Hochverfügbarkeit mit MySQL

MySQL zählt zu den verbreitetsten relationalen Datenbanksystemen. Leider hält sich trotzdem bis heute das Vorurteil, es sei eine nicht transaktionssichere, Spiel- und Basteldatenbank. Das mag daher rühren, dass mancher MySQL nur von Webservern her kennt, wo es eine Handvoll Datensätze verwaltet. Weitaus weniger bekannt ist, dass sich auch MySQL-Datenbanken hochverfügar auslegen lassen und dabei sogar mit Commodity-Hardware gut skalieren. Lösungsvorschläge für ein solches Setup diskutiert dieser Beitrag. Michael Spindler

Für den geplanten MySQL-Cluster ist zunächst grundsätzlich festzulegen, ob der Datenbestand über ein verteiltes Filesystem allen Knoten zugänglich sein soll, und falls ja, ob ihn die beteiligten Rechner dann gleichzeitig oder wechselweise nutzen. Diese Entscheidung fällt nicht schwer, denn im Gegensatz zu Oracle mit seiner RAC-Lösung gibt es derzeit keine sinnvolle Verwendung einer Shared-Disk-Cluster-Lösung mit MySQL in Active/Active-Konfiguration, bei der also mehrere Server parallel mit den Daten arbeiten. Kommen dennoch Shared Disks oder etwa DRBD gemeinsam mit MySQL zum Einsatz, so handelt es sich in der Regel um einen Active/Passive-Cluster mit zwei Knoten. Hier greift immer nur eines der Systeme exklusiv auf den Datenbestand zu.

Versagt in diesem Szenario das aktive System oder entscheidet ein externer Cluster-Manager, dass ein Failover nötig ist, so übernimmt der passive Knoten den Service und hat dann seinerseits exklusiven Zugriff auf das Dateisystem. Will der Admin hingegen die zentrale Datenhaltung vermeiden (Shared-Nothing-Architektur), dann bieten sich Cluster-Lösungen auf Basis der MySQL-eigenen Replikation oder der MySQL-Cluster an. Während die Replikation bereits ein ausgereiftes System zur Verteilung von Daten auf mehrere Server ist, handelt es sich beim Cluster von MySQL um ein relativ neues Produkt (die verwendete Storage Engine ist jedoch älter und stammt ursprünglich von der Firma Alzato). Beide Lösungen ermöglichen es außerdem, durch eine größere Anzahl von Rechnern die Performance zu verbessern (horizontale Skalierung).

Dieser Beitrag beleuchtet drei verschiedene Ansätze für solche Hochverfügbarkeits-Cluster mit MySQL: Den MySQL-eigenen Cluster, die MySQL-Replikation und eine Multi-Master-Architektur mit zwei MySQL-Servern und DRBD-gespiegeltem Datenbereich.

Der MySQL-Cluster

Der MySQL-Cluster besteht aus einer neuen Storage Engine NDB (Network Data Base), sowie einigen Daemons, die den Betrieb und die Datenspeicherung des Cluster kontrollieren. Ist im Folgenden von einem Node die Rede, so handelt es sich in der Regel um einen Prozess und nicht zwingend um einen eigenen physikalischen Rechner. Teil des Systems sind folgende Komponenten:

