MySQL HA – Lösungen für Back- und Frontend

Matthias Klein, InnoGames GmbH

Bereits seit dem Jahr 2001 bietet MySQL Replikations-Lösungen an. Während diese auf der Datenbank-Ebene bereits sehr ausgereift sind und stetig verbessert werden, muss man zur Anbindung seiner Anwendung meist noch auf Software von Drittanbietern zurückgreifen. Je nach Anwendungsfall ergeben sich hier unterschiedliche Möglichkeiten und Empfehlungen.

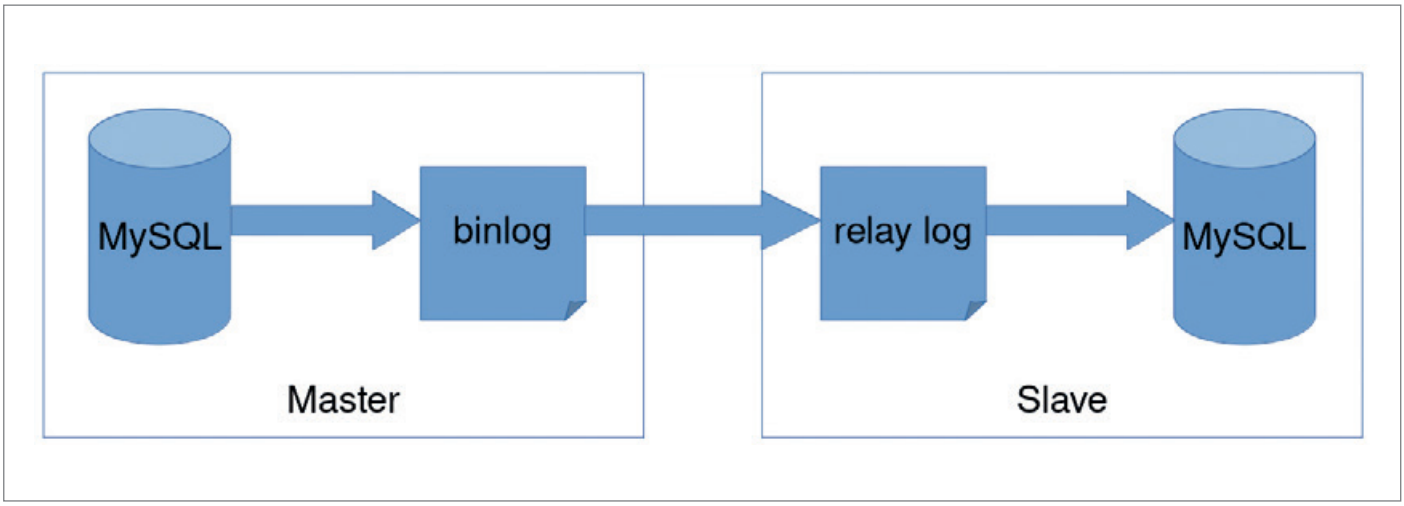

Abbildung 1: Asynchrone Replikation

Schon mit Einführung der Replikation vor etwa siebzehn Jahren konnte man zwischen einer synchronen und einer asynchronen Variante wählen. Während die synchrone Variante mit der „ndb"-Engine sehr spezielle Voraussetzungen bezüglich Hardware und an die eingesetzte Anwendung hat, funktioniert die asynchrone Replikation grundsätzlich mit allen Engines.

In der jüngeren Vergangenheit hat Codership Oy die Galera Cluster entwickelt. Damit stand erstmals eine synchrone Variante der Replikation zur Verfügung, die sich durch Unterstützung der InnoDB-Engine und eine recht einfache Konfiguration auszeichnet. Die Einschränkungen hinsichtlich der zugreifenden Anwendung sind im Gegensatz zu „ndb" nicht mehr existent, auch ist ein Betrieb mit schwächerer Hardware möglich.

Mit MySQL 5.7 führte Oracle ebenfalls eine zusätzliche Cluster-Variante (Group Replication, GRE) ein. Diese integriert sich besser in die bisher bekannten Abläufe, so kann sie zum Beispiel analog zur asynchronen Replikation mit „START GROUP_REPLICATION" gestartet werden. Um aber eine Anwendung an die Datenbank anbinden zu können, kann man (mit Ausnahme von MySQL Fabric) auf eine Software von Drittanbietern zurückgreifen. Die meisten dieser Varianten haben allerdings den Nachteil, dass sie nicht von sich aus hochverfügbar sind. Bei einigen Varianten vermindert das die Ausfallsicherheit nicht wesentlich, bei anderen jedoch enorm. Welche Maßnahmen dagegen ergriffen werden können, ist abschließend aufgezeigt.

