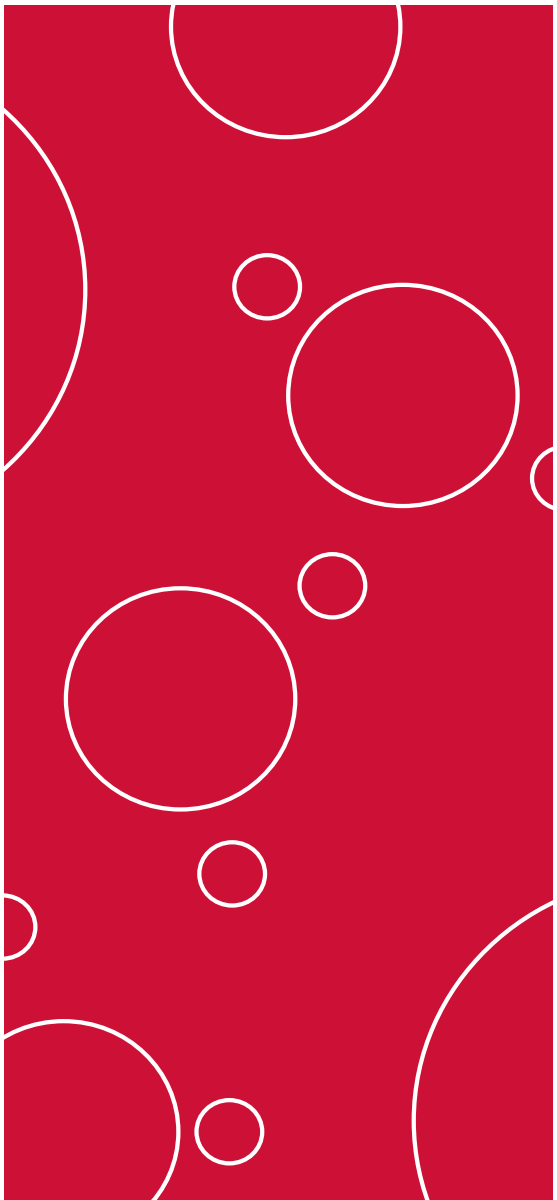
MySQL replizieren



Steigen die Anforderungen an eine MySQL-Datenbank, suchen die Admins oft nach einer Möglichkeit, Leistung und Verfügbarkeit gleichzeitig zu erhöhen. Ein bewährtes Mittel, um dieses Ziel zu erreichen, ist die Replikation der Datenbank auf mehrere Hosts. Dafür bieten sich verschiedene Verfahren an – aber welches ist wann das richtige? Wie steht es um die Einstiegskosten oder die Handhabung in den einzelnen Fällen? Welche Stolpersteine gibt es wo und wie wären sie zu umgehen? Ist ein Upgrade auf eine höhere MySQL-Version sinnvoll oder gar notwendig? Das alles erörtert dieser Beitrag an Beispielen. Dennis Brandenburg

Jede Synchronisation birgt grundsätzlich das Risiko von Divergenzen zwischen den Beteiligten. Im Fall der MySQL-Replikation kommt aber als weitere Schwachstelle hinzu, dass abhängig vom Replikationskonzept bestimmte Prozeduren geradezu zu einer solchen Abweichung führen müssen. Nur wer sich dieser Schwachstellen bewusst ist, kann sie vermeiden oder zumindest ihre Auswirkungen minimieren. Dafür bieten sich verschiedene Techniken an, etwa die Replikation auf Blockebene, der Einsatz einer speziellen Speicher-Engine oder unterstützende Tools.

Grundlagen der MySQL-Replikation

In der klassischen MySQL-Replikation finden alle Schreibzugriffe auf der Originaldatenbank (dem Master) statt. Anschließend gelangen sie asynchron auf eine oder mehrere Replikate – die so genannten Slaves. Voraussetzung hierfür ist, dass Master und Slave zu Beginn über einen identischen Datenbestand verfügen. Der Master schreibt jede Datenbankänderung in ein Binär-Log. Jeder Slave liest kontinuierlich über eine Socketverbindung des MySQL-Daemons aus dem Binär-Log des Masters (Binlog Dump Thread) und überträgt alle Einträge in ein lokales Relay-Log (dies geschieht im sogenannten I/O-Thread). Durch „Abspielen“ dieses Logs werden die Änderungen auf den Slaves übernommen (SQL-Thread), wodurch sich Master und Slave synchronisieren.