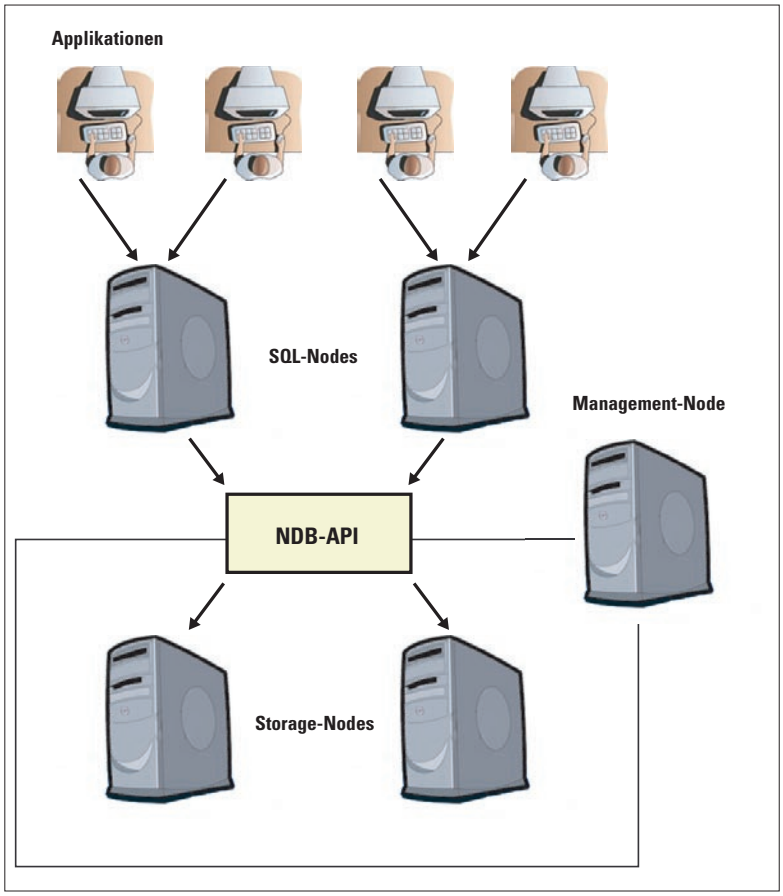

Abbildung 1: Das Architekturschema des MySQL-Cluster: SQL-Nodes vermitteln zwischen Anwendern und NDB-Engine, die ihrerseits die Daten auf verschiedene Storage Nodes verteilt.

- **SQL-Nodes:** Das Frontend eines Cluster bilden die SQL-Nodes. Dabei handelt es sich lediglich um einen MySQL-Daemon, der mit der Storage Engine NDB kommuniziert. Zu den SQL-Nodes verbinden sich die Applikationen.
- **Storage Nodes:** Die Storage Nodes bilden die Basis des Cluster. Wie der Name schon andeutet, halten sie die Daten vor. Jeder Node hält einen Teil der Gesamtdaten und entscheidet beim Eintreffen einer Abfrage, ob er diese allein beantworten kann oder noch Daten von anderen Storage Nodes braucht.
- **Management-Node:** Über den Management-Node ist der Administrator in der Lage, Kommandos an jede Komponente im Cluster zu senden und deren Status abzufragen. Zudem arbeitet der Node bei Kommunikationsproblemen als Schlichter zwischen den Cluster-Komponenten. Der Management-Node übernimmt in Split-Brain-Situationen die

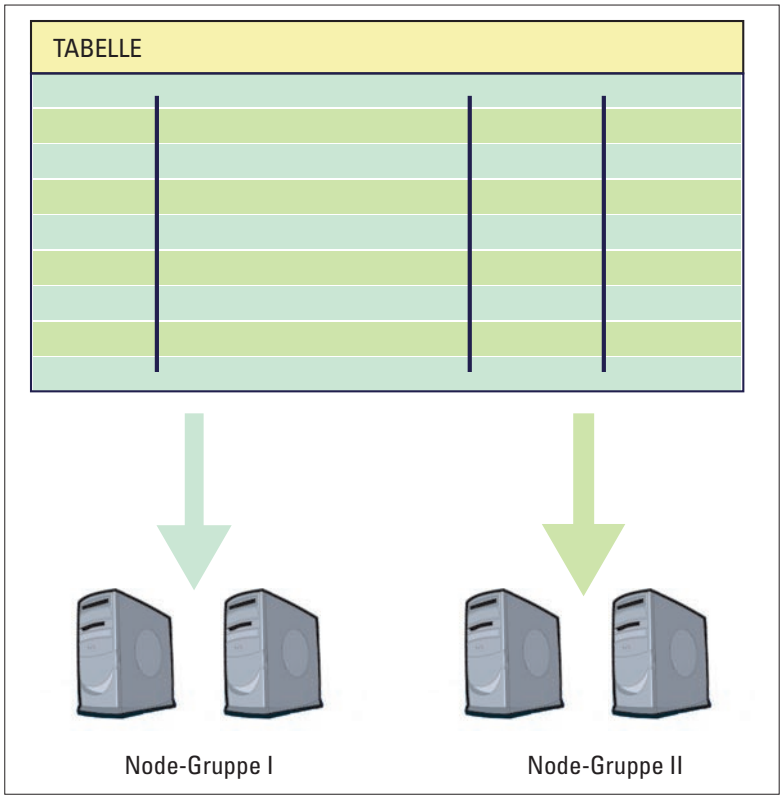

Abbildung 2: Die Tabelle wird anhand ihres Primary Key in einzelne Fragmente aufgesplittet und auf verschiedene Gruppen von Storage Nodes verteilt. Dort sind die Daten redundant gespeichert.