Asynchrone Replikation

Diese Art der Replikation zeichnet sich dadurch aus, dass sie unabhängig von der verwendeten Datenbank-Engine arbeitet. Es sind sogar unterschiedliche Engines auf Master und Slave für eine Tabelle möglich. Außerdem besteht die Möglichkeit, Datenbanken und/oder Tabellen von der Replikation auszuschließen, diese nur auf bestimmten Replikaten zu replizieren oder auf dem Replikat einen anderen Namen zu verwenden.

Die asynchrone Replikation verwendet „binlog" auf dem Master und „relaylog" auf dem Slave, um die Änderungen zu übertragen. Dabei wird nach erfolgreichem Anwenden einer Änderung in der Datenbank diese auf dem Master in das „binlog" geschrieben. Der Slave schreibt nun diese Änderungen in sein „relaylog" und wendet sie lokal an.

Dies kann über das ausgeführte SQL (Statement-based) geschehen oder es werden die Zeilen vor und nach der Änderung (Row-based) übertragen. Generell benötigt die Statement-Methode weniger Platz in den Logs, ist aber fehleranfällig, wenn „LIMIT"- oder „GROUP"-Funktionen zum Einsatz kommen. Hier besteht die Möglichkeit, mittels „MIXED" derartige Statements Row-based zu replizieren. Die Row-based Replikation benötigt zwar mehr Platz in den Logs, jedoch muss der Slave das SQL nicht mehr ausführen und kann die Änderungen direkt anwenden *(siehe Abbildung 1)*.

Aus dieser Art der Replikation ergeben sich hauptsächlich Probleme mit der Anwendung der Änderungen auf dem Slave. So kann es hier zu Verzögerungen kommen, da der Slave nur einen Thread zum Anwenden der Änderungen benutzen kann, der Master jedoch mit mehreren gleichzeitig Änderungen durchführt. Hier kann man sich zwar mit der Option „slave_parallel_workers" behelfen, jedoch nutzt das nur dann, wenn mehrere Datenbanken repliziert werden.

Auch ist es per Default möglich, auf einen Slave zu schreiben. Dies kann dazu führen, dass der Slave die Replikation unterbricht und der Administrator entscheiden muss, welcher Datenstand der richtige ist. Allerdings kann dies auch unbemerkt passieren, wenn dieser Datensatz nicht wieder angefasst wird. Um dies zu verhindern, sollten alle Slaves „read_only" (ab Version 5.7 auch „super_read_only") aktiviert haben.

Synchrone Replikation

Obwohl die „ndb"-Engine die älteste Form der synchronen Replikation von MySQL ist, führt sie eher ein Nischendasein. Dies liegt vor allem an den speziellen Anforderungen, die sie an Hardware und Anwendung stellt, um noch performant zu sein. Die Daten sind über mindestens zwei Knoten verteilt, wovon einer immer eine Kopie der Daten hält.

Die „ndb"-Engine partitioniert die Daten automatisch auf die vorhandenen Knoten. Zusätzlich ist noch mindestens ein System für das Management erforderlich sowie mindestens ein SQL-Knoten, der die Anfragen der Anwendung entgegennimmt. Konnten die Daten ur-

