MySQL Replikationstechnologien – eine Übersicht

Lenz Grimmer
Oracle Deutschland B.V. & Co. KG
Hamburg, Germany

Keywords:

MySQL, Replikation, Backup, Scale-Out, Hochverfügbarkeit, Open Source, Cluster, DRBD, Galera, PBXT, Tungsten

Einführung

MySQL Replikation kann für verschiedene Zwecke eingesetzt werden: zur Erhöhung der Redundanz/Verfügbarkeit, zur Leistungssteigerung (Scale-Out) oder zur Erstellung von Sicherungskopien (Backups). Dabei gibt es gibt unterschiedliche Wege, die Datenbankinhalte eines MySQL-Servers von einem System zum nächsten zu replizieren.

Neben der in MySQL bereits enthaltenen Technologie gibt es diverse externe Lösungen, die zum Einsatz kommen können. Dieser Vortrag gibt eine Übersicht über die Möglichkeiten und Technologien zur MySQL-Replikation, mit einem Schwerpunkt auf Linux-basierten Applikationen unter einer Open-Source Lizenz.

Vorgestellt werden unter anderem:

- MySQL Replikation (incl. MySQL Cluster)
- DRBD
- Galera
- PBXT Replikation
- Continuent Tungsten

Replikation: Definition und Klassifizierung

Bevor ich auf Replikation im Kontext von MySQL eingehe, möchte ich kurz ein paar grundlegende Informationen zum Thema Replikation und Datenbanken vermitteln.

Laut Wikipedia bezeichnet Replikation „*die mehrfache Speicherung derselben Daten an meist mehreren verschiedenen Standorten und die Synchronisation dieser Datenquellen.*“. Im Datenbank-Kontext bedeutet dies, daß die Inhalte einer Datenbank oder Tabelle in einer bestimmten Art und Weise zu einem anderen Datenbankserver transferiert werden. Hierbei kommen zwei grundlegende Methoden in Frage: die anweisungsbasierte sowie die zeilenbasierte Replikation.

Anweisungsbasierte Replikation

Bei diesem Verfahren werden lediglich die datenverändernden SQL-Anweisungen (z.B. `INSERT`, `UPDATE`, `DELETE`) repliziert. Der Empfänger wendet diese Kommandos auf seinen lokalen Datenbestand an und transformiert ihn damit in den gleichen Zustand wie auf der Quellseite. Diese Art der Replikation hat den Vorteil, sehr kompakt zu sein – es müssen nur wenig Daten (die SQL-Anweisungen)

tatsächlich repliziert werden. Weiterhin erleichtet diese Methode die Überwachung einer Applikation (Auditing), da die Anweisungen im Klartext übertragen werden.

Nachteilig an dieser Methode ist die Tatsache, daß der Empfänger alle Anweisungen erneut ausführen muß, was zusätzliche CPU-Zeit und eine erhöhte Belastung des Systems bedeuten kann. Dies ist insbesondere dann relevant, wenn eine relativ komplexe Anweisung nur zu verhältnismäßig wenigen Zeilenänderungen führt. Auch die Replikation von nicht-deterministischen Funktionen gestaltet sich auf diesem Wege schwierig – das Ausführen der gleichen Anweisung auf einem anderen System kann unter Umständen zu verschiedenen Ergebnissen führen. Andere Anweisungen liefern wiederum systemspezifische Rückgabewerte und erzeugen daher ebenfalls unterschiedliche Ergebnisse, wenn sie auf verschiedenen Systemen ausgeführt werden.

Zeilenbasierte Replikation

Diese Methode repliziert nur die tatsächlichen Änderungen, die aus einer bestimmten Anweisung resultieren, d.h. jede Zeile, die durch ein `UPDATE`/`INSERT`/`DELETE`-Kommando verändert, hinzugefügt oder gelöscht wurde. Diese Änderungen werden dann entsprechend beim Empfänger in der lokalen Kopie des Datenbestands eingepflegt. Das Endergebnis ist ein identisches Abbild der Quell-Datenbank auf der Empfängerseite. Da die tatsächlichen Änderungen übertragen werden, ist auch die Replikation von Ergebnissen nicht-deterministischer oder systemspezifischer Funktionen ohne weiteres möglich.