Entscheidung, welcher Cluster-Teil weiterarbeiten soll und welcher nicht. Normalerweise gibt es in einem Cluster-Setup nur einen Management-Node, es ist aber durchaus möglich, mehrere zu konfigurieren, um auch hier eine höhere Verfügbarkeit zu erreichen.

Ein hochverfügbares Storage Backend im Cluster

Wie erreicht nun der MySQL-Cluster Hochverfügbarkeit? Das zentrale Konzept des MySQL-Cluster ist die Aufteilung der vorhandenen Daten auf Storage Nodes (**Abbildung 1**). Die Partitionierung der Daten übernimmt eine Funktion, die auf dem Primary Key einer Tabelle arbeitet – ist keiner vorhanden, fügt sie einen versteckten Primärschlüssel ein. Die Funktion zerlegt die Tabellen in Fragmente. Jedes Tabellenfragment landet dabei im Cluster mehrfach auf verschiedenen Nodes (Replicas). Diese Nodes sind wiederum auf Node-Gruppen aufgeteilt, wobei alle Nodes innerhalb einer Gruppe dieselben Tabellenfragmente erhalten. Die Nodes innerhalb einer Node-Gruppe sind auf verschiedene physische Rechner verteilt (**Abbildung 2**). Die Größe einer Node-Gruppe bestimmt die Anzahl der Replicas – so befinden sich bei zwei Replicas zwei Nodes in jeder Node-Gruppe. Mit höherer Anzahl an Replicas erhöht sich natürlich auch die Verfügbarkeit. Mehr als vier sind jedoch selten sinnvoll.

Fällt ein Node aus, so bleibt jedes Datenfragment innerhalb einer Node-Gruppe dennoch mindestens einmal erhalten und der Cluster mit dem kompletten Datenbestand online. Ein Failover zwischen den Nodes erfolgt in weniger als einer Sekunde, so dass die zugreifende Applikation in der Regel nichts von einem Ausfall merkt. Meldet sich der wiederhergestellte Node dann im Cluster zurück, so erhält er die inzwischen geänderten Fragmente von dem durchlaufenden Knoten. Das geschieht transparent und automatisch.

Sichern der SQL-Frontends

Auch die SQL-Nodes müssen Fehler abfangen können. Dafür liefert MySQL keinen Mechanismus mit, und der Administrator muss hier selbst eingreifen. Um die SQL-Nodes hochverfügbar zu halten, stehen verschiedene Möglichkeiten zur Verfügung:

- **Round Robin DNS:** Der Admin weist hier einem DNS-Namen einfach alle IP-Adressen der SQL-Nodes zu. Zusätzlich konfiguriert er die Time To Live (TTL) so klein, dass die zugreifenden Applikationen immer wieder einen neuen Server für die nächste Verbindung zugewiesen bekommen. Sollte ein SQL-Node ausfallen, ist jedoch ein zusätzlicher Eingriff in das DNS-System nötig, da der DNS-Server natürlich weiterhin die IP-Adresse des ausgefallenen Systems weitergibt. Für einfache Setups oder Applikationen, die nach einem gescheiterten Connect fortlaufend weitere Versuche unternehmen, kann diese Methode unter Umständen trotzdem eine einfache und kostengünstige Alternative darstellen.
- **Linux-HA und Round Robin DNS:** Das gerade beschriebene System lässt sich entscheidend verbessern, wenn man zusätzlich auf jedem Node die Software Heartbeat 2 (1) aus dem Linux-HA-Projekt verwendet. Mittels Heartbeat 2 lassen sich Cluster mit mehr als zwei Knoten aufbauen. Jeder SQL-Node bekommt eine virtuelle IP-Adresse, die der Cluster-Manager »heartbeat« verwaltet. Diese virtuellen IP-Adressen hinterlegt man

im DNS entsprechend dem obigen Schema. Sollte ein SQL-Node einmal ausfallen, so migriert Heartbeat 2 seine IP-Adresse auf einen der übrig gebliebenen Nodes. Hierdurch bleiben alle im DNS zugewiesenen Server erreichbar. Den Ausfall eines SQL-Node können die in Heartbeat 2 eingebauten Systemchecks oder eigene Skripte feststellen.