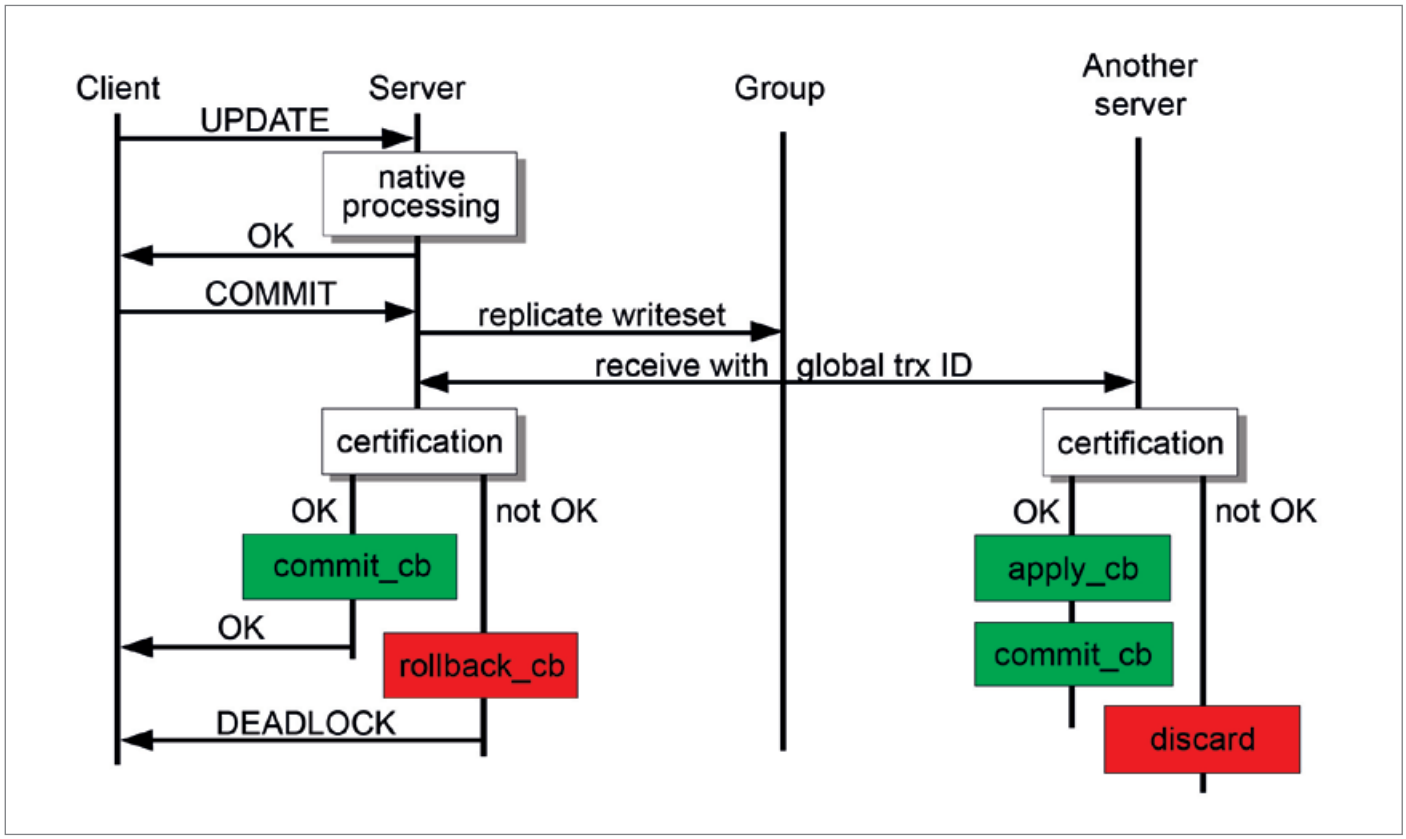

Abbildung 2: Schreib-Prozess bei Galera Cluster

sprünglich ausschließlich im RAM der Datenknoten abgelegt werden, ist das mittlerweile auf Kosten der Performance auch auf Festplatte möglich.

Bei der „ndb"-Engine bestehen meist Performance-Probleme. Gerade bei vielen Datenknoten sind die Daten stark partitioniert, was zum Beispiel Joins extrem teuer macht. Diese Operationen werden von den Datenknoten an den SQL-Knoten hochgereicht, der dann das endgültige Ergebnis noch berechnen muss. Dies lässt sich nur durch entsprechende Anpassung des Schemas und der Anwendung verhindern. Ebenfalls problematisch sind Schema-Änderungen, die erst seit der „ndb"-Version 7.5 online erfolgen können. Verglichen mit InnoDB dauern diese auch deutlich länger.

Seit etwa fünf Jahren gibt es mit Galera Cluster eine stabile synchrone Replikation, die auf InnoDB basiert. Oracle hat mit MySQL 5.7 die Group Replication (GRE) eingeführt. Beide Methoden unterscheiden sich in den technischen Details, die Konzepte sind jedoch so ähnlich, dass beide zusammen betrachtet werden können.