Bei dieser asynchronen Replikationstechnik kontrolliert der Master die replizierten Ergebnisse auf den Slaves nicht und bietet keine Möglichkeit, Abweichungen durch fehlgeschlagene oder fehlerhafte Änderungen in der Replikation festzustellen und darauf zu reagieren. Die Synchronisierung der MySQL-Replikation unterliegt also keiner Integritäts- und Konsistenzprüfung. Der Synchronisationszustand der Slaves ist zudem abhängig vom zeitlichen Abstand der Änderungen auf Master und Slave, den deshalb ein Monitoringsystem überwachen sollte.

Der entsprechende Wert »seconds behind master« wird mit dem Kommando »show slave status« auf der MySQL-Konsole des Slaves abgefragt und beträgt im Idealfall null Sekunden. Nimmt dieser Wert zu, sind oftmals aufwändige Update-Kommandos und Transaktionen die Ursache. In diesem Fall sind Master und Slave zumindest vorübergehend nicht mehr synchron. Das kommt daher, dass der Master erst nach erfolgreicher Beendigung eines Kommandos die Änderungen in das Binär-Log schreibt und damit die Daten serialisiert. Dagegen laufen schreibende Kommandos auf dem Master parallel ab. Zudem können die zu replizierenden Tabellen dem SQL-Thread den Zugriff mit Hilfe von Locks verwehren, was den zeitlichen Unterschied zwischen Master und Slave weiter vergrößert. In den meisten Fällen schafft es der Slave allerdings, die Lücke wieder zu schließen. Dennoch kann nur der Master für die aktuellsten Daten garantieren.

Häufige Anwendungsfälle

Abhängig vom Anwendungsfall kann die MySQL-Replikation effizient zur Synchronisierung genutzt oder auf Grund der beschriebenen Schwachstellen zu einem konzeptionellen Problem werden. Nicht für jedes Szenario eignet sie sich optimal. Dabei kann man Einzelnen die folgenden Fälle unterscheiden:

Failoverlösungen: In zeitkritischen Anwendungsfällen wie Failoverlösungen eignen sich die asynchronen Techniken der MySQL-Replikation häufig nicht, dort können auf Grund des beschriebenen zeitlichen Versatzes zwischen Master und Slave SQL-Kommandos kollidieren und Transaktionen verloren gehen. Deshalb ist es ratsam, dafür synchrone Replikationstechniken wie die Netzwerkspeicherlösung DRBD oder die Speicher-Engine NDB zu verwenden. Einen Kompromiss bietet die kürzlich als stabil freigegebene MySQL-Version 5.5 mit einer semi-synchronen Replikation.

Backupstrategien unterstützt die MySQL-Replikation besonders gut, denn der Slave lässt sich problemlos anhalten und dann sichern. Verwendet man außerdem Snapshots des LVM (Logical Volume Manager), ist dies im laufenden Betrieb möglich. Das Binärlog ist zudem dafür verwendbar, um nach einer Wiederherstellung der letzten Vollsicherung „vorzuspulen“. Slaves lassen sich zudem als „Lazy Slave“ künstlich in zeitlichem Versatz zum Master halten, damit sich mögliche logische Fehler erkennen lassen, bevor sie im Replikat landen.

Skalierung: Für die Skalierung lesender Datenbankzugriffe ist die MySQL-Replikation ebenfalls ein geeigneter und häufiger Anwendungsfall. Die zeitlichen Divergenzen der Slaves können vorgeschaltete Loadbalancer überwachen. Ein für die Anwendung tolerierbarer Schwell-