- **Hardware Load Balancer:** Deutlich professioneller und sicherer, aber auch erheblich kostenintensiver, sind Hardware Load Balancer, die eine IP nach außen präsentieren und dann eingehende Verbindungen auf die verfügbaren SQL-Nodes verteilen. Der Load Balancer kann den TCP-Port des MySQL-Servers abfragen und so die Erreichbarkeit jedes Datenbankservers berücksichtigen. Zudem beherrschen diese Systeme sehr viel ausgefeiltere Verteilungsmechanismen als einfaches Round Robin. Um einen Single Point of Failure zu vermeiden, kommt man jedoch auch hier um eine Cluster-fähige Lösung nicht herum, was Kosten und Komplexität weiter erhöht.
- **Linux Virtual Server:** Das Linux-Virtual-Server-Projekt (2) stellt eine Softwarelösung als Alternative zu Hardware Load Balancern zur Verfügung. Auch hier stellen mehrere Server eine virtuelle IP-Adresse für die Applikationen zur Verfügung und können sie dann an die eigentlichen SQL-Nodes im Hintergrund verteilen. Hochverfügbarkeit für die Load Balancer stellt wieder der der Cluster-Manager »heartbeat« her.
- **JDBC-Clustering:** Als letzte Alternative sei noch das in JDBC integrierte Clustering erwähnt. Der Treiber bietet von vornherein die Möglichkeit, mehrere MySQL-Server zu konfigurieren, unter denen eine Lastverteilung mittels Round Robin möglich ist. Außerdem wählt der Treiber dann automatisch einen anderen Server, sollte der aktuelle nicht mehr verfügbar sein (3). Kommen ausschließlich Applikationen zum Einsatz, die eine solche Schnittstelle zur Datenbank verwenden, dann erspart man sich so die sonst nötige Erweiterung der SQL-Nodes.

Platzprobleme

Der MySQL-Cluster bis Version 5.0 war eine reine In-Memory-Datenbank und damit meistens von vornherein aus dem Rennen, wenn eine Entscheidung zwischen hochverfügbaren Datenbanksystemen zu treffen war. Sämtliche Daten und Indizes mussten im RAM der Cluster-Knoten Platz finden. Dabei bestimmte die Anzahl der Replicas (Minimum: zwei) über die nötige Menge an RAM.

In der Regel konfigurierte man nicht mehr als vier Replicas. Mehr beschleunigten zwar den Lesezugriff, die Schreibgeschwindigkeit litt jedoch, denn jeder Schreibzugriff war erst dann abgeschlossen (comitted), wenn alle Replicas die Daten erhalten hatten. Zwar erfolgen diese Zugriffe parallel, mit zunehmender Anzahl der Transaktionen steigt jedoch auch hier die Latenz für die Schreibzugriffe. Ist die Datengröße bekannt, kann der Admin anhand der Anzahl der Replicas folgende Näherung für den Gesamtbedarf an Hauptspeicher verwenden: Datengröße * Anzahl Replicas * 1,1. Der Faktor 1,1 ist dabei nur ein Richtwert, der eher nach oben zu korrigieren ist. Den so ermittelten Gesamtbedarf teilt man nun durch die Größe des Hauptspeichers, der pro System zur Verfügung steht, und erhält so die nötige Anzahl der Systeme. MySQL liefert zudem das Skript »ndb_size.pl« für eine Abschätzung der Speicherparameter. Das Skript ist auch getrennt über MySQL-Forge zu beziehen (4).

