JUST DO IT
HÖHERE ABTEILUNG FÜR
INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Ausarbeitung

Replikation

Systemtechnik
5BHIT 2016/17

Daniel May

Version 1.0
Note:
Betreuer: T. Micheler & M. Schabel
Begonnen am 1. November 2016
Beendet am 7. November 2016

Inhaltsverzeichnis

1 Einleitung

1.1 Definition

Replikation (lateinisch *replicatio* [1] "Wiederholung", "kreisförmige Bewegung") [2] ist "ein Verfahren der Datensicherung bei dem dieselben Daten von einem primären Speichermedium auf ein oder mehrere sekundäre Speichermedien kopiert werden." [3]

Demnach wäre Replikation also nichts anderes als ein simples Backup. Replikation wird jedoch besser definiert als mehrfache Speicherung **identer** Daten (an unterschiedlichen Orten) inklusive der Synchronisation selbiger. [4]

Zwar gibt es inkrementelle Backups, die ebenfalls eine gewisse Synchronisierung der Daten beinhalten, jedoch erfolgt diese bei Backups stets einseitig. Bei der Replikation kann sie auch bidirektional implementiert werden. Ebenfalls wird bei einem Backup nie direkt mit den Sekundärdaten, also dem eigentlich Backup, gearbeitet, diese dienen lediglich als Absicherung im Verlustfall.

1.2 Caching vs. Replikation

Caching (Cache = Versteck, geheimes Lager) wird, im Unterschied zur Replikation, definiert als **temporäres** Speichern von redundanten Daten, die dynamisch ausgewählt werden. Zusätzlich sollte der Cache für die Administration möglichst unsichtbar sein, wovon bei der Replikation nicht die Rede sein kann. [5]

2 Pro & Kontra

2.1 Gründe für Replikation

Die Replikation ist also die bewusste Erzeugung redundanter Daten, obwohl dies eigentlich den Normalformen, die beim Planen einer Datenbank beachtet werden sollten, widerspricht. [5] Trotzdem haben diese absichtlichen Redundanzen Vorteile:

2.1.1 Skalierbarkeit

Aufgrund mehrerer identer Datenbankinstanzen können Lesezugriffe verteilt werden. Somit wird die Last an den einzelnen Knoten reduziert und das Gesamtsystem wird dadurch leichter skalierbar. [5]

2.1.2 Verfügbarkeit

Mithilfe von Replikation wird ebenfalls die Verfügbarkeit des Gesamtsystems erhöht, da im Fehlerfall eines einzelnen Knotens die Erreichbarkeit des Systems nicht beeinträchtigt wird. Andere Knoten können das System weiterhin aufrecht erhalten, womit der Extremfall "Totalausfall" ausgeschlossen wird. Zusätzlich gehen im Gegensatz zu einfachen verteilten Datenbanken keine Informationen verloren, sollte eine Instanz (dauerhaft) ausfallen, da alle Instanzen den identen Datensatz zur Verfügung haben. [6]

2.1.3 Performance

Replikation verbessert die Antwortzeiten auf zwei Arten:

Erstens werden die Antwortzeiten des Systems verkürzt, da die Last auf mehrere Server verteilt wird. Somit haben einzelne Knoten mehr Ressourcen zur Verfügung um einem Client schneller zu antworten. Das ist der angenehme Nebeneffekt der Skalierbarkeit (Unterpunkt 2.1.1 auf Seite 1). [6]

Zusätzlich werden die lokalen Antwortzeiten des Systems verkürzt, da stets auf eine ortsnahe oder netztopologisch günstige Instanz der Datenbank zugegriffen wird. [6]

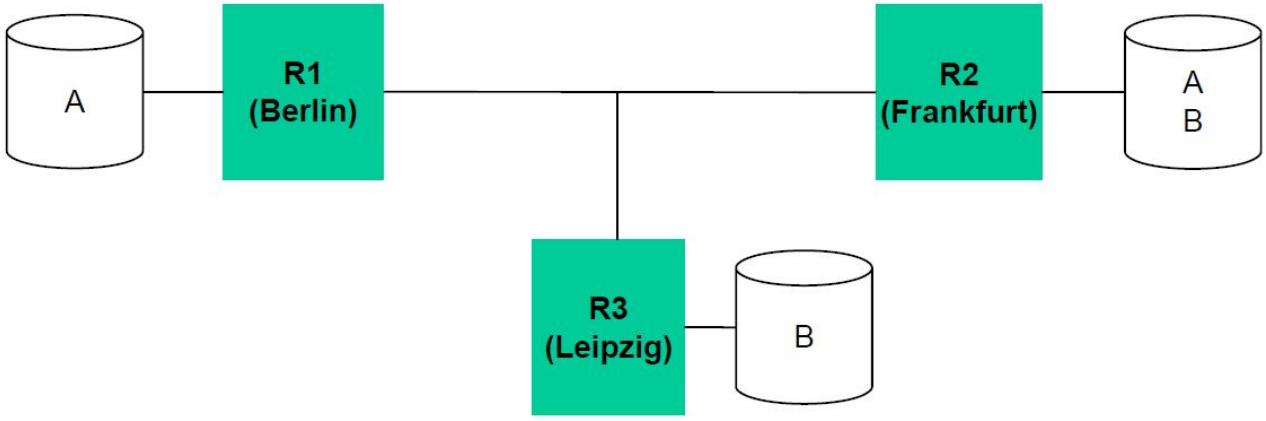

Abbildung 1: Replikate an verschiedenen Standorten [7]

Die verbesserten Antwortzeiten gelten meist nur für Lesezugriffe, da Schreibzugriffe in der Regel nicht vollkommen autonom ablaufen können. Dazu später mehr.