Die Anwendung der Änderung im Cluster erfolgt in einem mehrstufigen Prozess. Dabei werden die Änderungen erst lokal auf dem Server verarbeitet. Sobald der Commit erfolgt, findet auf den anderen Servern ein Zertifizierungsprozess statt. Dieser prüft, ob die Änderungen anwendbar sind, und sendet das Ergebnis zurück. Da hier tatsächlich keine Daten geschrieben werden, ist er sehr schnell.

Lassen sich die Änderungen auf allen Servern durchführen, werden dem Client der Commit bestätigt und die Daten endgültig geschrieben, im Fehlerfall die Daten nicht geschrieben und der Client bekommt eine entsprechende Meldung. Hier zeigt sich der Unterschied zwischen Galera und GRE deutlich: Während Galera nur ein generisches DEADLOCK zurückliefert, konnte Oracle entsprechende Fehlercodes implementieren *(siehe Abbildung 2)*.

Ob sich am Ende Galera oder GRE durchsetzen oder beide eine Koexistenz führen werden, ist im Moment noch unklar. Galera profitiert durch den früheren Start. So konnte Codership bereits mehr Erfahrungen sammeln und seine Implementierung weiter verbreiten. MySQL punktet durch die direkte Implementierung im Code, was sich durch eine angenehmere Bedienung über „mysql cli" zeigt. Es ist davon auszugehen, dass Oracle fehlende Features in naher Zukunft nachrüsten wird. So kann man bei Galera mit einem Arbitrator Ressourcen sparen. Auch gibt es hier Möglichkeiten, die Kommunikation des Clusters über WAN-Verbindungen zu beschleunigen. Allerdings ist bei GRE die Kommunikation des Clusters untereinander deutlich Ressourcen-schonender und wird mit mehr Servern auch nicht spürbar langsamer.

Synchrone Replikation sollte immer mit einem dedizierten Server für Schreibzugriffe betrieben werden. Obwohl die Cluster-Software das zeitgleiche Schreiben auf unterschiedlichen Cluster-Nodes unterstützt, bringt dies am Ende keine Vorteile. Da die Daten auf jedem Node geschrieben werden müssen, werden Schreib-Operationen nicht schneller. Je mehr gleichzeitig auf unterschiedliche Nodes geschrieben wird, desto höher ist die Chance, dass der Zertifizierungsprozess fehlschlägt. Auch sollte darauf geachtet werden, dass die zu schreibenden Änderungen möglichst klein sind, da sonst der Zertifizierungsprozess und die Anwendung der Änderung sehr langsam werden, was unter Umständen den kompletten Cluster blockieren kann. Die bei Galera empfohlenen Grenzen liegen hier

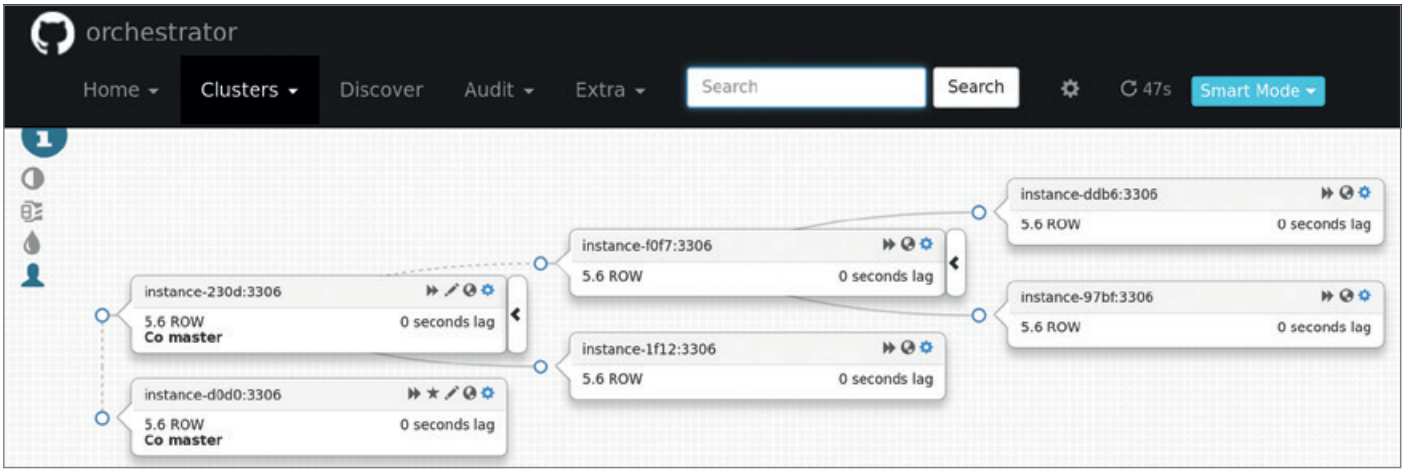

Abbildung 3: Orchestrator-Web-Oberfläche

bei 128 KB pro Zeile beziehungsweise 1 GB pro Transaktion.

