn andere Strategien mit dieser erweitert werden, ist die Wahrscheinlichkeit eines Lost Update dennoch gegeben. Ein weiteres großes Problem dieser Strategie ist die Notwendigkeit, alle Uhrzeiten

der Datenbanken zu jeder Zeit synchron halten zu müssen. Da allerdings Zeitsynchronisation kein triviales Thema ist, könnte zum Beispiel eine logische Uhr verwendet werden.

2. Teilnehmerbasiert

   Eine andere Methode ist die Teilnehmer-basierte Strategie. Diese basiert auf den vorhandenen Datenbanken und den Beziehungen zwischen ihnen. Diese Strategie geht von der Tatsache aus, dass zumindest einer der Teilnehmer wichtiger ist als die anderen. Die Daten der weniger wichtigen Datenbanken würden demnach einfach überschrieben werden. Ein wichtiger Punkt dieser Strategie ist, dass die einzelnen Prioritäten der Datenbanken vorher festgelegt werden müssen. Dies kann, in Abhängigkeit von der Applikation, statisch mit Hilfe eines einfach zugewiesenen Wertes oder dynamisch mit Hilfe eines Algorithmus passieren, der beispielsweise auf der Netzwerkauslastung basieren kann.

   Die einfachste Möglichkeit ist, jedem Teilnehmer eine eindeutige Nummer zuzuweisen und nach dieser zu operieren. Wenn dies nicht so gehandhabt wird können Konflikte zwischen zwei gleichwertigen Datenbanken entstehen. Falls hier die Implementierung dieser Strategie nicht ausreicht, kann diese auch beispielsweise mit einer zusätzlichen Strategie erweitert werden. Ein möglicher Einsatzbereich dieser Strategie wäre ein Szenario wo eine zentrale Datenbank existiert und verschiedene verteilte Datenbanken synchronisieren ihre Daten ausschließlich mit dieser zentralen Datenbank. In diesem Fall wird die zentrale Datenbank wohl die höchste Priorität erhalten.

3. Datenbasiert

   Den kompliziertesten Ansatz für die Konfliktbehebung stellt die Daten-basierte Strategie dar, denn die benötigt eine bestimmte Interpretation der Daten. Die Idee hinter diesem Verhalten ist, dass die in Konflikt stehenden Daten analysiert werden und ein logischer Merge durchgeführt wird. Um dies zu realisieren, wird der Verfasser der in Konflikt stehenden Daten, die Datentypen, die Daten selbst und in manchen Fällen auch eine Kopie der

Daten vor der Änderung benötigt. Auch muss festgelegt werden, wie detailliert die Synchronisation stattfinden soll. Der Vorteil dieser Strategie ist, dass in Konflikt stehende Daten besser synchronisiert werden können und alle Änderungen der verteilten Datenbanken in Betracht gezogen werden können, ohne ein Lost Update in kauf nehmen zu müssen. Auf der anderen Seite ist eine Interpretation der Daten sehr komplex und schwierig zu implementieren, da, abhängig von der Applikation, verschiedene Faktoren beachtet werden müssen wie die Daten analysiert werden.

Als Beispiel für die Komplexität einer solchen Interpretation kann eine Textanalyse genannt werden. Auch wenn der in Konflikt stehende Text der einzelnen Teilnehmer vollständig vom Algorithmus erkannt und verstanden wird, ist es fraglich ob dieser Text so bearbeitet werden kann, dass eine generell gültige Version dieses Textes, für alle verteilten Datenbanken erstellt werden kann. Eine vereinfachte Möglichkeit könnte die Interpretation einzelner Datensätze sein, welche wiederum nach Wichtigkeit eingeordnet werden und gegebenenfalls unwichtige Datensätze gelöscht werden. Allerdings würde dies wieder bedeuten, dass Lost Updates entstehen können.