Version 5.1 verbessert die Speicherplatz-Situation entscheidend, denn nun ist es auch möglich, Daten auf der Festplatte abzulegen. Allerdings gilt dies nur für nicht indizierte oder mit Schlüsseln belegte Spalten. Es muss also immer noch genügend Hauptspeicher für sämtliche Schlüsselspalten vorhanden sein, das Gros der Daten kann jedoch kostengünstig auf Festplatten liegen.

Performance

Neben dem reinen Platzbedarf hat der Cluster leider meist auch noch einen Performancenachteil im Vergleich zu einem einzelnen MySQL-Server. Zwar scheint es im ersten Moment so, als müsse eine auf viele Systeme verteilte Datenbank bessere Antwortzeiten liefern, aber das ist leider nur bedingt richtig.

Jede Anfrage an die Datenbank erhält nämlich zunächst einen SQL-Node, den sie dann an die Data Nodes weitergibt. Die Antwort muss dementsprechend auch wieder den gesamten Weg zurück. Bei einem einzelnen MySQL-Server stellt fast immer der Disk-I/O den Flaschenhals der Performance dar. Im MySQL-Cluster hingegen spielt vor allem das Netzwerk eine aus-

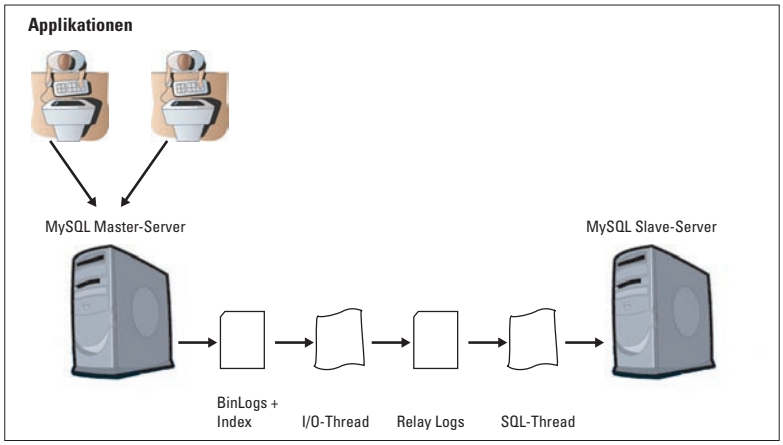

Abbildung 3: Asynchrone Replikation gibt es in MySQL schon sehr lange. Auch auf diesem Wege lässt sich die Verfügbarkeit der Datenbank deutlich verbessern.

schlaggebende Rolle. Mindestens 100 MBit sind Pflicht, Gigabit-Ethernet ist wünschenswert, ein SCI-Interconnect (Scalable Coherent Interface) wäre die Kür.

Ab Version 5.1 kommt für alle Data Nodes auch das Disk-I/O-Problem wieder, da es jetzt auch möglich ist, nicht indizierte oder mit Keys belegte Spalten auf der Platte abzulegen. Wenn allerdings ein einzelner Server nicht mehr genug Performance bieten kann oder der Ausfall eines Servers im Cluster zu keinem minutenlangen Stillstand führen darf, dann ist der MySQL-Cluster das Mittel der Wahl.

Noch mehr Einschränkungen

Leider verhält sich der MySQL-Cluster noch nicht ganz so wie ein einzelner MySQL-Server. Es bestehen eine ganze Reihe wichtiger Einschränkungen: Er unterstützt keine temporären Tabellen, keine FULLTEXT-Indizes und keine Foreign Keys. Alle Nodes innerhalb des Cluster müssen die gleiche Maschinenarchitektur haben. Pro Tabelle kann der MySQL-Cluster nur eine einzige AUTO_INCREMENT-Spalte verwenden. Aufgrund der Partitionierung der Daten erhalten Tabellen ohne Primary Key grundsätzlich eine versteckte Spalte als Primary Key, die dann das AUTO_INCREMENT-Flag besitzt. Daher ist ausschließlich die Primary-Key-Spalte als AUTO_INCREMENT verwendbar. Als Isolationslevel steht lediglich READ COMMITTED zur Verfügung Diese Auswahl ist natürlich noch nicht vollständig, eine komplette Liste der Limitierungen ist im Referenzhandbuch unter (5) zu finden.

