Clusteren met MySQL (3)

Zeer beschikbaar: MySQL op eigen houtje

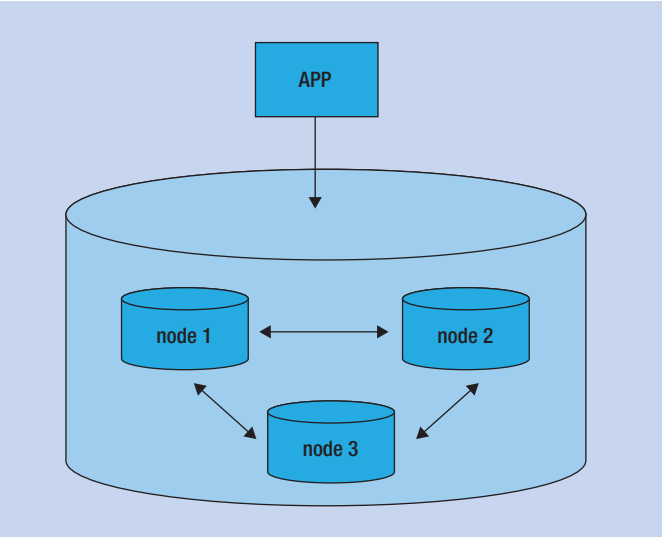
Rick van Rein

In de vorige afleveringen van dit drieluik bespraken we DRBD, en MySQL op DRBD, als methoden om data beschikbaar te houden terwijl een halve serverruimte instort. In deze laatste aflevering kijken we naar technieken die MySQL zelf beschikbaar houden gedurende zo'n rampenscenario.

Bij een hoge mate van beschikbaarheid gaat het voornamelijk om het niet down gaan van een database terwijl de hardware aan alle kanten uit elkaar rammelt. Het algemene idee is dat je op geen enkel punt in je architectuur afhankelijk wilt zijn van een enkele machine.

Two-phase commit

Om afhankelijkheid van een enkelvoudig punt te voorkomen zal de software bestaan uit lagen die elk alle onderliggende lagen aanspreken. Een voorbeeld daarvan is de SQL-applicatie (bijvoorbeeld een set PHP-scripts) die meerdere onderliggende databases aanspreekt om daarin dezelfde wijzigingen aan te brengen. Dit is te realiseren met elke database die two-phase commit ondersteunt; dat wil zeggen een transactie die met PREPARE wordt klaargezet voor een COMMIT, maar nog net niet zo vergaand dat er geen ROLLBACK meer mogelijk is. Als alle onderliggende databases de PREPARE hebben geaccepteerd krijgen ze allemaal een COMMIT toegezonden, maar als er een weigering optreedt bij één database dan wordt het een ROLLBACK.

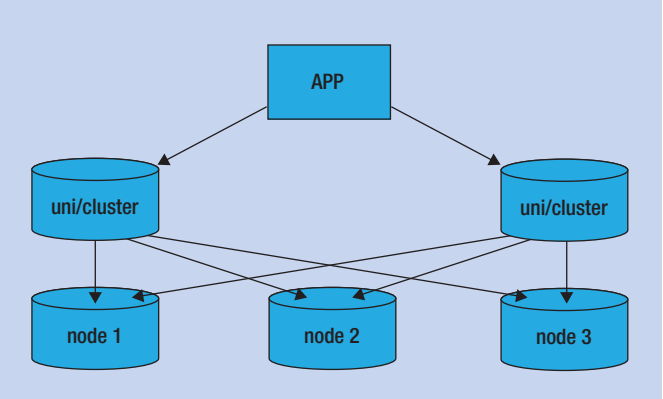


Afbeelding 1: Een MySQL cluster wordt vanuit de applicatie aangesproken alsof het een enkele node is, maar in werkelijkheid zijn het er meer.

