Clusteren met MySQL (1)

Zeer beschikbaar: MySQL op DRBD

Rick van Rein

MySQL is al lang niet meer de 'speelgoed-database' die alleen onder de websites van sproetenneuzen draait. Zo biedt de database tegenwoordig goede aansluiting op High-Availability Linux.

Het doel van het High-Availability Linux project is om een Linux-systeem te bouwen dat zo betrouwbaar mogelijk online is, zonder meteen de investering van een mainframe noodzakelijk te maken. De invalshoek van de oplossing is om clusters te bouwen van relatief goedkope machines. In zo'n cluster kan een systeem uitvallen zonder dat het geheel meteen ten onder gaat.

Het grote probleem in een cluster is niet zozeer het repliceren van een functie, maar vooral de beschikbaarheid van data. Wanneer die voortdurend worden aangepast, zoals in een database, dan is het een probleem om ervoor te zorgen dat alle servers altijd de meest recente data aan boord hebben. Gewone backups lopen per definitie achter, dus er is een methode nodig om onmiddelijke backups te hebben. Die uitdaging wordt in deze serie aangepakt op basis van open source componenten. Op een cluster kan een veelvoud aan applicaties draaien, mits aan bepaalde voorwaarden wordt voldaan. Die voorwaarden zijn zo redelijk dat het geen probleem is om MySQL betrouwbaar op een HA-Linux cluster te draaien.

Componenten van HA-Linux

Het HA-Linux project heeft twee belangrijke componenten opgeleverd, die de essentie vormen van het idee van High-Availability Linux. Het eerste component is een gedistribueerd block device, het tweede is een heartbeat protocol dat detecteert als een ander systeem uit de lucht gaat.
Het gedistribueerde block device kan worden vergeleken met mirrorred RAID, waarbij dan één schijf over het netwerk wordt aangesproken. Een normale netwerkverbinding kan zich aardig meten met een IDE- of SCSI-interface qua snelheid, dus zo raar is dat niet eens. In vroege versies van HA-Linux werdt wel geëxperimenteerd met een constructie via de al lang aanwezige driver voor netblocks, maar dat bleek niet zo stabiel. Daarom is een nieuwe driver met als naam DRBD in het leven geroepen. DRBD staat voor Distributed Redundant Block Device.
Heartbeat is een protocol dat regelmatig de andere host of hosts polst om te zien of ze zich goed voelen; als enige tijd niet meer wordt gereageerd in het kader van het heartbeat-protocol dan wordt geconcludeerd dat de andere kant weggevallen is, of niet langer bereikbaar. Dit kan ertoe leiden dat een passieve slave tijdelijk verandert in de master van een cluster.

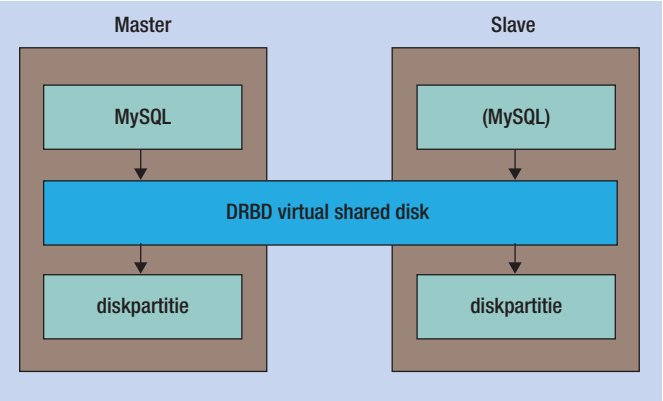
Vinger aan de pols

Het heartbeat protocol heeft als hoofdtaak om in de gaten te houden welke computer in een cluster de master is, en wat de eigen rol is van de computer waarop het draait. Zo zorgt elke computer in een cluster via heartbeat voor een overzicht over het cluster.
De eenvoudigste clusters bestaan uit twee computers. Daarbij functioneert één als de master, de andere als slave. Alle wijzigingen worden op de master gemaakt, en de slave volgt dat op. Zodra de slave merkt dat de master uit de lucht gaat, schakelt die om naar de master-toestand. Dat wordt gedaan door een aantal programma's te stoppen en te starten, zodat de draaiende programma's de computer tot master maken. Dit duurt een klein aantal seconden. Na die tijd is het cluster weer klaar om verder te gaan.