2.1.4 Disconnected Computing

Clients können ausgewählte Daten lokal replizieren, um damit später offline zu arbeiten. Im Anschluss müssen die Daten des lokalen Replikats mit den anderen Instanzen synchronisiert werden. [6] Die replizierten Daten sind in diesem Fall zwar meist nur temporär gespeichert, was eher auf die Funktion eines Caches hinweisen würde, jedoch wurden die zu replizierenden Daten im Vorhinein bewusst ausgewählt und im Anschluss werden sie wieder synchronisiert.

2.2 Nachteile

Für statische Daten sind die genannten Vorteile leicht mittels Replikation zu erreichen. Sollten sich die Daten ändern, müssen die Änderungen an alle Instanzen weitergegeben werden. Bei stark dynamischen Daten entstehen damit mehr Probleme, als gelöst werden. Deshalb wird Replikation typischerweise bei Stammdaten eingesetzt. [5]

Durch die Synchronisation entstehen folgende Probleme:

2.2.1 Aufwand

Durch die benötigte Synchronisation der einzelnen Knoten wird das Netzwerk zwischen den Servern zusätzlich belastet. Eine Änderung an einem Knoten muss letztendlich an alle propagiert werden, weshalb der Rechenaufwand und die Netzauslastung bei stark dynamischen Daten schnell zum Problem werden.

2.2.2 Speicherplatzbedarf

Die vielen Instanzen benötigen insgesamt logischerweise mehr Speicherplatz, womit die Kosten für das Gesamtsystem wesentlich ansteigen. Aufgrund der Replikation ist die Menge der Informationen trotz erhöhten Speicherbedarfs gleich. Deshalb sollten die Gesamtsysteme so konzipiert werden, dass die zu replizierenden Daten der einzelnen Knoten sinnvoll ausgewählt werden.

Beispielsweise werden in der Zentrale in Frankfurt alle Datensätze benötigt. In Berlin benötigt eine andere Filiale derselben Firma nur einen Teil der Daten. In Leipzig wird wiederum nur ein anderer Teil der Daten benötigt (siehe Abbildung 1 auf Seite 2).

2.2.3 Komplexität

Mit der Anzahl der Knoten wächst auch die Komplexität des Gesamtsystems. Dadurch wird die Installation, Konfiguration und Wartung des Systems aufwendiger, was wiederum zu erhöhten Kosten führt.

3 Anwendung

Bei der Implementierung von Replikation gibt es im Prinzip drei Ziele, die miteinander im Konflikt stehen.

Das erste Ziel ist die Erhaltung der Datenkonsistenz. Dies kann entweder über eine **1-Kopien-Äquivalenz** oder eine kleine Kopienanzahl erreicht werden. Die 1-Kopien-Äquivalenz besagt, dass jeder Zugriff auf eine beliebige Instanz des Replikationsverbundes den neuesten konsistenten Zustand liefert. [7] Das bedeutet, dass es möglichst zu jeder Zeit nur völlig idente Replikate gibt.

Das nächste Ziel, die Minimierung des Änderungsaufwands, wird ebenfalls durch eine kleine Anzahl von Kopien erreicht. Damit der Änderungsaufwand aber so gering wie möglich bleibt, sollen Replikate eher selten synchronisiert werden. [7]

Der eigentliche Benefit der Replikation ist die Erhöhung der Verfügbarkeit und die Performance-Steigerung (vor allem) bei Lesezugriffen. Im Gegensatz zu den beiden anderen Zielen, wird dieses entweder mithilfe einer großen Anzahl an Kopien oder mit dem Zugriff auf beliebige, möglichst wenige Kopien erreicht. [7] Letzteres erhöht jedoch nicht die Verfügbarkeit und kann nur implementiert werden, falls mehrere ortsnahe Knoten zur Verfügung stehen.