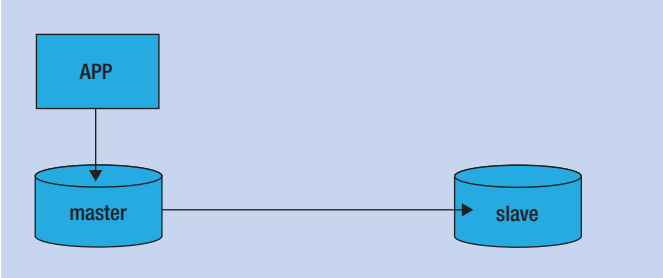
De reden waarom dit algemene principe niet zo aantrekkelijk is, is dat het de logica in de applicatie niet afdoende scheidt van de opslagmethode. Zo'n scheiding komt doorgaans ten goede aan de helderheid van de applicatiecode, en dus aan de kans dat die foutloos functioneert.
De samenhang tussen de componenten in een two-phase commit kan al snel complex worden en werkt dan dus foutbevorderend. Om die reden is deze opzet van redundantie dus niet van praktisch nut. Praktische oplossingen voor redundantie (en dus een hoge mate van beschikbaarheid van data) maken het daarom mogelijk de applicatie vrijwel onaangeroerd te laten en slechts de onderliggende databaselaag anders in te richten.

MySQL cluster

De nieuwste methode voor een zeer beschikbaar MySQL is MySQL cluster. Dat is ontworpen om geen enkelvoudig punt te hebben waarop het kan falen. De database bestaat uit een aantal 'nodes' die op afzonderlijke machines draaien en die met elkaar communiceren om te zorgen dat ze dezelfde antwoorden geven op SQL-query's. De applicatie spreekt één van de nodes aan en krijgt antwoord als het cluster de query heeft verwerkt.
Het onhandige aan MySQL cluster is, dat het draait op een speciale storage engine, ofwel een geheel eigen opslagmechanisme. Een volwassen applicatie die allerhande aannames doet op gebieden als transacties (bijvoorbeeld voor de toegepaste vormen van locking) loopt dus kans niet goed te werken op MySQL cluster. Er zijn erg veel beperkingen die aangeven dat MySQL cluster nog een beetje onvolwassen is voor het algemene geval. Wat natuurlijk niet wegneemt dat het een geschikte basis kan zijn om een nieuwe applicatie mee op te bouwen. Alleen voor oude applicaties moet expliciet zijn wat de aannames over de storage engine zijn, zodat die kunnen worden afgecheckt.
Een hele rare trek van MySQL cluster is echter dat het niets op de schijf vastlegt. Alles wordt in het geheugen bewaard, en als alle nodes tegelijk zouden crashen zouden dus alle data verloren gaan! Omdat volledige onafhankelijkheid tussen nodes inhoudt



Afbeelding 2: Met Continuent spreek je niet de database aan, maar een intermediair die met kennis van SQL slim omgaat met het verzenden van de query's naar de diverse nodes.



Afbeelding 3: De eenvoudigste manier om MySQL replicatie te gebruiken is een 'live backup' slave die een master volgt. De applicatie spreekt alleen de master aan.

dat zelfs het systeembeheer en de versies (en fabrikanten?) van de software onafhankelijk zouden moeten zijn, is dat dus geen echt solide basis voor een zeer beschikbaar systeem.
Een laatste punt waar ook de halve wasdom van MySQL cluster uit blijkt is dat het nog niet mogelijk is om dynamisch nodes aan een cluster toe te voegen en eruit te verwijderen. MySQL cluster is dus echt bedoeld voor een vast op te stellen kluitje machines die samen als een geheel naar buiten gaan treden. Het is dan ook niet gebouwd om een machine op afstand mee te laten draaien in het cluster, dat zou het geheel te zeer vertragen.

Continuent uni/cluster

Het commerciële product uni/cluster van Continuent is een uitbreiding op MySQL dat conceptueel eenvoudig, en tegelijk best krachtig is. Het gaat om een intermediair waarheen SQL-query's worden gestuurd. Deze intermediair is geen MySQL-node, maar stuurt de query's door naar de daadwerkelijke nodes. Door SELECT-query's naar een enkele node te sturen en UPDATE's naar alle nodes, wordt zelfs aan een verdeling van de werklast gedaan.
Een verwarrende beperking aan het uni/cluster systeem is echter wel dat het slechts twee van deze intermediairs ondersteunt. Het is dus niet mogelijk om uitgaande van een uptime per node en de gewenste uptime van het gehele databasesysteem te bepalen hoeveel nodes nodig zijn; een 'single point of failure' wordt een 'double point of failure' maar nooit een 'triple point of failure'. Als een node 99 procent online is, dan kan met uni/cluster nooit meer dan 99,99 procent uptime worden gehaald. Continuent heeft overigens websites op continuent.com en continuent.org.