In einem simplen Webseitenstatistik-Systemen werden alle Zugriffe auf eine Webseite protokolliert und die Datenbanken dazu (A und B) in bestimmten Zeitabständen untereinander synchronisiert. In diesem Zeitabstand ändern sich die Zähler. Vor der Änderung stand die offizielle Anzahl an Zugriffen auf 10, während A bereits eine Zahl von 12 besitzt. Der Stand von B liegt bei 18. Nun muss die alte offizielle Anzahl von den neuen Zahlen in A und B jeweils abgezogen werden. So erhält man die Anzahl der jeweils neu hinzugekommenen Benutzer für A und B. Für A kommt man auf 2 neue Besucher und für B auf 8 neue Besucher. Die Zahlen werden anschließend addiert, was 10 ergibt und mit der alten Anzahl 10 addiert um die neue offizielle Anzahl an Zugriffen zu erhalten und zwar 20.

[Woh]

1.2.3 MySQL Cluster

Eine Möglichkeit für verteilte Umgebungen bietet MySQL Cluster welches eine hochverfügbare und hochredundante Version von MySQL darstellt. Die NDB (Network Database)-Speicher-Engine ermöglicht den Betrieb mehrerer MySQL Server in einem Cluster. Hierbei wird die Architektur des "Shared Nothing" verwendet, was bedeutet dass jeder Rechner-Knoten innerhalb des Systems seine eigenen Festplatten und Arbeitsspeicher besitzt. Im Prinzip kann die gesamte Datenbank im Arbeitsspeicher gehalten werden, wenn genügend Speicher vorhanden ist. Ab Version 5.0 wird lediglich eine solche Speicherung mit periodischen Speichervorgängen auf eine Festplatte unterstützt. Ab Version 5.1 existiert die Möglichkeit lediglich Indexfelder im RAM und den Rest auf der Festplatte zu speichern. Die NDB-Speicher-Engine ist eine unabhängige Komponente, die persistente Speicherung von Daten ermöglicht und für die Koordination aller Zugriffe auf Knoten in einem MySQL Cluster zuständig ist. Anwendungen können entweder direkt auf die NDB-Speicher-Engine über verschiedene NoSQL-Schnittstellen oder über einen MySQL-Knoten via SQL zugreifen.[Ora]

2 Couchbase Lite

2.1 Allgemein

Couchbase Lite ist eine leichtgewichtige, dokumenten-orientierte und leicht synchronisierbare Datenbank welche speziell für den Einsatz in mobilen Anwendungen und Geräten geeignet ist.

Leichtgewichtig. Die Datenbank Engine ist eine in der Applikation gebundene Bibliothek und kein separater Serverprozess. Sie besitzt sehr klein gehaltenen Code damit Apps die auf die Schnittstelle zurückgreifen rasch heruntergeladen werden können. Sie garantiert eine kurze Startzeit da mobile Geräte meist geringere CPU-Leistung haben als PCs. Mobile Datensätze sind zwar relativ klein, allerdings können manche Dokumente große multimediale Anhänge haben, weswegen ein geringe Speicherauslastung von Nöten ist. Sie bietet drüber hinaus eine gute Performance, wobei diese zu einem großen Teil von der implementierten Applikation abhängt.

Dokumenten orientiert. Sie speichert Einträge im flexiblen JSON-Format, weshalb keine vordefinierten Schemata benötigt werden. Dokumente können eine frei wählbare Größe von Binary-Anhängen besitzen, wie z.B. multimediale Inhalte. Das Datenformat der Applikation kann sich über die Zeit weiterentwickeln, ohne dass die Datenbank geändert werden muss. MapReduce[1] Indizierung bietet eine schnelle Datensatzabfrage, ohne dass spezielle Query-Languages verwendet werden müssen.

Leicht synchronisierbar. Zwei Kopien einer Datenbank können problemlos über einen Replikations-Algorithmus synchronisiert werden. Die Synchronisation kann On-Demand oder fortlaufend stattfinden.