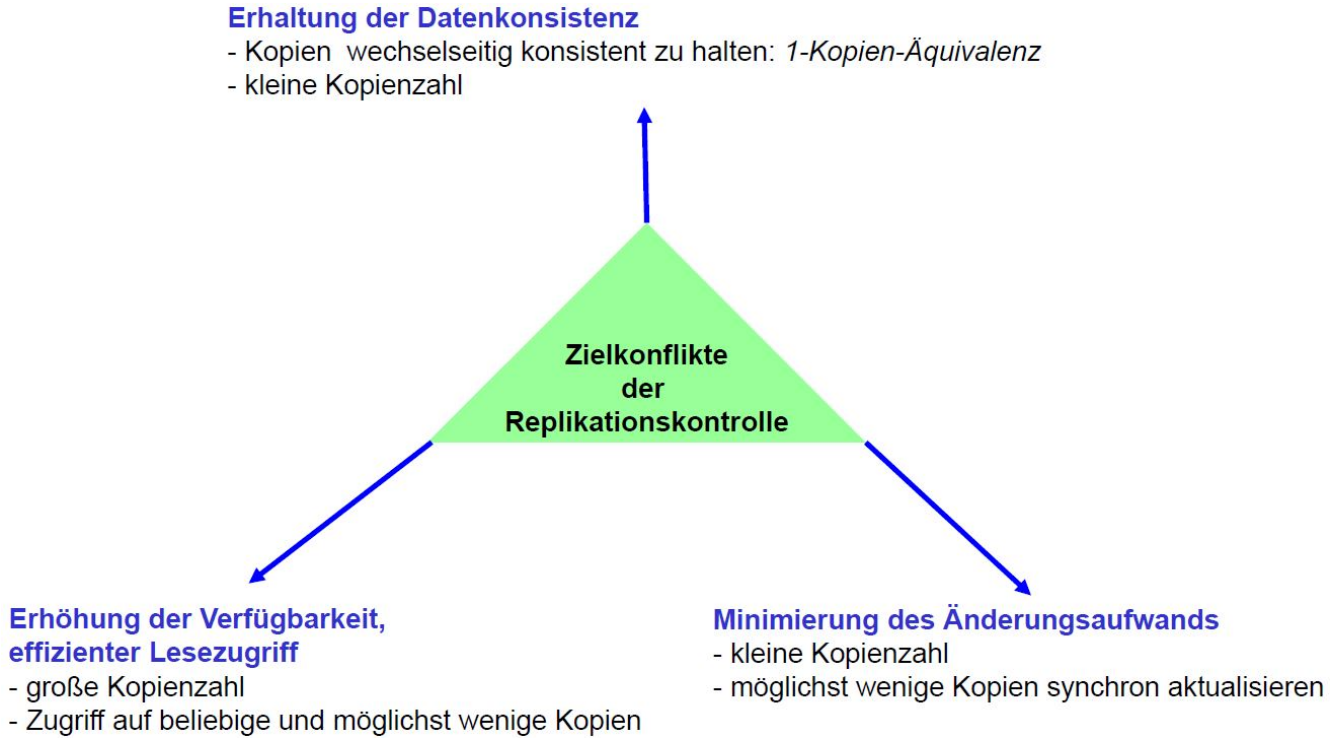

Abbildung 2: Zielkonflikte der Replikation [7]

3.1 Mobile Computing

3.1.1 Szenario

Für Außendienstmitarbeiter ist es wichtig, den Arbeitsort und die Arbeitszeit so flexibel wie möglich zu gestalten. Deshalb setzen Unternehmen zunehmend auf mobile Endgeräte. Aufgrund der Zeit- und Ortsflexibilität ist die Datenbank der Zentrale nicht ständig verfügbar. Damit die Endgeräte auch offline arbeiten können, müssen sie stets mit aktuellen, für den Mitarbeiter relevanten, Daten versorgt werden. Die benötigten Daten können je nach Termin variieren und werden daher täglich aktualisiert. In der Zentrale befindet sich die konsolidierte Datenbank aller Datensätze. [5]

3.1.2 Strategie

Ausgewählte Datensätze der konsolidierten Datenbank werden also regelmäßig auf mobile Endgeräte repliziert. Finden auf den mobilen Endgeräten Änderungen statt, stellt sich die Frage, wie diese gespeichert werden. Nun gibt es zwei Arten dieses System zu implementieren. Entweder sind die replizierten Daten reine Lesekopien und Änderungen werden nur lokal gespeichert, d.h. es können keine Verkäufe oder ähnliche Schreibzugriffe getätigt werden. Ein andere Möglichkeit wäre es, das System bei erneuter Konnektivität zu synchronisieren. [6]

Dabei ist das Hauptziel die Erhöhung der Verfügbarkeit, die über eine hohe Kopienanzahl erreicht wird. Die Erhaltung der Datenkonsistenz ist bei dieser Art von Daten (Kundenstammdaten) leicht einzuhalten, da sie recht statisch sind. Der Änderungsaufwand ist ebenfalls gering, falls nur Lesezugriffe stattfinden. Die Endgeräte werden einfach täglich aktualisiert und arbeiten den restlichen Tag mit dem gleichen Datensatz. Kompliziert wird es jedoch, wenn die Synchronisation beidseitig funktionieren soll. Mit den Schreibzugriffen wird es schwieriger die Daten konsistent zu halten, da bei n Endgeräten am Ende des Tages n verschiedene Replikate vorhanden sind. Werden diese am Ende des Tages synchronisiert kann dies einen erheblichen Änderungsaufwand verursachen.

3.1.3 Vorteile durch Replikation

Diese Szenario wäre ohne Replikation nur mit einer zentralen Datenbank bewältigbar. Dadurch fehlt aber die Möglichkeit offline zu arbeiten, was auch der größte Vorteil der Replikation in diesem Szenario ist. Zusätzlich ist die Antwortzeit bei einer zentralen Datenbank wesentlich höher. [5]

3.2 Skalierbarkeit von Leselast

3.2.1 Szenario

Webseiten verarbeiten täglich bis zu mehrere Millionen Anfragen. Bei diesen Anfragen handelt es sich bei ca. 97 - 99% um Lesezugriffe. [5] Einzelne Server sind logischerweise nicht in der Lage diese Vielzahl an Zugriffen zu verarbeiten. Deswegen setzt man einen Lastverteilungsverbund ein, in dem die Anfragen auf einzelne Knoten verteilt werden. Somit wird das Gesamtsystem auch für viele gleichzeitige Lesezugriffe einsetzbar.