MySQL replicatie

Naast al deze tamelijk geavanceerde mogelijkheden biedt MySQL nog een laatste, tamelijk eenvoudige mogelijkheid die in de praktijk erg krachtig blijkt, en dat is MySQL redundantie. Het idee van MySQL redundantie is dat een server die gewijzigd wordt een update naar een geregistreerde slave kan zenden, waarna die de update overneemt. De master en slave zijn in sync gebracht door een initiële dataset met een bijbehorend volgnummer van de master op de slave te installeren, samen met de contactgegevens van de master.

Dit volgnummer, tezamen met de identiteit van de master, wordt in de slave opgeslagen, en gebruikt om in samenhang met de master te bepalen of er updates zijn om te downloaden. Dit mechanisme kan er onder andere tegen als de verbinding tussen master en slave uitvalt, tot enkele dagen aan toe. In zo'n zee van tijd zijn hele machineparken te verhuizen, dus het is ruim voldoende om onderbrekingen te overbruggen. Zodra master en slave elkaar hervinden wordt bijgepraat totdat de slave weer kan melden dat hij "0 seconds behind master" is.

Architectureel elastiek

Het systeem is verder zodanig flexibel dat er meer architecturen mee kunnen worden opgezet dan een enkele master/slave-combinatie. Het is bijvoorbeeld mogelijk om een hele sliert van slaves achter een master te hangen, waarbij elke volgende slave de informatie van de voorliggende master overneemt.

> Bij het repliceren worden SQL-statements rondgepompt en niet data

Hiervoor moet een speciale optie log-slave-updates worden ingesteld in de configuratie-file, om ervoor te zorgen dat ontvangen updates worden doorgeseind. Het is echter niet nodig om alles achter elkaar te hangen; een master kan ook best meerdere slaves bedienen. Wat niet lijkt te kunnen, is alternatieve masters aan een enkele slave toekennen.

In een drieluik bespreekt Rick van Rein open source technieken rond clustering en high-availability en andere methoden om MySQL zeer beschikbaar te maken.

Persistentie in RAM

MySQL cluster gebruikt een storage engine die alleen in RAM opslaat. Dat is een beetje eng, maar is erop gebaseerd dat er bij een crash altijd wel een andere node zal zijn die de gegevens uit RAM kan terugroepen als het nodig is. Ook bij MonetDB zagen we in het verleden al een dergelijke keuze, waarbij de bewerkte data in het geheugen werden gehouden.
Vroeg of laat krijgen deze modellen vermoedelijk de overhand. Er zijn nieuwe technieken in de maak die zowel RAM als harde schijf opvolgen, door beide te vervangen door random access (dus zonder seek-tijden toegankelijk) geheugen dat persistent blijft (dus de inhoud vasthoudt als de spanning wegvalt). Flash is ongeschikt voor die situatie doordat de cellen kapot zijn na een paar miljoen herschrijvingen van de inhoud, maar er zijn opvolgers van flash in de maak die dergelijke problemen niet kennen, bijvoorbeeld een siliciumimplementatie van het aloude ringkerngeheugen! Voor details op dit punt verwijzen we naar Storage Magazine nummer 1 van 2008.

Bij dit alles speelt vroeg of laat de vraag op wat de persistentie is die moet worden verzekerd. Als de master meteen na een update uitvalt door een disk crash, is het dan een ramp om terug te vallen op een slave, en daarbij de laatste paar seconden aan werk te verliezen? Dat zijn gecommitteerde transacties geweest, dus hoe schadelijk is het als dat verloren gaat?