Ein wichtiger Faktor beim Einsatz von Cluster-Systemen ist die Hardware. Der gesamte Cluster ist nur so schnell wie sein schwächstes Mitglied. Es wird auch eine ungerade Anzahl von Nodes (mindestens drei) empfohlen, um Split-Brain-Szenarien vorzubeugen. Dabei handelt es sich um Situationen, in denen die eine Hälfte der Nodes nicht mehr mit der anderen kommunizieren kann. Dabei wird der Cluster funktionsunfähig, da die Nodes keine Mehrheit für die Masterwahl mehr herstellen können.

Failover mittels MySQL Fabric

Die Hochverfügbarkeit auf der Datenbank-Ebene ist einfach herzustellen und arbeitet stabil. Um jedoch die Anwendung anzubinden, ist in den meisten Fällen Drittanbieter-Software oder eigene Programmierarbeit notwendig.

Oracle selbst bietet mit MySQL Fabric eine eigene Lösung an. Diese erfordert allerdings einen speziellen Treiber für die Anwendung und ist nur für asynchrone Replikation geeignet. Auch der Management-Server, der die Koordination übernimmt, ist selbst nicht hochverfügbar. Dessen Ausfall würde den alten Status beibehalten. Ein zusätzlicher Ausfall des Masters würde zu einer Downtime führen, was jedoch recht unwahrscheinlich ist.

Neben dem Routing kümmert sich Fabric auch um die Wiederherstellung der Replikation. So wird im Fehlerfall aus einem Pool von Servern ein neuer Master bestimmt und alle verbliebenen Server als dessen Slave konfiguriert. Unschön ist an dieser Stelle, dass der ausgefallene Server nach seiner Wiederherstellung manuell in Fabric wieder aktiviert werden muss. Damit ist man zwar auf der sicheren Seite, wenn man auf diesem Server Daten wiederherstellen musste; wenn jedoch nur ein Neustart etwa aufgrund von Softwareaktualisierungen stattgefunden hat, ist das Ganze recht umständlich.

Failover mittels MHA oder Orchestrator

Einen ähnlichen Weg gehen Master High Availability Manager (MHA) und Orchestrator. Beide Tools können asynchrone Topologien managen, also im Fehlerfall einen neuen Master bestimmen und die Slaves entsprechend konfigurieren. Beide benötigen eine Management-Instanz. Nicht per Default integriert ist allerdings, die Anwendung über die Topologie-Änderung zu informieren (wie bei Fabric) beziehungsweise auf eine Art den Betrieb sicherzustellen. Dies liegt an den verschiedenen Möglichkeiten, mit Fehlersituationen umzugehen. So kann man beispielsweise die Anwendung per API über die Änderung informieren, sofern sie dies unterstützt, oder einfach deren Konfiguration austauschen. Falls ein Proxy benutzt wird, kann dieser die Änderung abfragen.

Die am häufigsten verwendete Methode ist jedoch, den Wechsel mithilfe einer virtuellen IP zu vollziehen. Dabei ist die IP-Adresse, die die Anwendung zur Verbindung nutzt, nicht fest an einen Datenbank-Server gebunden, sondern wird je nach Bedarf und Topologie zusätzlich vergeben.

MHA und Orchestrator haben bei unterschiedlichen Ereignissen (wie Master up/down) Hooks integriert, über die man dann passende Skripte ausführen kann. Dies überlässt dem Administrator die volle Kontrolle über die zu treffenden Maßnahmen; so kann hier beispielsweise auch der ausgefallene Master nach der Wiederherstellung automatisch als Slave weiterlaufen. Dazu liefern beide Tools Beispiel-Skripte mit, die schnell an die eigenen Anforderungen angepasst sind.