Zeilenbasierte Replikation ist typischerweise weniger CPU-intensiv, da die Zeilen und Änderungen ohne großen zusätzlichen Aufwand auf der Empfängerseite übernommen werden können. Das Volumen der zu replizierenden Daten erhöht sich bei dieser Methode im Vergleich zur anweisungsbasierten Replikation jedoch erheblich. Dies gilt es bei der Kapazitäts- und Bandbreitenplanung entsprechend zu berücksichtigen.

Neben der Frage, in welcher Form die Daten übertragen werden, gibt es weiterhin noch den Aspekt, wann die Replikation der Daten erfolgt und zu welchem Zeitpunkt die Erfolgs-Rückmeldung an die Applikation erfolgt. Hierbei gibt es zwei Ansätze:

Asynchrone Replikation

Bei diesem System besteht eine Verzögerung zwischen der Erstellung der Daten und der Replizierung und Festschreibung auf einen oder mehrere Empfänger. Die Datenbank bestätigt der Applikation die ordnungsgemäße Ausführung einer Transaktion bereits, während diese noch auf die anderen Knoten repliziert wird.

Asynchrone Replikation ist üblicherweise robuster gegenüber (temporären) Netzwerkausfällen, ohne den laufenden Betrieb zu unterbrechen – die Replikation wird einfach wieder aufgenommen und an der Stelle weitergeführt, an der sie abgebrochen wurde. Auch für Weitverkehrsnetze mit geringerem Durchsatz und längeren Signal-Laufzeiten (z.B. zwischen verteilten Rechenzentren) ist asynchrone Replikation meist die bessere Wahl. Durch diese Latenz ist allerdings das Risiko eines Datenverlusts höher – fällt der Hauptknoten aus während die Empfänger-Knoten noch dabei sind, vorherige Transaktionen zu replizieren, sind alle folgenden Transaktionen verloren, obwohl der Applikation bereits der erfolgreiche Abschluß dieser Vorgänge vermeldet wurde.

Synchrone Replikation

In diesem Modus wurden die Daten hingegen garantiert bereits auf alle teilnehmenden Knoten repliziert, bevor die Applikation über den erfolgreichen Abschluß der Transaktion informiert wird – eine Änderungsoperation an einem Datenobjekt kann nur dann erfolgreich abgeschlossen werden, wenn sie auch auf den Replikaten durchgeführt wurde. Die dafür erforderliche Kommunikation zwischen Sender und Empfänger verlängert konsequenterweise die Wartezeit auf die Rückmeldung für die Applikation – die Latenzzeit zwischen Applikation und Datenbank wird deutlich länger.

Semi-Synchrone Replikation

Diese Sonderform der Replikation versucht einen Kompromiss zwischen Datenverlust und Latenz zu erreichen, indem eine Transaktion als erfolgreich abgeschlossen vermeldet wird, wenn die Daten zu mindestens einem Empfänger-Knoten repliziert worden sind. Dies stellt sicher, daß es zumindest eine vollständige Kopie aller Datensätze auf einem anderen Knoten gibt, falls der Hauptknoten mit Datenverlust ausfallen sollte.

Anwendungsgebiete für Datenbank-Replikation

Es gibt diverse Szenarien, in denen die Replikation von Datenbankinhalten von Nutzen sein kann:

Hochverfügbarkeit durch Redundanz: Verteilte Kopien an verschiedenen Standorten ermöglichen einen ausfallsicheren Betrieb. Im Falle eines Ausfalls des Primärsystems wird der Daten-Verkehr einfach an die verbleibenden Replikas weitergereicht. Dabei müssen diese Systeme so dimensioniert sein, daß sie diese zusätzliche Last abfangen können, ohne selbst an ihre Kapazitätsgrenzen zu stoßen und damit weitere Ausfälle zu provozieren.

Skalierung: Dies ist besonders im MySQL-Umfeld eine beliebte Methode der Leistungssteigerung, wenn ein einzelnes System an seine Belastungsgrenzen stößt. Anstatt die Leistung und Kapazität eines einzelnen Datenbank-Servers immer weiter nach oben zu treiben (Scale-Up), wird die (Lese-)Last per Replikation und mit Hilfe eines Load Balancers auf mehrere physikalische Server verteilt, die ihre Inhalte vom Primärsystems replizieren (Scale-Out).