3.2.2 Strategie

Es gibt also einen Serverbund aus möglichst identen Datenbanken, um die Vielzahl der Zugriffe aufzuteilen. Üblicherweise gibt es in diesen Konfigurationen einen Master-Server, auf dem auch Änderungen (Aktualisierungen der Informationen) durchgeführt werden können. Alle anderen Knoten sind Slave-Kopien, welche die Daten vom Master-Server erhalten. Auch hier werden Daten eingesetzt, die eher statisch sind.

Das Hauptziel ist hier die Skalierbarkeit, also die Lastverteilung. Diese wird durch eine mittlere - hohe Anzahl an Kopien erreicht, auf die möglichst wahlfrei zugegriffen werden kann. Da die Daten recht statisch sind, ist es ein leichtes, diese konsistent zu halten. Der Änderungsaufwand bleibt dadurch ebenfalls gering. Würde es sich um stark dynamische Daten handeln, würde der Änderungsaufwand und die fehlende Konsistenz den eigentlichen Nutzen der Replikation übersteigen.

3.2.3 Vorteile durch Replikation

Dieses Szenario wäre anders nur bewältigbar, wenn jeder Server verschiedenste Aufgaben erledigt und das System somit die Last verteilt, ohne teilweise redundant zu sein. Das Problem dieser Implementierung ist, dass die Aufgaben nicht immer gerecht geteilt werden und die Nutzungshäufigkeit der einzelnen Aufgaben nicht vorhergesehen werden kann.

3.3 Hochverfügbarkeit

3.3.1 Szenario

Viele Dienste müssen rund um die Uhr verfügbar sein und verwalten deshalb redundante Serverkopien. Fällt ein einzelner Knoten aus, so kann der nächste übernehmen.

3.3.2 Strategie

Dies könnte mit der vorhin beschriebenen Master/Slave Konfiguration mit vielen Slave Kopien gelöst werden. Wie bereits beschrieben, ist dieses Prinzip nur für statische Daten rentabel. Ändern sich die Daten sehr häufig, ist es sinnvoll die Anzahl der Slave Kopien zu reduzieren. Ansonsten ist der zu betreibende Änderungsaufwand zu hoch. Im Extremfall sollte mit einer einzelnen Slave Kopie gearbeitet werden, um den Synchronisierungsaufwand zu reduzieren. [5] Damit die Performance des Systems nicht beeinträchtigt wird, werden die durchzuführenden Änderungsanweisungen an den Slave geschickt, der im Prinzip dieselben Aufgaben etwas zeitversetzt abarbeitet. Würden beide Server bei jeder Transaktion synchronisiert werden, würde die Bearbeitungszeit einer Anfrage wesentlich erhöht werden.

Das Hauptziel ist also die erhöhte Verfügbarkeit, die bei statischen Daten über eine große Kopienanzahl realisiert werden kann. Sind die Daten dynamisch, wird die Anzahl der Kopien eingeschränkt. Die Daten sollten so konsistent wie möglich sein, da der Hauptserver jederzeit ausfallen kann. Der Änderungsaufwand wird gering gehalten, indem eine Master/Slave Konfiguration eingesetzt wird.

3.3.3 Vorteile durch Replikation

Eine gewisse Ausfallsicherheit kann durch ein fehlertolerantes Design des Systems (USV, Notstromaggregat, usw.) erreicht werden. Ist beispielsweise aber die Serverhardware an sich defekt, so kann nur Redundanz und die damit verbundene Synchronisation der Schlüssel zum Erfolg sein.

4 Asynchrone & synchrone Replikation

Abhängig von den oben genannten Zielen und Einsatzmöglichkeiten, kann die Replikation unterschiedlich implementiert werden. Der erste Unterschied zwischen einzelnen Implementierungen ist der Zeitpunkt der Synchronisation der Replikate.

4.1 Synchrone Replikation

Zugriffe auf Replikate sollten sich nicht von Zugriffen auf die Ursprungsdaten unterscheiden. Dies wäre der Fall, wenn eine **1-Kopien-Äquivalenz** (siehe Seite 4) gegeben wäre. Ein Weg zu diesem Ziel wäre die synchrone Replikation. [8]

Hierbei werden alle Änderungen, die an einem Knoten vorgenommen werden, verzögerungsfrei auch an den anderen Knoten abgearbeitet. Somit gibt es jederzeit nur identische Kopien. Bevor also ein Schreibvorgang erfolgreich abgeschlossen ist, müssen alle Knoten des Systems den Vorgang bestätigen. Schlägt die Operation auf einem Knoten fehl, wird die Änderung auf allen Knoten rückgängig gemacht und die gesamte Transaktion ist fehlgeschlagen. [6]

4.1.1 Vorteile

Das ACID (Atomicity Consistency Isolation Durability) Prinzip wird mit synchroner Replikation weiterhin unterstützt. Jede Transaktion wird, für das Gesamtsystem gesehen, als atomare Einheit betrachtet. Man spricht auch von transaktionaler Konsistenz. [5] Während der Änderung auf einem anderen Knoten werden Sperren auf den anderen eingesetzt, um ein eventuell gleichzeitiges Bearbeiten zu verhindern. Synchronisierungskonflikte können somit gar nicht erst entstehen. [9]

4.1.2 Nachteile

Sind alle Instanzen des Systems untereinander erreichbar, dann erhält man stets die neuesten Daten. Ist ein Teil des Verbunds aber nicht erreichbar, dann können veraltete Kopien entstehen. Meistens wird dies aber verhindert, indem Änderungen bei einem Teilausfall nicht möglich sind. [5]

Somit wird auch die Verfügbarkeit des Systems beeinträchtigt. In einer solchen Konfiguration ist das Gesamtsystem nur erreichbar, wenn alle Teilsysteme erreichbar sind. Dies verschlechtert sogar die Verfügbarkeit des Gesamtsystems. Die Daten des Systems sind nur bearbeitbar, wenn in dem Moment keine Änderung stattfindet. Somit bringen mehrere Server keinen Benefit puncto Skalierbarkeit mehr.