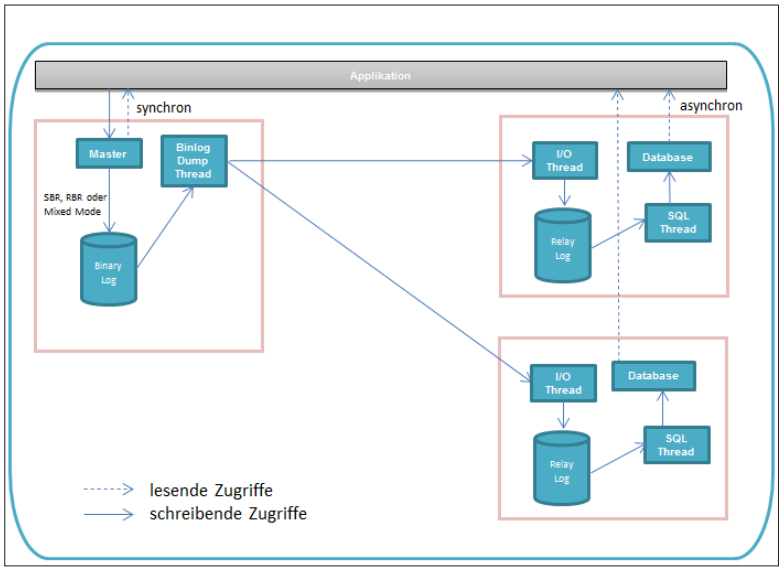


Abbildung 1: Nach diesem Schema funktionieren die verscheidenenen Spielarten der MySQL-Replikation im Prinzip.

wert entscheidet dann darüber, auf welche Slaves die Anwendung zugreift.

Unter dem Dach der asynchronen Replikationstechniken kommen die drei im Folgenden beschriebenen Konzepte zum Tragen. Diese Konzepte unterscheiden sich jeweils in der Art des von ihnen verwendeten Binärlog-Formats und können abhängig vom Anwendungsfall unterschiedliche Synchronizitätsprobleme hervorrufen oder solche Probleme gegebenenfalls auch verringern.

Anweisungs- oder datensatzbasiert replizieren

Die anweisungsbasierte Replikation (Statement-Based Replication, SBR) ist vermutlich auf Grund der Kompatibilität zu älteren MySQL-Versionen in den meisten Installationspaketen standardmäßig aktiv und ist seit ihrer Implementierung in der MySQL-Version 3.23 eine gegenüber anderen Konzepten sehr effiziente und performante Alternative. Dieses Konzept erlaubt allerdings unzählige Situationen, die einen inkonsistenten Datenbestand auf den Slaves hinterlassen. Grund hierfür ist, dass ausschließlich SQL-Kommandos repliziert werden, die auf den Slaves unter Umständen nicht so wie auf dem Master ausführbar sind.

Die zu replizierenden SQLStatements werden vom Master serialisiert in das Binärlog geschrieben, wodurch sich die Reihenfolge der Ausführung in verschachtelten Transaktionen auf den Slaves verändern kann. Das Ergebnis kann ein vom Master abweichender Datenbestand sein, der sich unbemerkt und ohne Fehlermeldungen einschleicht. Zudem existieren zahlreiche weitere Schwachstellen in der SBR, die durch spezielle Situationen besonders in Triggern und Stored Procedures zu unterschiedlichen Ergebnissen auf den Replikaten führen können.

Tabellen der Slave-Datenbanken können bei dieser Methode also generell falsche Ergebnisse zurück liefern, weil sie nicht mehr synchron zum Master sind. Das kann gerade auch dort problematisch sein, wo der Admin die Master-Datenbank aus einem Backup des Slaves wiederherstellen möchte. In einigen Fällen ist es deshalb effizienter die MySQL-Replikation datensatzbasiert (Row-Based Replication, RBR, seit MySQL 5.1 implementiert) zu betreiben, was die genannten Probleme vermeidet. Die Einstellung »binlog-format=row« bewirkt hier, dass anstelle der SQL-Kommandos die Veränderungen der Datensätze selbst in das Binärlog gelangen. Dies kann allerdings besonders bei großen Update-Kommandos, die eine Vielzahl an Zeilen ändern, zu einem deutlichen Anwachsen der Binärlogs führen und zudem die Performance der Datenbank mehr oder minder stark beeinträchtigen.