Datensicherung: Das Durchführen von Backups auf dem primären Datenbankserver kann Performance-Einbußen mit sich ziehen und damit den ordnungsgemäßen Betriebsablauf behindern. Die Daten werden stattdessen auf einen weiteren, dedizierten Backup-Server repliziert, auf dem die Datensicherung durchgeführt wird. Dies reduziert die I/O-Last auf dem Primärsystem. Im Falle von asynchroner Replikation sind damit auch Offline-Backups möglich: die Datenbank kann zur Erstellung einer konsistenten Sicherungskopie heruntergefahren werden und wird danach wieder hochgefahren. Im Anschluß daran werden alle Änderungen nachgezogen, die in der Zwischenzeit auf dem Hauptserver aufgelaufen sind.

Schema-Änderungen und System-Wartungsarbeiten ohne Ausfall: Bestimmte Änderungen der Tabellenstruktur können z.B. bei MySQL eine nicht unerhebliche Zeit dauern und würden damit den Produktivbetrieb negativ beeinflussen. Stattdessen werden solche Änderungen auf einem Replikations-Server durchgeführt, der sich danach wieder mit dem Primärsystem synchronisiert. Sobald der aktuelle Stand erreicht wurde, wird der Replikations-Server mit der geänderten Tabellenstruktur zum neuen Primärsystem ernannt und übernimmt dessen Aufgabe, bis auch dort die Anpassung der Tabellenstruktur durchgeführt worden ist und die Inhalte wieder synchronisiert wurden.

Neben diesem zusätzlichen Nutzen stellt Datenbank-Replikation allerdings auch neue Herausforderungen dar. Mit steigender Anzahl von Knoten in einem Replikations-Setup steigen die Anforderungen –

viele Kopien eines Systems in konsistenten Zustand lauffähig zu halten ist kein triviales Prolem. Falls es also schon schwierig ist, einen einzelnen Datenbankserver zu verwalten und zu warten, wird es damit noch schwieriger. Die Automatisierung und Integration eines Clusters gehört damit zu den wichtigsten Aufgaben eines System-Betreuers. Auch für die Datenbank-Anwendungen stellen sich neue Probleme dar – sie muß die richtige Kopie der Daten für eine bestimmte Operation finden und benutzen.

Replikation im MySQL Server

Der MySQL Server bietet bereits seit Version 3.23 anweisungsbasierte Replikation von Datenbanken und Tabellen an. Die dafür erforderliche Funktionalität ist fester Bestandteil des Servers und kann daher als sehr ausgereift bezeichnet werden. Die Änderungen werden dazu wie weiter oben beschrieben als SQL-Anweisungen übertragen und auf dem Slave wiederholt. Hierzu verwaltet der Master-Server ein oder mehrere sogenannte Binärlog-Dateien und eine Index-Datei. Das Binärlog enhält alle datenverändernden SQL-Anweisungen wie z.B. `INSERT`/`UPDATE`/`DELETE` in einem komprimierten Format, die Index-Datei führt Buch darüber, welche Binärlogs im Moment verfügbar sind.

MySQL-Replikation arbeitet nach dem Pull-Verfahren – es ist Aufgabe des Replikations-Slaves, sich regelmäßig mit dem Master in Verbindung zu setzen und die aktuellsten Änderungen seit dem letzten Replikationslauf abzuholen und nachzuvollziehen. Weiterhin arbeitet das MySQL-Replikationsverfahren unidirektional: ein Master kann einen oder mehrere Slaves bedienen. Es ist im Moment nicht vorgesehen, daß mehrere Master ihre (unterschiedlichen) Inhalte auf einen gemeinsam genutzen Slave replizieren. Es ist möglich, daß zwei Server gleichzeitig Master und Slave zueinander sind und somit Änderungen theoretisch auf beiden Systemen gleichzeitig durchgeführt werden können. Diese Vorgehensweise sollte jedoch mit Vorsicht genossen werden: aufgrund der asynchronen Natur der Replikation ist im Falle eines Ausfalls von einem der Knoten die Konsistenz der Daten nicht mehr gewährleistet.

MySQL-Replikation erfolgt ausschließlich asynchron: Replikations-Slaves können also den Änderungen auf dem Master-Server „hinterherhinken“. Weiterhin erfolgt die Replikation der Daten auf dem Slave-Server single-threaded, d.h. die Änderungen werden nicht wie auf dem Master in parallel laufenden Threads vorgenommen, sondern werden in serialisierter Form aus dem sogenannten Relay-Log abgearbeitet.