Da auf die Rückmeldung aller Knoten gewartet werden muss, wird die Performance von Schreibzugriffen drastisch beeinflusst. [5] Das Resultat dieser Einschränkung ist, dass synchrone Replikation nur über geringe Distanzen oder Hochleistungsnetzwerke verwendet werden kann. [3] Somit ist eine weltweite Verteilung auch nicht realisierbar.

4.1.3 Einsatz

Für mobile Computing ist synchrone Replikation also ungeeignet, da es hier um die Verfügbarkeit geht, die mit synchroner Replikation eher eingeschränkt wird. Die Hochverfügbarkeit wäre damit nicht möglich, da zu lange Wartezeiten entstehen und das Service damit wieder nicht verfügbar ist.

Bei eher statischen Daten kann die Leselast gut mit synchroner Replikation skaliert werden.

4.2 Asynchrone Replikation

Da die synchrone Replikation oft nicht möglich ist bzw. teilweise mehr Nachteile als Vorteile bringt, gibt es auch die asynchrone Replikation.

Hierbei werden Änderungen vorerst nur lokal vorgenommen. Damit das Datenbanksystem trotzdem noch die Forderung der Datenkonsistenz erfüllt, wird der Begriff Konsistenz durch **Konvergenz** ersetzt. Die Konvergenz stellt nur sicher, dass die Replikate nach einer gewissen änderungsfreien Zeit ident sind. [5]

4.2.1 Vorteile

Da nicht auf das OK der anderen Knoten gewartet werden muss, reduziert die asynchrone Replikation die Antwortzeit bei Schreiboperationen drastisch gegenüber der synchronen Replikation. [5]

Die Verfügbarkeit wird ebenfalls verbessert, da die Knoten auch bei Teilausfällen verfügbar sind.

4.2.2 Nachteile

Bei verspäteter Synchronisierung der Datensätze kann es zu Konflikten kommen. Hierbei können Aktualisierungs-, Eindeutigkeits- und Löschkonflikte auftreten.

Bei Aktualisierungskonflikten wurde das gleiche Datenobjekt von unterschiedlichen Replikaten aus verändert. Nun muss entschieden werden, welche Version gültig ist. [8]

Eindeutigkeitskonflikte betreffen das Einfügen von Datensätzen. Versuchen zwei Replikate ein Datenobjekt, mit dem gleichen Primärschlüssel oder anderen Daten die eindeutig sein müssen, hinzuzufügen, ergibt sich ebenfalls ein Konflikt. [8]

Löschkonflikte entstehen, wenn der Datensatz auf einem Replikat gelöscht wird und auf einem anderen später noch bearbeitet wird. [8]

Mit diesen Konflikten steigt der Updateaufwand erheblich an, es sei denn, die Konsistenz bzw. Konvergenz des Systems ist nicht von höchster Bedeutung.

4.2.3 Einsatz

Für dynamische Datensätze wird ausschließlich asynchrone Replikation eingesetzt.

Für mobile Computing eignet sich asynchrone Replikation perfekt, da die Außendienstmitarbeiter somit auch offline arbeiten können. Hochverfügbarkeit kann ebenfalls mit asynchroner Replikation gelöst werden, jedoch muss hier beachtet werden, dass Informationen verloren gehen, da die neuesten Änderungen im Fehlerfall noch nicht übertragen wurden.

Für die Skalierung reiner Leselast ist diese Art der Replikation ungeeignet, da die Replikate nicht konsistent sind und kein Performanceunterschied zur synchronen Replikation besteht.

5 Unidirektionale & bidirektionale Replikation

Ein weiterer Unterschied zwischen den Implementierungsvarianten ist die Richtung der Synchronisation.

5.1 Unidirektionale Replikation

Zur Skalierung der Leselast implementiert man einen Hauptserver, auf dem Änderungen durchgeführt werden können, und mehrere Nebenserver, auf denen Daten lediglich gelesen werden können. Dieses Master/Slave oder Publish-Subscribe Modell ist der Grundgedanke der unidirektionalen bzw. asymmetrischen Replikation.