Replikation

Die asynchrone Datenübertragung zwischen Servern ist bereits seit Jahren Bestandteil von MySQL (seit Version 3.23.15). Es gibt viele Varianten, um die Verfügbarkeit und im besten Fall zusätzlich die Leseperformance durch die MySQL-Replikation zu verbessern. Dieser Beitrag stellt beispielhaft einen Verbund aus zwei MySQL-Servern in einer Multi-Master-Architektur vor.

Grundsätzlich arbeitet bei jeder MySQL-Replikation ein Server als Master und ein oder mehrere als Slave(s). Der Master-Server schreibt sämtliche Statements, die Daten manipulieren, in seine Binärlogs und führt zudem einen Index über diese Logs. Der Slave meldet dem Master seine letzte Leseposition. Ein I/O-Thread liest anschließend von dort alle noch nicht übertragenen Daten, und ein weiterer SQL-Thread führt die Statements auf dem Slave aus (**Abbildung 3**). Hieraus ergibt sich die eigentliche Asynchronität der Verbindung. Erst nachdem der Master sein Binärlog geschrieben und der Slave es anschließend gelesen und dann ausgeführt hat, sind beide auf demselben Stand.

Jeder Slave kann wiederum selbst als Master fungieren und weitere Slaves versorgen. Im Fall einer Zwei-Node-Multi-Master-Architektur ist der erste Master-Server wiederum der Slave-Server seines Slave.

Das Hauptproblem bis Version 5 von MySQL waren die Auto-Increment-Felder. Kommt es durch einen Fehler zu einer Unterbrechung der Replikation während auf beide Systeme geschrieben wird, so führt beim Wiederanlauf der Replikation ein Insert auf die gleiche Tabelle mit einem Auto-Increment-Feld zu einem Duplicate-Key-Fehler.

Diese Situation lässt sich nun mit den Variablen »auto_increment_increment« und »auto_increment_offset« verhindern. Die Variable »auto_increment_increment« gibt an, um wie viel der nächste Schlüssel zu erhöhen ist, wogegen »auto_increment_offset« den initialen Versatz des Servers festlegt. Setzt man nun »auto_increment_offset« bei n Servern auf eine Zahl zwischen 1 und n, und »auto_increment_increment« auf n, so sind Duplicate-Key-Fehler aufgrund der Auto-Increment-Problematik ausgeschlossen.

Trotz dieser Verbesserung erhöht eine echte Multi-Master-Architektur und der gleichzeitige Zugriff auf beide MySQL-Server die Komplexi-

tät derart, dass eine einfachere Active/Passive-Lösung auf Basis der Multi-Master-Technologie meist die bessere Entscheidung ist.
Im Beispiel dieses Artikels soll ein Cluster-Manager eine virtuelle IP-Adresse immer dem gerade aktiven Server zuteilen, der über diese Adresse von außen ansprechbar ist. Der zweite Server fungiert als Slave. Gibt es ein Problem mit dem Master-Server, so migriert der Cluster-Manager die virtuelle IP-Adresse auf den zweiten Knoten, der daraufhin der neue Master ist, den die weiteren Anfragen erreichen. Kommt der ausgefallene Server anschließend wieder in den Cluster zurück, so holt er als Slave des neuen Master die in der Zwischenzeit geänderten Binärlogs und bringt sich automatisch auf den aktuellen Stand. Der Teufel steckt bei dieser Variante aber im Detail, und es ist entscheidend, eine detaillierte Überwachung der Fehlerlogs und der Replikation einzurichten. Kommt es zu einem Problem bei der Replikation (beispielsweise weil der Slave nicht Schritt halten kann und immer weiter hinter den Datenbestand des Master zurückfällt), so verliert man im Failover-Fall unter Umständen erheblich Daten und erhält eine inkonsistente Datenbank.