[1] Ein Programmiermodell für die Prozessierung verteilter Daten

Geräte können auch nur Teilmengen einer riesigen Datenbasis eines Remote-Servers synchronisieren. Die Sync-Engine erlaubt auch das Synchronisieren über unbeständige und unzuverlässige Netzwerkverbindungen. Konflikte können einfach über einen Merge-Algorithmus, gefunden und behoben werden. Revisions-Bäume erlauben auch komplexe Replikations-Topologien, wie z.B. Server-to-Server (für mehrere Datenzentren) und Peer-to-Peer, ohne Datenverlust oder Konflikte befürchten zu müssen.

2.2 Features

Couchbase Lite besitzt eine native API für sowohl iOS als auch Android und bietet eine geringe Latenz und offline Zugang zu Daten. Interessant beim Einsatz von Couchbase Lite ist das zu Grunde gelegte **JSON Format**. Jedes Dokument innerhalb der Datenbank ist ein JSON Objekt, bestehend aus Key-Value Paaren. Diese Values können Arrays sein oder sogar nested Objects. Dies lässt eine zur mobilen Applikation passende Datenbankstruktur zu, ohne großartig komplexe joins o.ä. berücksichtigen zu müssen.

Einer der wichtigsten Punkte, speziell im Hinblick auf ein dynamisches Datenbankmodell, ist die **Schemalosigkeit**. Dies bedeutet, dass ein Datenmodell nicht zwingend bereits frühzeitig vorhanden sein muss. Das Datenmodell besteht aus Dokumenten welche verschiedene Strukturen haben können. Durch die native API können Datenbank Dokumente direkt mit dem eigenen Objektmodell assoziiert werden und ebenfalls ist es möglich mit den JSON Strukturen zu arbeiten. Für Applikationen mit Web-Technologien kann zusätzlich die Couchbase Lite REST API verwendet werden.

Auch wird die **Replikation** zu kompatiblen Datenbankinstanzen unterstützt. Dabei ist vor allem die peer-to-peer Unterstützung interessant, die es erlaubt unter Hinzunahme eines zusätzlichen HTTP Listeners die Daten per peer-to-peer Prinzip auszutauschen.

Darüber hinaus unterstützt Couchbase Lite **Offline-Zugriff** auf Daten. So können auch Daten offline bearbeitet und später wieder mit der Datenbank synchronisiert werden.[Coua]

2.3 Synchronisation mit Couchbase Sync Gateway

Couchbase Sync Gateway ist eine Erweiterung zum Couchbase Server 2.0 oder neuer, um Couchbase Lite als Replikations-Endpunkt einzurichten. Es besitzt einen HTTP-Prozess welcher als passiver Replikations-Endpunkt dient und benutzt einen Couchbase Server Bucket als persistenten Speicherort für alle Datenbank Dokumente.[Coub]