Der Mixed Mode (ebenfalls seit MySQL 5.1 implementiert) bildet einen Kompromiss der beiden genannten Konzepte. Grundsätzlich wird in diesem Modus die SBR verwendet, aber in speziellen Fällen automatisch auf die RBR umgeschaltet. Beispielsweise liefert die MySQL-Funktion »uuid()« eine eindeutige 128-Bit-Zahl. Repliziert man ein Kommando, das dieses Statement enthält, muss es auf jedem Slave ein anderes Ergebnis liefern. Der Mixed-Mode repliziert Statements mit solchen Kommandos wie »uuid()« deswegen automatisch datensatzbasiert. Ähnliche Fälle, die der Mixed-Mode explizit datensatzbasiert bearbeitet, sind Update-Kommandos auf MySQL-Clustertabellen, Änderungen mehrerer AUTO-INCREMENT-Zeilen oder INSERT-DELAYED-Kommandos. Eine ausführliche Liste der datensatzbasiert ausgeführten Kommandos und Funktionen ist im MySQL Reference Manual enthalten. Der Mixed Mode kann also eine gute Wahl für den zuverlässigen Betrieb in den Fällen sein, wo die anweisungsbasierten MySQL-Replikation problematisch wäre. Performanceprobleme durch stark beanspruchte Binärlogs der rein zeilen-

basierten MySQL-Replikation lassen sich dabei vermeiden.

Semisynchrone MySQL-Replikation

In der neuen MySQL-Version 5.5 wurde die schon seit längerem von Google als Patch verfügbare semisynchrone MySQL-Replikation integriert. Diese Technik basiert darauf, dass ein erfolgreicher Schreibvorgang auf dem Master der Applikation erst dann bestätigt wird, wenn mindestens einer der als semi-synchron konfigurierten Slaves das daraus resultierende Event in das lokale Relay-Log geschrieben und diese Änderungen auf die Festplatte geflusht hat. Anhand eines Timeouts lässt sich definieren, wie lange der Master auf die Bestätigung eines Slaves wartet, bevor er der Anwendung den erfolgreichen Schreibvorgang bestätigt. Ist dieser Wert in einer Transaktion überschritten, wird die Replikation automatisch asynchron vorgenommen. Die Statusvariable »Rpl_semi_sync_master_no_tx« gibt Auskunft über die Anzahl nicht bestätigter Anweisungen und ist ein Indikator zur Überwachung der semisynchronen Replikation.

Synchrone Replikationstechniken

Eine zuverlässige synchrone Replikationstechnik ermöglicht das Distribuited Replicated Block Device (DRBD) vorwiegend für Failoverkonzepte als Active-Passive-Cluster, weil es eine Synchronisation im Echtzeitbetrieb zur Verfügung stellt. Das DRBD synchronisiert seine Datenpartitionen auf Blockebene via Netzwerk und benötigt dafür kein gemeinsam genutztes SAN oder NAS (Shared-Nothing-Lösung). Damit stellt das unter der GPL2 lizenzierte DRBD-Modul eine kostengünstige und zuverlässige Replikationstechnik dar, die sich hervorragend eignet, um eine hochverfügbare MySQL-Datenbank zu betreiben.

Das DRBD-Modul ist seit der Kernelversion 2.6.33 fester Bestandteil des Kernels und wird häufig als Netzwerk-RAID-1 bezeichnet. DRBD unterstützt asynchrone, semisynchrone und synchrone Replikationstechniken. Im asynchronen Modus werden Schreibvorgänge als abgeschlossen gewertet, wenn sie sich nach dem lokalen Schreibvorgang des primären Systems bereits im lokalen TCP-Send-Buffer auf dem Weg zum sekundären (synchronisierenden) System befinden. Im semi-synchronen Modus hingegen gelten Schreibvorgänge als abgeschlossen, wenn sich die zu replizierenden Daten im Arbeitsspeicher des sekundären Systems (aber noch nicht zwingend auf dessen Platte) befinden. Da der Zugriff auf dem Plattenspeicher in der Regel der aufwändigste aller Vorgänge ist, erreichen diese beiden Verfahren eine gute Performance, können allerdings bei einem Stromausfall oder bei Hardwareproblemen einen nicht-synchronen Datenbestand hinterlassen.