Überwachung ist wichtig

Entsprechende Skripte zur Überwachung finden sich unter (6). Die Failover-Zeit dieser Lösung beschränkt sich auf die Timeouts der Überwachungssysteme. Diese Zeitspanne ist nötig, um Probleme auf dem zweiten Knoten festzustellen und die Migration der IP-Adresse zu vollziehen. Kommt es zu einer Split-Brain-Situation, so hilft die behobene Auto-Increment-Problematik, auch einen solchen Fehlerfall zu überstehen.
Mit einer solchen Hochverfügbarkeitslösung ist die Replikation noch nicht ausgeschöpft. Gibt es zudem die Anforderung, die Performance lesender Zugriffe weiter zu verbessern, so lassen sich weitere Server als Slaves an den Master-Server binden. Die Applikation muss nun in der Lage sein, die Lese- und Schreibzugriffe auf verschiedene Server aufzuteilen, alle schreibenden Zugriffe aber ausschließlich an den Master-Server zu richten. Problematisch ist in einem solchen Fall das Failover, da keine Multi-Source-Replikation möglich ist, das heißt, ein Slave kann nicht von mehreren Master-Servern Daten beziehen. Zudem ist zu bedenken, dass so zwar die Lesegeschwindigkeit steigt, der einzelne Master-Server aber immer noch alle Schreibzugriffe übernehmen muss. Wichtig ist, sich zu vergegenwärtigen, dass die MySQL-Replikation nicht synchron arbeitet. Es ist nie sichergestellt, dass jede Transaktion tatsächlich bereits auf einem Slave gelandet ist.
Für die beiden schreibbaren Master-Server braucht man zudem identische und leistungsstarke Hardware, obwohl immer nur einer der beiden Server von Applikationen verwendet wird. Die Situation verschärft sich bei Einsatz von weiteren Servern entsprechend, wobei diese dann als Nur-Lese-Systeme dienen können, falls die zugreifende Applikation dies zulässt.

MySQL und DRBD

Das letzte Konzept, das dieser Artikel vorstellt, basiert auf dem Prinzip, dass sich zwei Server denselben Datenbereich teilen, allerdings in einer reinen Active/Passive-Konfiguration. Dafür kann man Shared Storage einsetzen, was aber hier außen vor bleiben soll. Stattdessen benutzt das folgende Beispiel eine Replikation auf der Grundlage von DRDB.
DRBD steht für Distributed Replicated Block Device und bietet die Möglichkeit, ein Blockdevice über ein IP-Netzwerk von einem Server auf einen anderen zu spiegeln. Im Gegensatz zur Replikation arbeitet DRBD blockbasiert, nicht zeilenbasiert.
Kommt es zu Problemen auf einem System im Cluster, so wartet der zweite Node einen Timeout ab, und danach erklärt er den primären Knoten für tot. In diesem Fall geht der sekundäre Server in den Standalone-Modus, wobei er periodisch versucht, den Partner im Cluster zu kontaktieren.
DRBD bringt ein Kernel-Modul und mehrere User-Space-Tools mit. Die maximale Größe der zu spiegelnden Daten liegt bei 4 TByte, die eigentliche Synchronisations-Geschwindigkeit hängt in der Regel von den lokalen Platten und der Netzwerkgeschwindigkeit ab. Kommt es zu einem Failover innerhalb eines DRBD-Verbundes, so ist es beim Wiederanlaufen des ausgefallenen Systems nicht nötig, erst den gesamten Datenbereich vollständig zu synchronisieren. Über eine Bitmap merkt sich das System die noch nicht replizierten Blöcke und überträgt nur diese, wodurch ein sehr schneller Resync möglich ist.
Für die Synchronisation zwischen den DRBD-Nodes stehen verschiedene Protokolle zur Verfügung: ▶

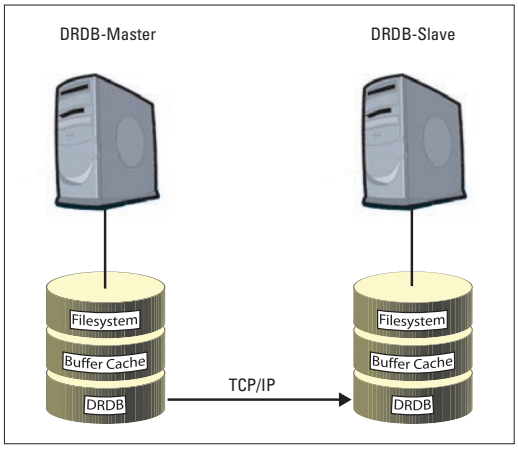

Abbildung 4: DRDB repliziert den Inhalt zweier Blockdevices transaktionssicher auf der Ebene der Datenblöcke und schafft so eine kostengünstige Alternative zu Shared Storage.