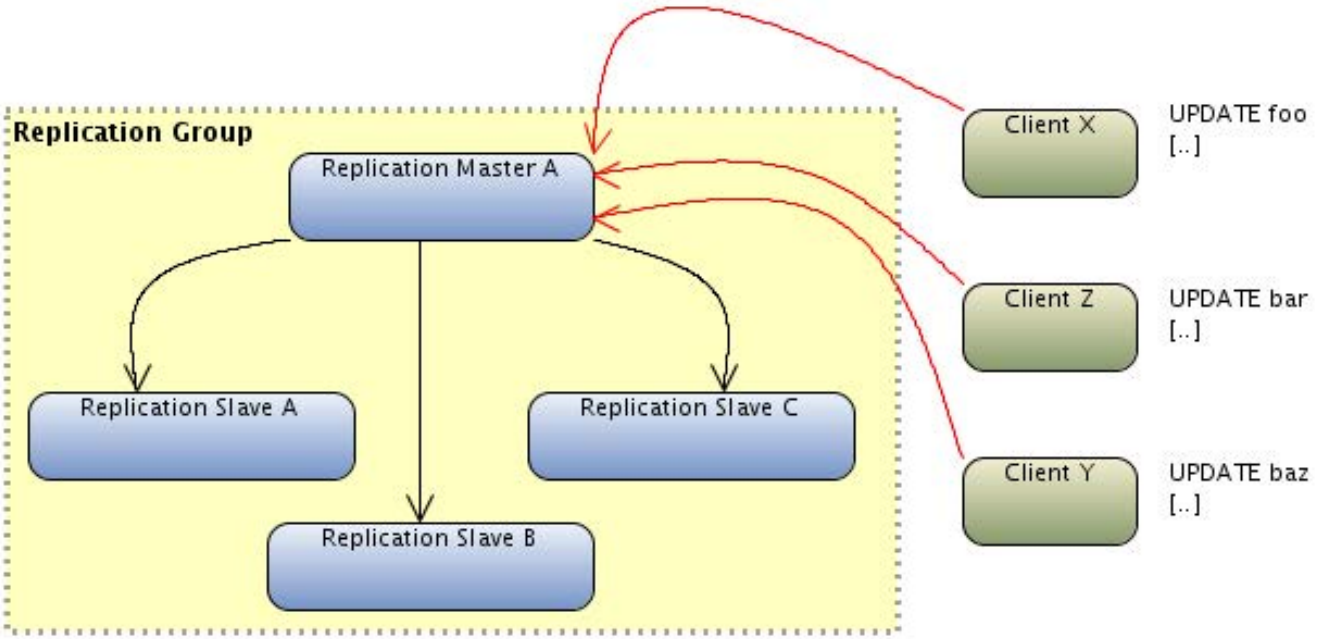

Abbildung 3: Asymmetrische Replikation [6]

Hierbei werden alle Änderungen nur auf einer Instanz durchgeführt. Diese propagiert die Updates weiter und kümmert sich darum, dass die Slave-Server die aktuellen Daten erhalten. Auf die Nebenserver kann also nur für Leseoperationen zugegriffen werden. [6]

5.1.1 Vorteile

Der größte aber auch einzige Vorteil ist, dass keine Konflikte entstehen können. [6]

5.1.2 Nachteile

Die Rechenlast, für das Management der Synchronisation, liegt einzig und allein beim Master-Server. Darunter leiden vor allem die Skalierbarkeit und Verfügbarkeit des Gesamtsystems, da ein einzelner Server für eine Vielzahl an Schreiboperationen nicht ausreicht.

5.1.3 Einsatz

Für mobile Computing ist unidirektionale Replikation eher weniger geeignet, da Außendienstmitarbeiter ihre Änderungen (Verkäufe) nicht durchführen können. Die Offline-Daten können lediglich für Produktinformationen und ähnliches verwendet werden.

Perfekt geeignet ist die asymmetrische Replikation für die Skalierung reiner Leselast. Auch für Hochverfügbarkeit ist sie geeignet, da im Fehlerfall einfach ein Slave-Server die Rolle des Master-Servers übernehmen kann.

5.2 Bidirektionale Replikation

Da es auch Szenarien gibt, in denen der Schreibzugriff an mehreren Stellen sinnvoll ist, steht im Gegensatz zur asymmetrischen Replikation die bidirektionale.