Mit dem synchronen Verfahren werden die geschriebenen Daten erst dann als abgeschlossen gewertet, wenn der Schreibzugriff sowohl auf dem primären als auch auf dem sekundären Plattensubsystem des DRBD-Clusters stattgefunden hat. Dies ermöglicht einen fortlaufend synchronen Datenbestand, der einen Datenverlust in Failoversituationen minimiert. Dieses Verfahren bietet sich vor allem für die angesprochenen Hochverfügbarkeitslösungen an, die MySQL zusammen mit einem Cluster Ressource Manager (CRM) wie Pacemaker/Corosync oder Heartbeat absichern. Die Performance von MySQL innerhalb eines DRBD-Clusters ist stark von der Anwendung und vom Verhalten der Datenbank abhängig. Im Produktivbetrieb ist immer auf eine zuverlässige Netzwerkverbindung für die Synchronisierung am besten im Gigabit-Bereich zu achten. Oft bilden auch die Festplatten im sekundären System einen Engpass, weshalb man auch dort auf entsprechend potente Hardware Wert legen sollte.

MySQL-Cluster

Eine weitere Möglichkeit für eine synchrone Replikation von MySQL-Datenbanken bietet der MySQL-Cluster (aktuell in der Version 7.1). Sein Kernbestandteil ist die Speicher-Engine NDB Cluster, die es ermöglicht, die enthaltenen Daten zwischen den Datenknoten synchron zu replizieren (**Abbildung 1**). Die NDB-Datenknoten laufen als Prozess (oder als mehrere Prozesse) auf unterschiedlichen Hosts. Die Schnittstelle zu diesen Datenknoten bilden SQL-Knoten in Form von MySQL-Servern. Hinzu kommen Managementdienste als Konfigurations- und Überwachungskomponenten des Clusters. NDB ist speicherresistent konzipiert. Die Replikation findet zunächst synchron im Arbeitsspeicher statt, während die persistente Datenspeicherung anhand definierter Checkpointintervalle asynchron abläuft.

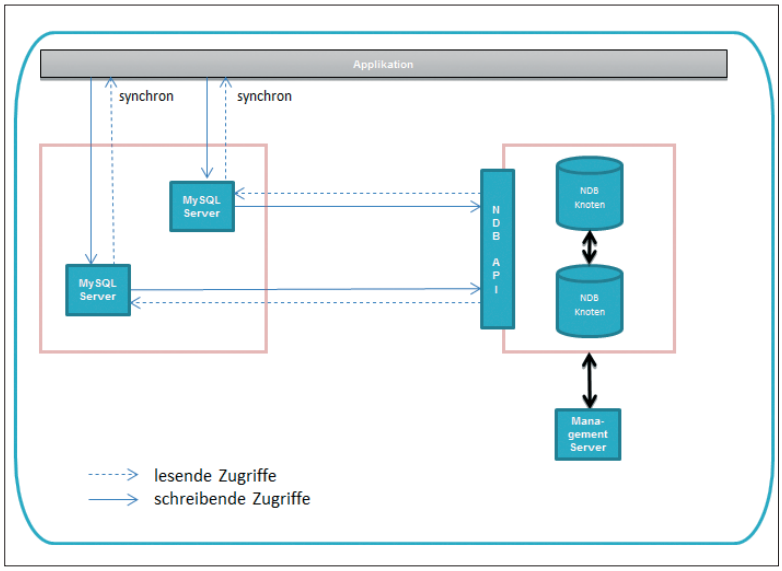


Abbildung 2: Das Funktionsprinzip eines MySQL-Clusters mit der speziellen NDB-Engine.

Vor diesem Hintergrund eignet sich diese Art der Replikation besonders gut für temporär erstellte und dynamische Inhalte wie den Aufbau eines Warenkorbs im E-Commerce- Shop, in dem Inhalte stark variieren und erst nach Bestätigung des Einkaufs persistent benötigt werden. Der Cluster skaliert nahezu linear, da NDB-Knoten beliebig (versionsabhängig, bis zu 255 Knoten) hinzugefügt und entfernt werden können. Dabei werden Datenbanken und Tabellen automatisch auf die vorhandenen NDB-Knoten verteilt.