Seit MySQL 5.1 ist alternativ zur anweisungsbasierten Replikation auch zeilenbasierte Replikation möglich: dabei werden nur die tatsächlich geänderten, gelöschten oder hinzugefügten Zeilen übertragen. Dies ist z.B. erforderlich, um Daten zwischen zwei MySQL Cluster-Instanzen zu replizieren. MySQL unterstützt darüberhinaus auch einen sogenannten „mixed mode“, bei dem der Server dynamisch zwischen beiden Modi wechselt, je nachdem was für die aktuelle Anweisung sinnvoller ist (standardmäßig wird anweisungsbasiert repliziert).

Mit MySQL 5.5 wird zusätzlich zur klassischen asynchronen Replikation ein Verfahren namens „semisynchrone Replikation“ eingeführt: normalerweise schreibt der Master alle Veränderungen in sein Binärlog, ohne Informationen darüber zu haben, ob oder wann ein Slave diese Daten abgeholt und verarbeitet hat. Im Falle eines Ausfalls des Masters besteht also die Möglichkeit, daß Transaktionen noch von keinem Slave verarbeitet wurden und damit verloren gegangen sind, falls im Rahmen einer Hochverfügbarkeitslösung nun der Slave die Rolle des Master-Servers übernimmt.

Semisynchrone Replikation bietet sich hier als Alternative an: ein Slave-Server meldet sich dem Master gegenüber als „semisynchron-fähig“ an. Falls diese Funktionalität auf dem Master ebenfalls

aktiviert ist, wartet eine auf dem Master ablaufende Sitzung, die eine Transaktion abschließen möchte so lange, bis zumindest einer der angeschlossenen semisynchronen Replikations-Slaves vermeldet, daß er die Transaktion vollständig empfangen hat (oder wenn ein definierbarer Zeitraum überschritten wurde). Erst dann bekommt die Applikation vom Server die Bestätigung, daß die Transaktion erfolgreich abgeschlossen wurde. „Vollständig empfangen“ heißt in diesem Kontext, daß der Slave die Transaktion in sein Relay-Log geschrieben hat, nicht daß er sie schon verarbeitet und in seine Kopie der Datenbank übernommen hat. Falls es zu einer Zeitüberschreitung kommt und bis dahin kein Slave den Empfang der Transaktion quittiert hat, fällt der Master zur klassischen asynchronen Replikation zurück, bis zumindest ein semisynchroner Slave sich wieder zurückgemeldet hat und wieder auf dem aktuellen Stand ist.

Für MySQL 5.6 ist weiterhin verzögerte Replikation in der Entwicklung. Hierbei kann der Slave explizit so konfiguriert werden, daß die Replikation einen konfigurierbaren Zeitintervall hinter dem Master nachläuft. Dies kann z.B. sinnvoll sein, um Anwenderfehler auf dem Master-Server „auszubügeln“ – ein versehentliches `DELETE FROM` ohne `WHERE...`-Bedingung läßt sich so abfangen, bevor es den verzögerten Slave erreicht und erlaubt damit eine Wiederherstellung des vorherigen Zustandes ohne einen Backup aus anderer Quellen wieder einspielen zu müssen. Aus zum Testen des Verhaltens einer Applikation bei Verzögerungen in der Replikation ist diese Funktionalität nützlich. Die Möglichkeit, schnell auf einen älteren Stand der Datenbank zugreifen zu können spricht ebenfalls für den Einsatz eines Slaves mit verzögerter Replikation. Durch Kaskadierung von Replikations-Slaves mit unterschiedlichen Verzögerungszeiten lassen sich interessante Archivierungsstrukturen aufbauen.

MySQL Cluster

MySQL Cluster ist ein transaktionelles, ACID-konformes relationales Datenbanksystem und wurde ursprünglich unter dem Namen „NDB“ (Network Database), bei Alzato (einem Eriksson Spin-Off) entwickelt. Die Firma inklusive des Produkts und der Mitarbeiter wurde im Oktober 2003 von MySQL AB übernommen.