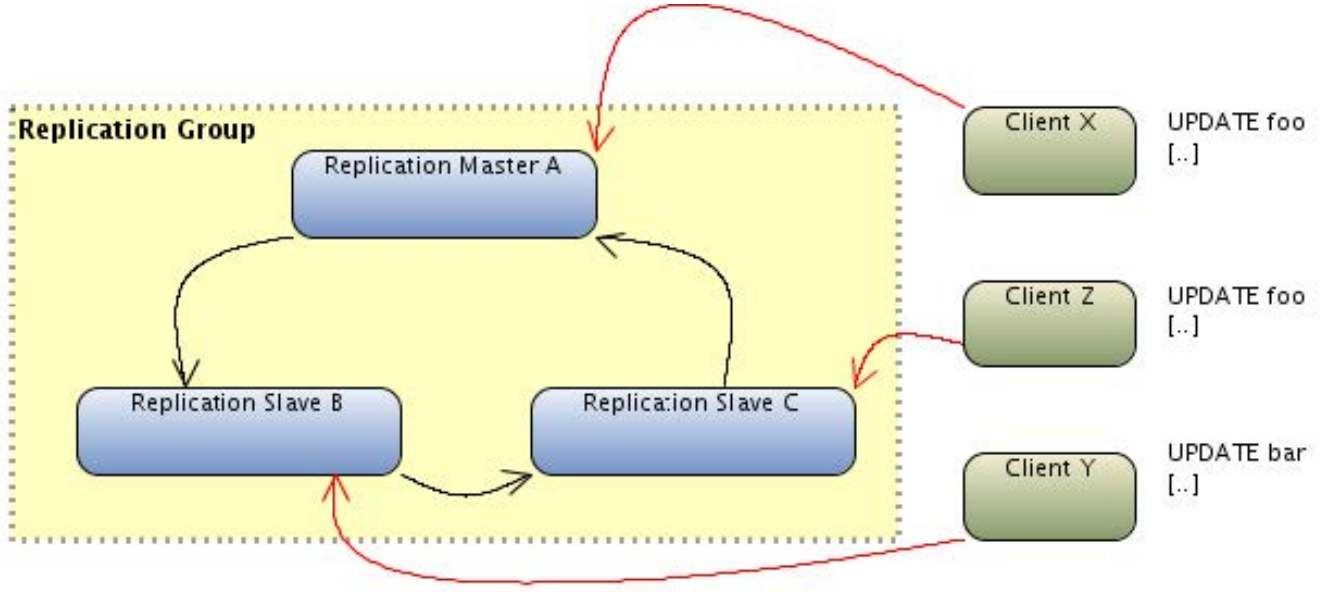

Abbildung 4: Symmetrische Replikation [6]

Bei dieser Implementierungsvariante ist das Ändern von Datensätzen an jedem Knoten erlaubt. [6] Damit besitzt jedes Replikat die identen Schreib- und Leserechte.

5.2.1 Vorteile

Durch die mögliche Änderung an jeder Stelle können auch Schreibzugriffe skaliert werden. Dies geschieht durch eine Lastverteilung der Anfragen.

Ein gleichzeitiger Nebeneffekt ist, dass die Antwortzeit sinkt, da stets eine ortsnahe Kopie gewählt wird. Dies gilt logischerweise nur in Kombination mit asynchroner Replikation, da es ansonsten ständig Sperren gäbe.

5.2.2 Nachteile

Der Nachteil ist, dass hier wieder Konflikte entstehen können. Zwar könnte bidirektionale Replikation auch synchron implementiert werden, jedoch ist dies in der Praxis nicht der Fall. Aufgrund der asynchronen Replikation ergeben sich wieder die Synchronisierungskonflikte (siehe Seite 8). Deshalb ist abzuwägen, ob die Lastverteilung im Verhältnis zum gestiegenen Kommunikationsaufwand für die Synchronisierung rentabel ist. [6]

5.2.3 Einsatz

Für mobile Computing eignet sich die bidirektionale Replikation sehr, da somit auch offline produktiv gearbeitet werden kann.

In der Hochverfügbarkeit könnte man symmetrische Replikation entweder mit synchroner kombinieren und würde somit ein System erhalten, das für einzelne Anfragen lange Antwortzeiten hat und somit nicht mehr ständig verfügbar ist. Oder man kombiniert sie mit asynchroner Replikation, was im Fehlerfall einen erhöhten Informationsverlust zur Folge hätte, da die Änderungen noch nicht synchronisiert wurden. Für die Skalierung der Leselast eignet sich diese Strategie nicht, da sie keinen zusätzlichen Benefit dafür bringt.

6 Klassifikation

Die unterschiedlichen Implementierungen lassen sich also durch die zwei bereits beschriebenen Eigenschaften klassifizieren.

1. **Wann** werden die Änderungen synchronisiert?
2. **In welche Richtung** werden Änderungen synchronisiert?

Bei asynchroner Replikation spricht man von fauler (lazy, manchmal auch optimistic) Replikation. Im Gegensatz dazu steht die synchrone Replikation, die als eifrige (eager manchmal auch pessimistic) Replikation bezeichnet wird. Das Änderungsrecht wird in der folgenden Tabelle mit dem Besitz der Daten gleichgestellt. [5]

Propagation vs. Ownership	**Lazy**	**Eager**
Group	n transactions n object owners	one transaction n object owners
Master	n transactions one object owner	one transaction one object owner

Tabelle 1: Klassifikation [10]

Replikation lässt sich auch anhand der folgenden Baumgrafik klassifizieren. Dabei wählt man zuerst das Korrektheitsziel, also ob die Daten transaktional konsistent oder nur konvergent sein sollen.