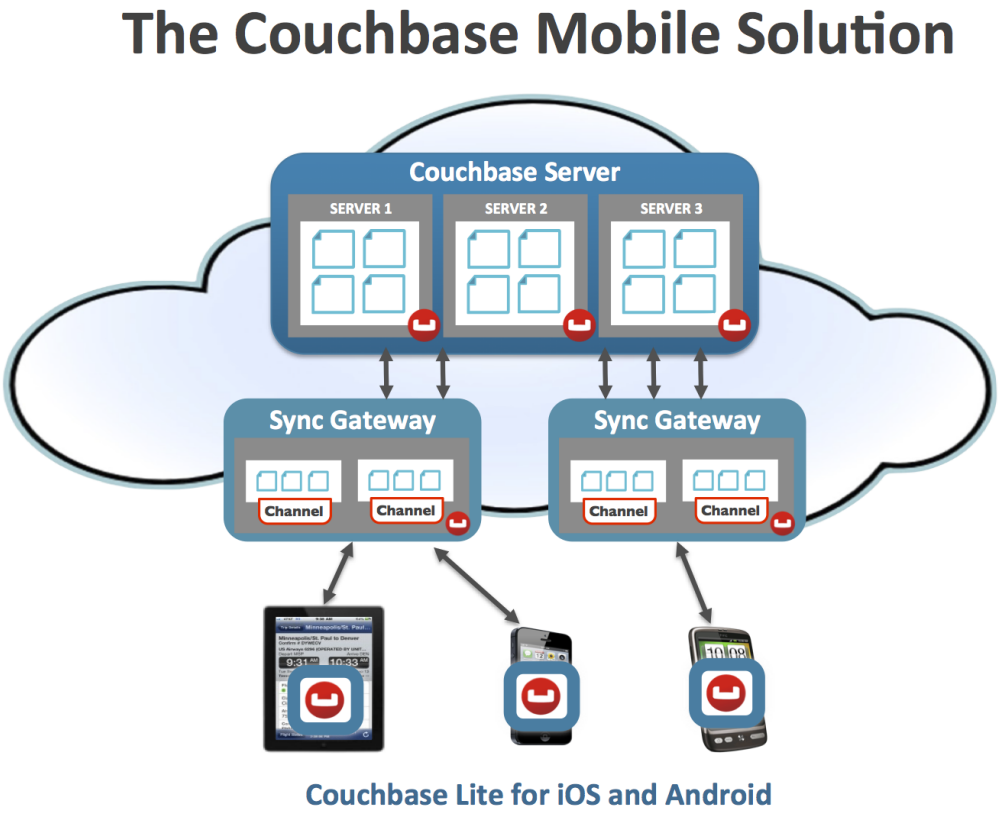

Abbildung 2.1: Architektur

2.4 Vorteile gegenüber SQLite

Allgemein kann man sagen, dass sowohl MySQL als auch Couchbase ihre Vor- und Nachteile haben. Bezogen auf das umgesetzte Diplomprojekt wäre jedoch eine Implementierung mit Couchbase wohl sinnvoller gewesen. Dieser Schluss ist insbesondere auf die Fähigkeit von

Couchbase und Couchbase Lite zurückzuführen, welche es erlaubt die Datenbankstruktur aufgrund der Schemalosigkeit dynamisch zu gestalten, ohne das Datenbankmodell ändern zu müssen. Dies war eine vom Auftraggeber gewünschte Option, jedoch kein Muss-Ziel. Trotzdem hätte es dem Auftraggeber in Zukunft einiges an Arbeit erspart, wäre nicht MySQL die vom Auftraggeber bevorzugte Lösung.

Außerdem wäre in einer erweiterten Implementierung der Software die Nutzung der Funktion von peer-to-peer Datenaustausch möglich falls später mehrere Geräte zum Einsatz kommen sollten. Auch der Support für Offline-Zugriff, der allerdings keinen direkten Vorteil zu SQLite darstellt, ist hier von Nutzen, da Probendaten offline gespeichert werden müssen. Dies würde vor allem zu einer saubereren Kommunikation und Synchronisation führen, als es bisher der Fall war.

Literaturverzeichnis

[Coua] Couchbase: *Couchbase Lite* - *http://docs.couchbase.com/couchbase-lite/cbl-concepts/*

[Coub] Couchbase: *Couchbase Sync Gateway* - *http://docs.couchbase.com/sync-gateway/*

[Ora] Oracle: *MySQL Cluster* - *http://dev.mysql.com/doc/refman/5.1/de/ndbcluster.html*

[RBDT] Rieppel, A. ; Backhausen, D. ; Dimitrijevic, D. ; Traxler, T.: *Diplomarbeit FI@D*

[WFKM] Wohlmuth, B. ; Frank, M. ; Kaltenböck, P. ; Masopust, T.: *Diplomarbeit Buga Tracking*

[Woh] Wohlmuth, B.: *Synchronisation of Distributed Databases*