MySQL Cluster wird als Hochverfügbarkeits-Clustering-Lösung für MySQL Server positioniert, die aber auch sehr gut zur Skalierung von schreibintensiven Lasten genutzt werden kann. NDB Cluster ist als Speicher-Engine für MySQL Server realisiert – die Tabellen werden anstatt auf dem lokalen Dateisystem in einem verteilten Cluster-Verbund redundant abgelegt.

Der MySQL-Server agiert nur als SQL-Frontend dazu – Cluster kann auch komplett unabhängig davon verwendet werden. Dafür stellt MySQL Cluster sehr flexible APIs bereit: Daten können u.a. mittels SQL (via MySQL Client-Server Protokoll), NDB-API (C++), Java, LDAP und HTTP ausgetauscht werden. Es handelt sich hierbei um eine sehr offene Plattform: eine Skalierung des Systems ist mit herkömmlicher PC-Hardware möglich. Insbesondere wird für MySQL Cluster kein Shared-Storage (z.B. SAN) benötigt.

Durch die Verwendung einer „shared-nothing“-Architektur ist ein Verfügbarkeitsgrad von 99.999% möglich. Der Ausfall eines Cluster-Knotens wird in Sekundenbruchteilen erkannt und behoben. Das System agiert selbstheilend: ausgefallene Knoten integrieren sich automatisch wieder in den Cluster und resynchronisieren sich. Neue Cluster-Knoten können ohne Ausfallzeit hinzugefügt werden; Schema-Änderungen, Versions-Upgrades und Patches lassen sich im laufenden Betrieb mittels „Rolling Updates“ durchführen.

Die Replikation zwischen den einzelnen Datenknoten findet synchron mittels Zwei-Phasen-Commit-Protokoll statt. Die Inhalte werden automatisch redundant über die verfügbaren Cluster-Knoten verteilt und partitioniert. Die Replikation zwischen MySQL Cluster Instanzen (z.B. zwischen zwei Rechenzentren) erfolgt über die herkömmliche, asynchrone MySQL-Replikation (zeilenbasiert) via SQL-Knoten. Die Skalierung von schreibenden Zugriffen wird durch die verteilte, parallele Architektur von MySQL Cluster begünstigt – eine Applikation kann parallel auf mehrere MySQL-Server Instanzen zugreifen, die Ihre Tabellen im gemeinsam genutzen Cluster-Verbund ablegen.

Komplexe Datenbank-Abfragen mit `JOIN`s über viele Tabellen oder full table scans sind wegen des erhöhten Kommunikationsbedarfs zwischen den Datenknoten allerdings meist nicht so performant wie auf einer Speicher-Engine wie InnoDB. Essentiell für MySQL Cluster ist daher ein dediziertes (und geschütztes) Netzwerk mit geringer Latenz, z.B. ein geswitchtes Gigabit-Ethernet-Segment. Darüberhinaus kann mit Hilfe von SCI-Karten (Dolphin Express) eine weitere Leistungssteigerung erzielt werden (Dolphin SuperSockets). Diese Karten sind besonders auf niedrige Latenzzeiten ausgelegt und werden sehr häufig im HPC-Clustering-Umfeld (Supercomputing) verwendet.

DRBD

DRBD (http://drbd.org/), das „**D**istributed **R**eplicated **B**lock **D**evice“ ist eine Entwicklung der Fa. Linbit aus Österreich und ist seit Version 2.6.33 offiziell im Linux-Kernel enthalten. Vereinfacht gesagt kann man dieses Verfahren auch als "RAID1 über das Netzwerk" bezeichnen. DRBD erstellt eine 1:1-Kopie eines Block-Geräts auf einem entfernen Server, indem jeder Datenblock, der sich auf dem Primärknoten ändert, über eine dedizierte Netzwerkverbindung auf einen Sekundärknoten übertragen wird. Je nach Anforderung an Konsistenz und Latenz kann die Blockreplikation in synchroner oder asynchroner Form erfolgen. Ein weit verbreitetes Anwendungsgebiet für DRBD ist die Verwendung als Teil einer Cluster-Hochverfügbarkeitslösung mit Pacemaker (Linux-HA). Dabei läuft der Sekundärknoten als passiver Teilnehmer im „Hot-Standby“-Betrieb. Temporäre Ausfälle eines Knotens meistert DRBD dank automatischer Resynchronisation im Hintergrund ohne Probleme.