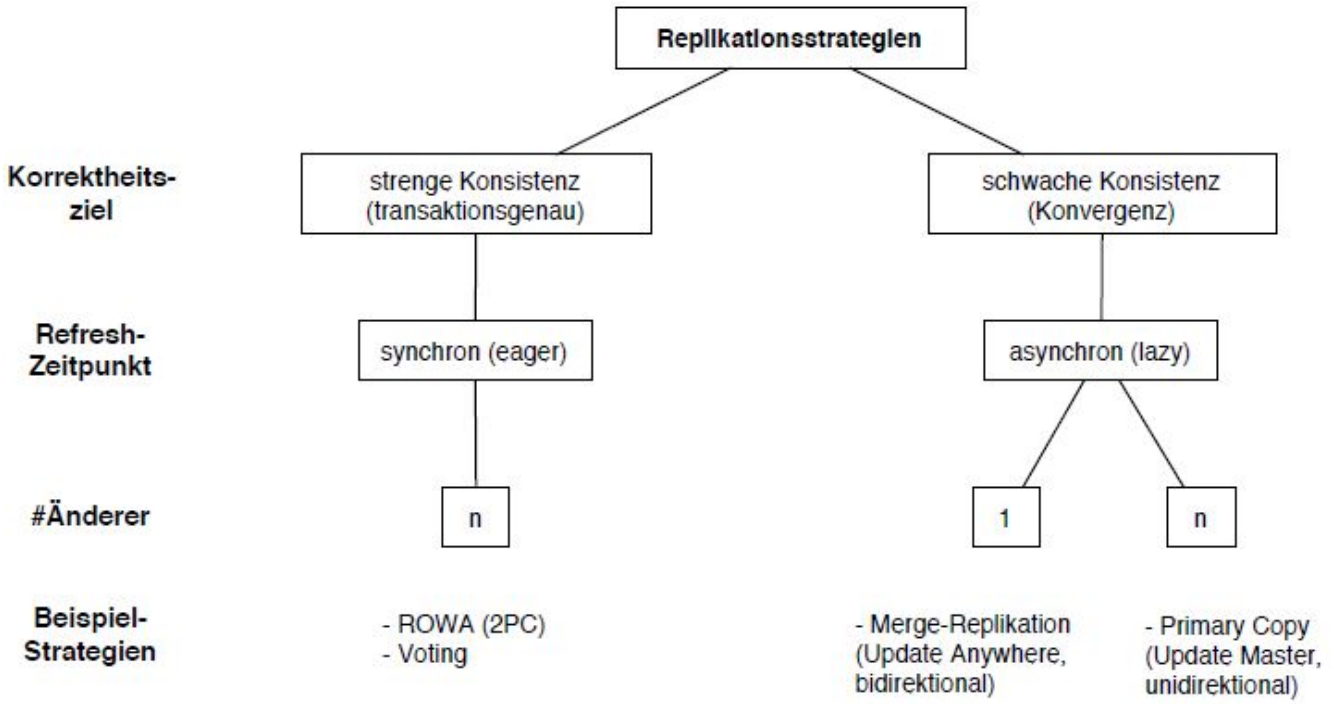

Abbildung 5: Klassifikation [6]

Bei der asynchronen Replikation kann man nun noch entscheiden, ob ein Knoten für Schreibzugriffe ausreichend ist, oder ob alle Knoten Veränderungen vornehmen können sollen.

Da bei der synchronen Replikation Sperren verhängt werden, bringt ein einzelner Master-Server keinen Konsistenzvorteil gegenüber einer Multi-Master Architektur. Dies ist der Grund, warum diese Variante in der Grafik gar nicht erst vorkommt.

7 Resümee

Replikation ist also die absichtliche Erstellung redundanter Daten und die Synchronisierung selbiger.

Die Gründe für den Einsatz der Replikation sind im Prinzip Performancesteigerungen, Verbesserungen der Verfügbarkeit und der Skalierbarkeit.

Dafür muss ein erhöhter Verwaltungsaufwand in Kauf genommen werden. Das große Problem mit der Replikation ist aber, dass bei der Zusammenführung veränderter Daten Konflikte entstehen.

Da je nach Anwendungsfall verschiedene Kriterien eine Rolle spielen, können Replikationsverfahren im Grunde durch zwei Parameter unterschieden werden. Erstens durch den Zeitpunkt der Synchronisierung und zweitens durch die Richtung der Synchronisierung.

Literatur

[1] Bibliographisches Institut GmbH. Duden: Replikation. http://www.duden.de/rechtschreibung/Replikation. [Online; Zugriff am: 7. November 2016].

[2] dict.cc. replicatio. http://browse.dict.cc/latein-deutsch/replicatio.html. [Online; Zugriff am: 7. November 2016].

[3] ITWissen. Replikation. http://www.itwissen.info/definition/lexikon/Replikation-replication.html. [Online; Zugriff am: 7. November 2016].

[4] Wikipedia. Replikation (Datenverarbeitung). https://de.wikipedia.org/wiki/Replikation_(Datenverarbeitung). [Online; Zugriff am: 7. November 2016].

[5] Florian Munz. Einsatz von Replikation. http://wwwlgis.informatik.uni-kl.de/archiv/wwwdvs.informatik.uni-kl.de/courses/seminar/SS2004/ausarbeitung14.pdf, Juli 2004. [Online; Zugriff am: 7. November 2016].

[6] Yves Adler. Replikation in Datenbanken. http://www.imn.htwk-leipzig.de/~kudrass/Lehrmaterial/DB2-VL/DB2-08/14A-Referat.pdf, Juni 2008. [Online; Zugriff am: 7. November 2016].

[7] Prof. Dr. E. Rahm. Replizierte Datenbanken. http://dbs.uni-leipzig.de/file/mrdbs-ws11-kap7.pdf, 2011. [Online; Zugriff am: 7. November 2016].

[8] Heiko Hornung. Replikation in verteilten Datenbanken. https://www.informatik.tu-darmstadt.de/BS/Lehre/Sem2000/proceedings/p110-final.ps.gz, 2000. [Online; Zugriff am: 7. November 2016].

[9] Ralf Burger. Desaster Recovery und Replikation. http://burger-ag.de/replikation.whtml. [Online; Zugriff am: 7. November 2016].

[10] Jim Gray, Pat Helland, Patrick O'Neil, Dennis Shasha. The Dangers of Replication and a Solution. http://db.cs.berkeley.edu/cs286/papers/dangers-sigmod1996.pdf, Mai 1996. [Online; Zugriff am: 7. November 2016].

Tabellenverzeichnis

Abbildungsverzeichnis