Protokoll A ist garantiert keine synchrone Datenreplikation und eignet sich lediglich für High-Latency-Netzwerke. Eine Schreiboperation auf dem Master gilt als abgeschlossen, wenn der Datenblock auf der lokalen Festplatte und im TCP-Send-Buffer gelandet ist.
Im Gegensatz zu Protokoll A gilt eine Schreiboperation in Protokoll B als abgeschlossen, wenn sie auf der lokalen Festplatte und im Remote Buffer Cache vollzogen ist. Auch hier ist keine echte Synchronisation garantiert.
Erst Protokoll C garantiert, dass eine Schreiboperation erst dann als abgeschlossen gilt, wenn sie sowohl die lokale als auch die entfernte Festplatte erreicht hat.
Mit Protokoll C lässt sich eine synchrone Datenübertragung zwischen zwei Servern sicherstellen. Das wiederum ist die Grundlage für einen MySQL-Verbund, bei dem MySQL nur auf dem Master-Server aktiv ist. Der Slave-Server bleibt passiv und agiert als DRBD-Slave, wobei DRBD für die Replikation sämtlicher Datenbankdateien sorgt. Kommt es dann zu einem Failover, so schwenkt der Cluster-Manager den Mount des Datenverzeichnisses auf den Slave-Knoten und startet danach auf dessen Seite MySQL. Zusätzlich sollte man eine virtuelle IP-Adresse für beide Knoten verwenden, damit der Failover transparent für die zugreifenden Anwendungen bleibt.
Es ist hierbei sinnvoll, auf MyISAM-Tabellen zu verzichten und lieber schwerpunktmäßig den Typ InnoDB einzusetzen. Datenbanktabellen mit InnoDB lassen sich nämlich schnell, automatisch und sicher wiederherstellen, wogegen es bei MyISAM-Tabellen unter Umständen zu Datenkorruption kommt und die Tabellen erst noch zu reparieren wären.
MySQL AB bietet inzwischen auch MySQL zusammen mit DRBD als eigenen Service für Hochverfügbarkeitslösungen an. Neben Consulting kann der Anwender zusammen mit MySQL Enterprise nun auch Support bei DRBD-Problemen erwerben (7).

Fazit

Kommt es nicht auf jede Transaktion an, kann schon eine einfache Replikation reichen, um wichtige Daten ständig auf zwei oder mehr Systemen verfügbar zu halten. Mit DRBD und MySQL beziehungsweise einem Shared Storage kommt die klassische Methode für einen Zwei-Node-Cluster hinzu. Gegebenenfalls muss man hier jedoch längere Schaltzeiten aufgrund des Recovery-Vorgangs in Kauf nehmen, und schließlich bleibt der MySQL-Cluster die Lösung, die am oberen Ende der Fahnenstange angesiedelt ist.
Das System hat ehrgeizige Ansätze, wobei leider die noch vorhandenen Beschränkungen einen Einsatz des Cluster oft verhindern und man in diesen Fällen eine einfachere Methode mithilfe eines Cluster-Managers und der beschriebenen Verfahren der Zeilen- oder Blockreplikation implementieren muss.
MySQL kann es durchaus mit den großen Datenbank-Management-Systemen aufnehmen, wenn es um Hochverfügbarkeit geht. Für nahezu jeden Anwendungszweck findet sich eine passende Lösung. MySQL AB unterstützt alle hier vorgestellten Varianten durch Support und Consulting. *(jcb)*

Infos

(1) Heartbeat 2: (http://www.linux-ha.org)
(2) LVS: (http://www.linuxvirtualserver.org)
(3) JDBC-Clustering: (http://www.mysql.de/products/connector/)
(4) Skript für die Speicherberechnung: (http://forge.mysql.com/projects/view.php?id=88)
(5) Einschränkungen des MySQL-Cluster: (http://dev.mysql.com/doc/#refman)
(6) Skripte zur Replikationsüberwachung: (http://forge.mysql.com/)
(7) MySQL Enterprise und DRDB: (http://www.mysql.de/products/enterprise/drbd.html)