Aufgrund der Tatsache, daß DRBD auf Block-Ebene agiert, ist es Anwendungs- und Dateisystem-agnostisch und unterstützt damit auch MySQL Server. Es empfiehlt sich, ein Journaling-Dateisystem wie XFS, ReiserFS, JFS oder ext3/4 zu verwenden, um einen zeitaufwendigen Dateisystem-Check im Falle eines Wechsels vom Primärknoten zum Sekundärknoten zu vermeiden. Im Falles eines Ausfalls des Primärknotens wird der Sekundärknoten zum neuen Primärknoten ernannt und bindet nun das replizierte Dateisystem ein. Nach erfolgtem Konsistenzcheck kann nun der MySQL-Server seine Arbeit aufnehmen.

Je nachdem, unter welchen Bedingungen der Wechsel durchgeführt wurde, muß InnoDB zuerst ebenfalls einen Konsistenzcheck und Log-Recovery durchführen, bevor neue Datenbankverbindungen angenommen werden. Aus diesem Grund ist MyISAM als Tabellenformat für den Einsatz mit DRBD denkbar ungeeignet, da die Tabellen sehr leicht nach einem unkoordinierten Abbruch in einen inkonsistenten Zustand geraten sein können. Ein nicht zu vernachlässigendes Detail an dieser Methode ist der Aspekt, daß sämtliche Caches und Puffer des Dateisystems und des MySQL-Servers nach einem solchen Failover erstmal „kalt“ sind und damit noch nicht die volle Leistungsfähigkeit nach einem Wechsel erreicht werden kann. Im Gegenzug stellt DRBD dank seiner synchronen Funktionsweise sicher, daß Daten konsistent und redundant gespeichert wurden und ein Datenverlust im Fehlerfall damit deutlich unwarscheinlicher ist.

Galera Cluster

Noch in der Entwicklungsphase befindlich ist die Replikationslösung „Galera“ der skandinavischen Firma Codership (http://codership.com/products/mysql_galera). Hierbei handelt es sich um ein System, das MySQL um eine multimaster-fähige, synchrone Replikationtechnologie auf Basis von InnoDB erweitert. Durch diesen Ansatz eignet sich Galera sowohl für Hochverfügbarkeitszwecke als auch zur Skalierung und Lastverteilung durch den Einsatz mehrerer MySQL-Server. Da die Replikation innerhalb der InnoDB Storage-Engine vorgenommen wird, ist sie vollständig transparent für die Applikation.

Um diese Funktionalität zur Verfügung zu stellen ist es erforderlich, einen Patch auf den InnoDB-Quelltext anzuwenden und den MySQL-Server damit neu zu übersetzen. Codership bietet von seinen Webseiten bereits fertig gepatchte und übersetzte MySQL-Pakete an. Zukünftig wird Galera auch mit dem InnoDB-Plugin kooperieren, so daß diese Funktionalität nachgerüstet werden kann, ohne den gesamten Server-Code kompilieren zu müssen – es wird lediglich der Plugin ausgetauscht und der MySQL Server neu gestartet.

Interessant an Galera ist der Umstand, daß der Großteil der Funktionalität in einer separaten Bibliothek (wsrep) ausgelagert ist – im InnoDB-Code befinden sich lediglich die Funktionsaufrufe an den relevanten Stellen. Die Replikation der Daten erfolgt beim Abschluß einer Transaktion nach einem zertifikat-basierten Schema. Aufgrund seiner Implementierung als externe Bibliothek ist Galera Datenbank-unabhängig ausgelegt; eine Portierung auf PostgreSQL ist ebenfalls angedacht. Zukünftig wäre es mit dieser Lösung theoretisch also sogar möglich, einen heterogenen Cluster bestehend aus MySQL und PostgreSQL-Datenbanken aufzubauen.

PBXT Replikation

Die PBXT Speicher-Engine wird von Paul McCullagh bei der Firma PrimeBase Technologies entwickelt und ist ein Speicher-Engine Plugin für MySQL 5.1 (http://www.primebase.org/). Sie basiert auf dem MVCC-Konzept und ermöglicht daher ein paralleles, blockierungsfreies Lesen von Tabelleninhalten. PBXT unterstützt Transaktionen und Foreign Keys und arbeitet log-basiert, d.h. Daten werden nur einmal und in sequenzieller Form in eine Log-Datei geschrieben. Das Log wird somit zum eigentlichen Speicherplatz der Tabellendaten.