Große Abfragen mit mehrfachen Joins auf Tabellen werden vom MySQL-Optimizer der aktuellen Version 5.5 allerdings nicht immer optimal verarbeitet und können auf Grund des implementierten Nested-Loop-Join-Algorithmus unter bestimmten Umständen zu einer großen Anzahl von Round Trips im NDB-Cluster und somit zu verzögerten Abfrageergebnissen führen. In späteren MySQL-Releases stehen neue Algorithmen wie BKA (Batched Key Access) oder SPJ (Push Down Joins) zur Optimierung zur Verfügung. Obwohl der NDB-Cluster transaktionsfähig ist, wird ausschließlich die Isolationsstufe »read commited« unterstützt. Mit der fehlenden Unterstützung für Foreign Key Contrains, Index Prefixe, Optimize-Operationen, sowie Savepoints und Rollbacks sind nur einige Einschränkungen genannt, die bei der Nutzung von NDB beachtet werden müssen.

Synchronizität prüfen

Besonders in der anweisungsbasierten MySQL-Replikation kann es wie beschrieben zu Synchronizitätsproblemen kommen, die oftmals unbemerkt Abweichungen zwischen den Replikaten verursachen. Dennoch wird diese Technik oft aufgrund ihrer Flexibilität zur Skalierung oder zur Realisierung von Backup- und Failoverlösungen benutzt. In diesen Fällen bieten sich regelmäßige Prüfungen der Slaves auf abweichende Datenbestände an. Dafür kann durch die Entwicklung geschickter Abfragen, wie durch das Zählen von Datensätzen einer Tabelle mit der Funktion »count()« oder das Erstellen von Tabellen-Prüfsummen mit »CHECKSUM TABLE« sowie der Funktion »CRC32()« die Synchronizität der Replikation festgestellt werden.

Je größer die Tabellen, desto aufwändiger fallen die Prüfungen allerdings oft aus, so dass sie ihrerseits merkliche Performanceeinbußen verursachen können. Kommen Abweichungen zu Tage, die der Admin resynchronisieren will, muss die Replikation gestoppt und durch Kopieren des Data-Directories vom Master auf den Slaves neu aufgebaut werden. Häufig kann allerdings die damit einhergehende Zwangspause nicht in Kauf genommen werden.

Abhilfe schaffen dann die aus dem Maatkit Toolkit stammenden und unter der GPL2 lizenzierten Perl-Programme »mk-table-checksum« und »mk-table-sync«. Das Tool »mk-table-checksum « erzeugt Prüfsummen von Tabellen, wenn nötig in Chunks, um langlaufende Locks zu vermeiden. Die Ergebnisse werden in eine dafür vorgesehene Prüfsummen-Tabelle geschrieben und stehen zur Auswertung bereit. Besonders vorteilhaft ist die Möglichkeit der Kapselung der Prüfsummen-Funktionen in schreibenden SQL-Kommandos. So lässt sich »mk-table-checksum« gut in die MySQL-Replikation integrieren, da die Prüfsummen auf allen Slaves an einer bestimmten Binärlogposition zum gleichen Prüfzeitpunkt erstellt werden. Die Replikation beeinträchtigt das so gut wie nicht. Ergänzend kann »mk-table-sync« auf Basis dieser Prüfsummen-Tabellen eine Resynchronisierung anstoßen.

Die in Perl geschriebenen Programme des Maatkits sind im Repository einiger Distributionen bereits inklusive aller weiteren Maatkit-Tools als Paket enthalten und lassen sich von (1) einzeln herunterladen. Sie benötigen einen Perl-Inter-