> Het is mogelijk om een hele sliert van slaves achter een master te hangen

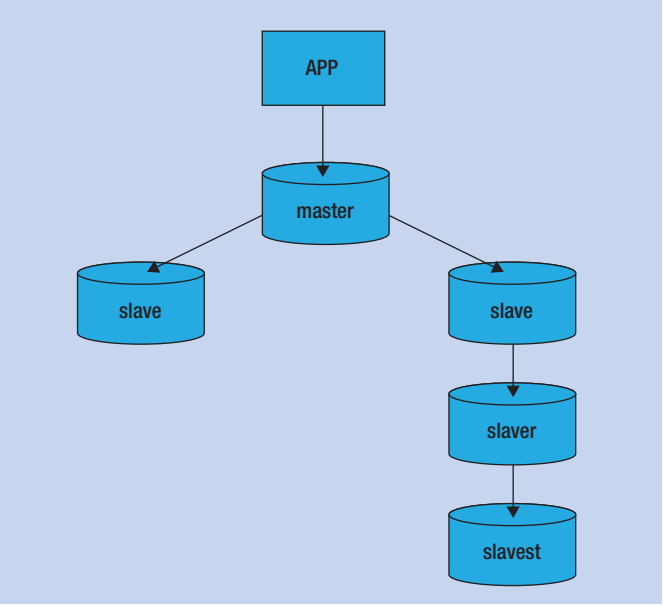
Het antwoord op deze vraag kan verschillen per onderdeel van een applicatie. Een beurstransactie mag vermoedelijk niet verloren raken door zo'n disk crash, maar de laatste inlog-tijd of een aanpassing van een voorkeursinstelling mogelijk wel. MySQL redundantie maakt het mogelijk verschillende graden van zorgvuldigheid te betrachten om hiermee om te gaan.

Standaard krijgt de applicatie van de master een akkoord op een COMMIT als de persistentie op de lokale node geregeld is. De COMMIT zegt dus niets over de toestand van de slaves, zelfs al is die normaal gesproken vrijwel direct up to date. Maar het is mogelijk om met SHOW MASTER DATA op te vragen welk sequentienummer de globale databasetoestand op de master heeft na zo'n commit, en vervolgens kan met MASTER_POS_WAIT worden gewacht tot die databasetoestand op een andere node is aangekomen. Het is mogelijk om daarbij een time-out op te geven, zodat kan worden ingesprongen op een slave die niet bijblijft met de master. Verder is het interessant op welke slave deze MASTER_POS_WAIT wordt uitgevoerd; is dat de laatste in een keten van nodes dan is de persistentie door de gehele database-architectuur geregeld, maar evengoed kan slechts één backup zekergesteld worden.
De toevoeging van de functie MASTER_POS_WAIT aan de standaard semantiek van lokale transacties maakt het dus mogelijk om zeer gevarieerd om te springen met de mate van persistentie van data, zelfs applicatie-afhankelijk. En in tegenstelling tot de eerder genoemde problemen door two-phase commit valt deze variatie in de aangesproken database node nogal mee als complicerende factor in de applicatiecode.

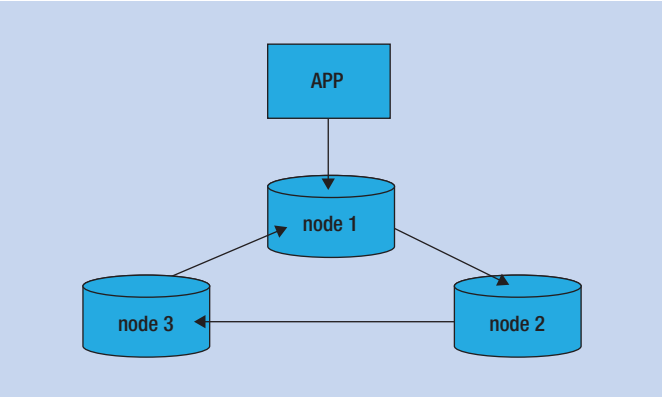
Toch is er ook nu weer sprake van een afweging. MySQL replicatie leent zich bijvoorbeeld uitstekend voor het vrijwel direct bijwerken van een server op grote (en veilige) afstand. Maar zodra gebruik gemaakt wordt van MASTER_POS_WAIT zal de applicatie wel moeten wachten totdat de update over die afstand (en via de niet-controleerbare paden van het Internet) is aangekomen op die veraf gelegen server. De gebruikelijke afweging tussen de snelheid van de updates en de mate waarin de inhoud veilig gesteld is blijft dus. Alleen is het wel prettig dat de traagheid van een veilige update alleen nodig is voor de updates die een centrale rol spelen in de applicatie.