PBXT wird ständig weiterentwickelt; in der Erprobungsphase für PBXT 2.0 befindet sich eine Möglichkeit, Daten mittels asynchroner Replikation auf Storage-Engine-Ebene zwischen einem Master und einem oder mehreren Slaves zu verteilen. Die Replikation erfolgt hierbei im "Push"-Betrieb – die „klassische“ MySQL-Replikation verwendet einen „Pull“-Ansatz. Die Replikation erfolgt zeilenbasiert, wobei hier ein PBXT-internes Zeilenformat und Tabellen-IDs verwendet werden, wie sie auch für die Speicherung auf Platte verwendet werden. Dies hat den Vorteil, daß die Replikation relativ wenig Overhead hat. Das Aufsetzen der Replikation erfolgt im Moment durch vollständiges Kopieren des Datenbankverzeichnisses auf dem Master auf den Replikations-Slave mit anschließendem Start beider Knoten. Obwohl sich diese Technologie noch in der frühen Entwicklungsphase befindet, ist der Ansatz vielversprechend.

Continuent Tungsten Replicator

Tungsten Replicator von Continuent (http://continuent.com/community/tungsten-replicator) ist eine weitere Replikationslösung die neben MySQL auch PostgreSQL und Oracle (in der Enterprise-Version) unterstützt. Der Replicator ist dabei eine Komponente einer Gesamtlösung, die noch aus weiteren Elementen besteht und stellt Methoden zur Datenbank-externen, asynchronen Master/Slave-

Replikation zur Verfügung. Die Software arbeitet betriebssystemunabhängig – sie benötigt zur Laufzeit lediglich eine Java Virtual Machine (JVM).

Die Replikationslösung erfordert keine Veränderungen am MySQL-Server und bietet eine logische Replikation der Daten (SQL-Anweisungen oder Tabellenzeilen). Tungsten arbeitet log-basiert, es liest die Binär-Logs des MySQL-Servers und analysiert und extrahiert neue Ereignisse während sie in die Log-Datei geschrieben werden. Zur Vermeidung von zusätzlicher I/O-Last auf einem MySQL-Master kann Tungsten Replicator auch mit Relay Logs arbeiten – hierzu verbindet er sich mit einem MySQL Master-Server wie ein herkömmlicher MySQL-Replikations-Slave und transferiert alle aufgelaufenden Änderungen zu sich.

Diese Vorgänge werden anschließend im Transaction History Log (THL) abgelegt und an die anderen Tungsten Replikator-Instanzen verbreitet. Das THL verwaltet eine durchgehende Liste von Replikationsereignissen in fortlaufender Reihenfolge, durch Anwenden derselben auf einem Slave-Server läßt sich damit der gleiche Inhalt wie der des Masters herstellen.

Bei den Replikations-Topologien sind sowohl klassisches „Fan-out“ (ein Master repliziert auf viele Slaves) als auch „Fan-in“ (Viele Master replizieren auf einen Slave) möglich. Letztere Art der Replikation ist in dieser Form mit MySQL-Replikation nicht realisierbar. Administriert wird ein solcher Replikationsverbund über eine kommandozeilenbasierte Konsole. Damit können die Rollen einzelner Knoten verändert werden (z.B. ein Slave zum Master befördern), neue Knoten hinzugefügt werden oder die Replikation zu einem bestimmten Knoten temporär ausgesetzt werden (z.B. zu Wartungszwecken).

Der Tungsten Replikator besitzt eine modulare Architektur: Daten können gefiltert und transformiert werden (über sogenannte „Pipelines“ und „Stages“). Im einfachsten Fall besteht ein Replicator nur aus einem Extractor der Logdaten ausliest und einem Applier, der sie wieder irgendwohin schreibt. Es können jedoch beliebige Filter dazwischengeschaltet werden, die den Datenstrom verändern.

Continuent Tungsten verfügt über diverse Hochverfügbarkeits- und Management-Features (z.B. automatisches Failover), diese sind allerdings der Enterprise-Version vorbehalten.

Kontakt:

Lenz Grimmer
Oracle Deutschland B.V. & Co. KG
Döringweg 7a
22529 Hamburg

Phone: +49(0)40-4126 7308
Fax: +49(0)40-4126 7311
Email lenz.grimmer@oracle.com
Internet: http://www.mysql.com/