preter und grundlegende Paketen inklusive DBI und DBD::mysql. Um die Synchronizität mit »mk-table-checksum« und »mk-table-sync« zu überprüfen und wiederherzustellen, bedarf es einer Prüfsummentabelle, die auf dem Master erstellt und auf die Slaves repliziert wird.
Die von »mk-table-checksum« erstellten Prüfsummen werden später noch einmal auf jedem Slave erzeugt und in die Prüfsummentabelle geschrieben. Als Ergebnis enthält diese Tabelle auf jedem einzelnen Slave die Prüfsummen aller Tabellen des Masters, sowie die der eigenen Replikate dieser Tabellen. Ein Vergleich beider Werte gibt Aufschluss darüber, ob die Tabellen synchron sind oder nicht. Ist die MySQL-Replikation unter Verwendung der Optionen »replicate-do-db«, »replicate-do-table«, »replicate-ignore-db« oder »replicate-ignore-table« konfiguriert, kann es zu Problemen mit »mk-table-checksum« kommen, die im schlimmsten Fall die Replikation unterbrechen. Daher ist es empfehlenswert sich die Dokumentation von »mk-table-checksum« (2) zu den Optionen »--replicate« und »--replicate-database« genauer anzusehen und ausgiebig zu testen.
Die Option »--replicate«, die die Prüfsummen in schreibenden Statements verpackt, löst auf jedem Slave die Berechnung der Prüfsummen aus und schreibt sie in die jeweilige Tabelle. Dies erledigt das folgende Kommando auf dem Master:

```
mk-table-checksum --replicate=test ➥
checksum --ask-pass h=localhost, u=root,➥
S=/var/lib/mysql/mysql.sock
```

Dabei wird die Prüfsummen-Tabelle (default ist »test.checksum«) angegeben, die lokalen Zugangsdaten transportieren die Optionen »h« (Host), »u« (Benutzername) und »S« (Socket). Die Option »ask-pass« fragt interaktiv nach dem Passwort des lokalen MySQL-Zugangs und kann lässt sich durch die Option »P« mit Angabe des Passworts ersetzen. Bei der Prüfsummenberechnung von großen Tabellen kann es bei MyISAM-Tabellen zu langlaufenden Read-Locks kommen. An dieser Stelle ist es möglich Prüfsummen von Tabellen in Teilen zu erstellen. So ermöglicht die Option »chunks« die Berechnung von Prüfsummen für eine bestimmte Anzahl von Datensätzen oder Datenmengen.

Resynchronisieren

Sind sowohl auf dem Master als auch auf den Slaves alle SQL-Kommandos zur Aufnahme der Prüfsummen durchgelaufen, können die Differenzen von Master und Slave mit einer Abfrage der Prüfsummentabelle auf jedem Slave ausgegeben werden. Als Ergebnis werden eine Liste von Tabellen und deren Prüfsummen, sowie die Anzahl der zwischen Master und dem Slave differierenden Datensätzen angezeigt. Sind die Differenzen festgestellt, lassen sich Tabellen auf den Slaves mit »mk-table-sync« gezielt resynchronisieren und auf den Stand des Masters bringen. Die Optionen »verbose« und »dry-run« erlauben es den Synchronisierungsvorgang zu testen. Bei einer besonders kleinen Anzahl an Differenzen ist alternativ MySQLDump in Verbindung mit dem Programm »diff« zur manuellen Resynchronisierung nutzbar. Bei einer besonders großen Anzahl differierender Tabellen mit vielen komplexen Foreign Keys empfiehlt es sich hingegen, den gesamten Datenbestand von der Master-Datenbank neu zu kopieren.

Fazit

MySQL bietet versionsabhängig viele Möglichkeiten zur Optimierung und Verbesserung der Replikation. Als Alternative zum neuesten Release 5.5 oder ergänzend sind MySQL-Cluster, DRBD und die Maatkit Tools effizient nutzbar, um eine zuverlässige Synchronizität für hoch verfügbare und performante Anwendungen zu erreichen.

Zudem gehört das Überprüfen der Synchronisierung von Slaves einer MySQL-Replikation oft zum Pflichtprogramm des Datenbankadmins. Die beschriebenen Tools können helfen, diese Aufgabe effizient zu bewältigen. Darüber hinaus bieten »mk-table-checksum« und »mk-table-sync« auch unabhängig von der Replikation Möglichkeiten zur Ermittlung und Synchronisierung von differierenden Tabellen. Das Maatkit-Toolset (1) hält zudem zahlreiche weitere Tools zur Entwicklung und Administration von Open Source-Datenbanken bereit. *(jcb)* ■■■

Infos

(1) Maatkit: (http://www.maatkit.org)
(2) mk-table-checksum: (http://www.maatkit.org/doc/mk-table-checksum.html)