Het kringetje rond

Als laatste bijzonderheid van MySQL replicatie is het nuttig op te merken dat het in een cycle kan worden opgesteld. In plaats van een vaste afvalvolgorde van de hoogste master naar de laagste slaves, is elke node in zo'n systeem gelijkwaardig.



Afbeelding 4: Flexibiliteit. De mogelijkheden van MySQL worden eigenlijk alleen beperkt doordat een node maar een enkele master kan hebben; er is geen beletsel om aan één master meerdere slaves te hangen, of om ze door te lussen.



Afbeelding 5: Een cyclische opstelling met MySQL replicatie maakt zelfs multi-master mode mogelijk, gegeven een paar randvoorwaarden.

Uiteraard moet hierbij ook met log-slave-updates worden aangegeven dat binnenkomende updates worden doorgezonden naar de volgende node in de cycle. Een probleem in zo'n opzet is uiteraard dat updates niet eindeloos rond moeten gaan sjezen, maar dat is in te stellen met de optie replicate-same-server-id=0 in de configuratie-file van MySQL. Deze optie geeft aan dat updates die binnenkomen maar die van de eigen node afkomstig zijn, niet nogmaals worden doorgestuurd. Ofwel, alle updates maken een rondje, tot ze terugkeren op de node die ze heeft uitgestuurd.
Bij het repliceren worden SQL-statements rondgepompt, en niet data. In veel toepassingen zal dat betekenen dat veel minder bandbreedte nodig is om een slave bijgewerkt te houden met een master dan door offline backups van data te transporteren. Daarnaast is de replicatie ook nog een stuk sneller, dus eigenlijk is het alleen al ter vervanging van (online) backups een uitstekend idee.

Multi-master mode

In cyclisch opgestelde MySQL nodes wordt het mogelijk om multi-master mode te gebruiken, ofwel dat niet alleen SELECT, maar ook UPDATE naar een willekeurige node kan worden verzonden. Of dat overigens veel oplevert is maar de vraag, want elke UPDATE wordt natuurlijk op elke node gedraaid.
Wanneer de applicatie updates verdeelt over nodes ontstaat bovendien de kans dat er timing- en locking-problemen ontstaan. Locks zijn lokaal en dus kan het gebeuren dat een andere node iets doet wat volgens de lokale lock niet zou mogen. Ofwel deze aanpassing van transactiesemantiek dient acceptabel te zijn voor de applicatie, ofwel de applicatie moet altijd dezelfde node als master aanspreken – tenminste, totdat die master crasht. Wanneer een applicatie daadwerkelijk multi-master draait, bijvoorbeeld omdat er meerdere locaties zijn voor zowel database als applicatie, dan moet rekenschap genomen worden van een aantal zaken. Ten eerste is het verstandig om CREATE TABLE en andere onderhoudscommando's altijd op dezelfde node uit te voeren zodat daar geen name clashes ontstaan. Ten tweede moet worden uitgekeken voor overlap van AUTO INCREMENT identifiers. Ook dit mechanisme werkt namelijk lokaal. Ook hiervoor zijn oplossingen – per node kun je een andere auto_increment_offset instellen en voor de groep nodes kun je een voldoende hoge auto_increment_increment instellen – maar die laten wel gaten achter in de identifier-reeks, en ook is het geen volledig geordende getallenreeks meer. Dat moet maar net passen in je applicatie.
Enfin, echt grote bezwaren treden niet op wanneer MySQL replicatie in multi-master mode wordt toegepast, maar het is wel iets waarmee bij de bouw van de applicatie rekening moet worden gehouden.

Conclusie

Zoals eerder opgemerkt is MySQL al lang niet meer de database die alleen door 'sproetenneuzen' op zolderkamertjes wordt toegepast. In dit drieluik zagen we eerder al de mogelijkheid om onder MySQL data te repliceren, namelijk met DRBD, en nu hebben we gezien dat er een cluster-systeem is dat wellicht binnenkort tot wasdom komt; en dat er replicatiemogelijkheden zijn op SQL-niveau die zelfs in multi-master mode gebruikt kunnen worden. Wat over de hele linie opvalt is de flexibiliteit van de mogelijkheden.

Rick van Rein
Dr. ir. H. van Rein (rick@openfortress.nl) is ontwikkelaar en beheerder bij OpenFortress Digital signatures.