Wie auch bei Fabric führt der Ausfall der Management-Instanz nur dazu, dass im Fehlerfall kein automatischer Failover stattfinden kann.

Während MHA einzig auf der Kommandozeile zu bedienen ist, bietet Orchestrator ein Web-Interface, über das man einen Überblick über die aktuelle Situation erhalten kann. Auch Änderungen der Topologie sind mit „Drag&Drop“ möglich, zudem lassen sich Server-Parameter einsehen und ändern. Zusätzlich sind diese Funktionen auch über ein Web-API zugänglich *(siehe Abbildung 3)*.

Als einziges vorgestelltes Tool kann Orchestrator von sich aus Hochverfügbarkeit herstellen. Dazu genügt es, mehrere Instanzen zu installieren und ein gemeinsames Backend zu nutzen.

Failover mittels Proxy

Eine weitere Möglichkeit, eine Anwendung an die Datenbanken anzubinden, besteht darin, einen Proxy zu verwenden.

Dabei wird in der Anwendung der Proxy als Ziel konfiguriert und dieser kümmert sich um die Weiterleitung der Verbindung. Hier bieten sich zum Beispiel „haproxy“ oder ProxySQL an. Allerdings ist bei einer Proxy-Lösung auch dafür zu sorgen, dass diese selbst hochverfügbar ist. Weitverbreitet ist die Nutzung von „keepalived“ für diesen Zweck, jedoch ist auch die Kombination aus „heartbeat“ und „pacemaker“ möglich. Entsprechende Anleitungen sind in der Dokumentation der jeweiligen Proxy-Software verfügbar. Mit Proxy lassen sich auch synchron replizierende Systeme anbinden.

Haproxy ist ein generischer Proxy, der für alle Dienste verwendet werden kann. In der einfachsten Variante wird lediglich überprüft, ob die Datenbank erreichbar ist. Dies ist allerdings in Replikations-Szenarien nicht ausreichend, da auch überprüft werden muss, ob der Server richtig repliziert und dessen Daten genügend aktuell sind. Realisieren kann man das mit eigenen Skripten, die von „haproxy“ via „xinetd“ gestartet werden.

Die bessere Wahl ist ProxySQL. Wichtig ist dabei, dass ProxySQL nicht allein arbeiten kann. Standardmäßig wird überprüft, welche Datenbank in der Topologie schreibbar ist. Auf diese werden dann die Schreib-Operationen geleitet, die anderen werden zum Lesen genutzt. Dies funktioniert sehr gut, wenn GRE benutzt wird. Standardmäßig wird nur ein Node des Clusters zum Schreiben freigegeben und die Cluster-Software kümmert sich selbst um die Topologie. Bei einer asynchronen Replikation gibt es eine solche Funktionalität jedoch nicht. Hier helfen MHA oder Orchestrator, die nach einem Failover die entsprechenden Einstellungen vornehmen. Zusätzlich bietet ProxySQL Query Routing und Firewall. So können Anfragen auf den für sie besten Server umgeleitet oder problematische Anfragen unterdrückt werden.

Fazit

Welches die richtige Replikationsmethode ist, hängt von der jeweiligen Anwendung ab. Grundsätzlich ist aufgrund der Robustheit eine Cluster-Lösung zu empfehlen. Hier empfiehlt sich die Group Replication, da sie sich besser in das bestehende System integriert. Auch dass hier Hosts automatisch auf „read_only“ gesetzt werden, hilft bei der Anbindung von Anwendungen mittels ProxySQL.

Wer über mehrere Rechenzentren ein Cluster bilden muss, sollte momentan noch zu Galera greifen. Hier überwiegen die Vorteile der von Codership implementierten WAN-Optimierungen deutlich. Man kann aber hier auch überlegen, ob man in den Rechenzentren GRE nutzt und diese untereinander durch asynchrone Replikation anbindet. Die asynchrone Replikation bietet sich an, wenn man wenig Hardware zur Verfügung hat oder große Datensätze schreiben muss.

Welches Tool zum Anbinden der Anwendung genutzt wird, richtet sich hauptsächlich nach der Replikationsmethode. Bei synchroner Replikation empfiehlt sich ProxySQL. Bei asynchroner Replikation bevorzugt der Autor aufgrund der besseren Übersicht im Web-Interface Orchestrator.

Matthias Klein
matthias.klein@innogames.com