Heartbeat blijft controleren wat de toestand is van de andere computers in een cluster. Zodra een tijdelijke master merkt dat de oude master weer online is gekomen, kan de rolverdeling bijvoorbeeld weer worden teruggedraaid. Wederom gebeurt dat door een aantal programma's te stoppen, en zonodig andere te starten.
Heartbeat functioneert dus als een supervisor over de computer, die zich baseert op andere computers die waarneembaar zijn in het cluster. De configuratie van heartbeat op alle computers in een cluster is zodanig dat het cluster als één geheel naar buiten treedt.

Drieluik

In een drieluik bespreken we open source technieken rond clustering en high-availability. In de volgende delen bespreken we andere methoden om MySQL zeer beschikbaar te maken, en andere toepassingen van het DRBD/HA mechanisme uit dit deel.



Afbeelding 1: De architectuur van DRBD doorsnijdt het grensvlak tussen servers, door een virtuele harde schijf aan te bieden. De data worden gemirrorred opgeslagen, zoals dat ook bij RAID zou gebeuren, maar dan op gedistribueerde schijven.

De kracht van DRBD

Een block device is een algemene primitieve op een Unix-systeem; daarbovenop bouwt men doorgaans file systemen, en toepassingen zoals databases. Er zijn aardig wat trucs die block devices op een creatieve manier uitbuiten. DRBD is ook zo'n truc – het biedt de diensten van een block device aan, en heeft daarvoor een partitie nodig voor lokale opslag en een remote block device voor de replica. Op de andere kant draait ook DRBD, met een omgekeerde kijk op de configuratie.
De DRBD-componenten communiceren rechtstreeks met elkaar. Conceptueel biedt DRBD een gedeeld block device op twee computers aan. Bij veranderingen op de ene wordt de ander tegelijk mee aangepast. En net als op een gewone schijf verschillende programma's kunnen worden losgelaten, kunnen op deze gedistribueerde schijf ook verschillende programma's op verschillende computers worden losgelaten.
Wanneer programma's niet op dezelfde computer draaien kan er wel een complicatie optreden met locks – een lock dat in-memory werkt bijvoorbeeld, wordt niet goed gedeeld. Dat kan ook het geval zijn met file system locks. Daarom zijn er speciale cluster file systemen die ook locks gedistribueerd maken.
Zo'n file systeem kan dan bovenop DRBD worden gedraaid.

Het kan echter ook simpeler. Als we bijvoorbeeld op een andere manier zeker stellen dat nooit twee programma's tegelijk naar dezelfde resource zullen grijpen, dan voegt locking niets meer toe. Als bijvoorbeeld twee computers elk een MySQL server op dezelfde database loslaten, dan gebeuren er dingen waar MySQL niet op gebouwd is. Maar als ervoor wordt gezorgd dat er altijd maar één MySQL server actief is, dan wordt dat probleem voorkomen en gaat het goed. Dit is het geval wanneer de heartbeat-scripts ervoor zorgen dat alleen de master een MySQL server draait.
Enfin, hier verdwalen we al in de hogere laag van de toepassing van een DRBD-device; op zich doet dat geen afbreuk aan de mogelijkheden van DRBD om een virtuele schijf tussen twee computers te delen. Het valt te verwachten dat DRBD andere software in de verleiding brengt om zich ook aan te passen aan deze gedeelde resources. Op zich zijn cluster file systemen al zo'n volgende stap. Het lijkt redelijk te verwachten dat key/value databases zoals GDBM en BerkeleyDB ook een keer zullen volgen, gevolgd door applicaties zoals RDBMS'en die daar weer op bouwen.

Uitbreidingen aan DRBD

Oorspronkelijk is DRBD ontworpen voor twee systemen die pal naast elkaar stonden. Inmiddels wordt gewerkt aan veralgemenisering van deze opzet, met meer dan twee systemen en voor systemen op langere afstanden. De combinatie van deze twee veralgemeniseringen is echter nog in de maak. De veralgemenisering naar langere afstanden is verre van triviaal. Immers, zulke afstanden moeten via Internet en dus kan aankomst niet worden gegarandeerd, en kunnen vrij grote vertragingstijden optreden. Dat houdt in dat de tijd tot een schrijfactie succes terugrapporteert veel langer wordt dan de tijd voor een bewerking met een lokale harde schijf. Dit kan een flinke vertraging veroorzaken.
Dit wordt in DRBD opgelost door een aantal protocollen te ondersteunen. Afhankelijk van het protocol is het regime van synchronisatie tussen de DRBD-instanties strakker of losser te maken. Protocol C is strikt synchroon tussen de twee gekoppelde schijven, B is losser en A is zo los dat een systeem al doorloopt voordat de andere kant heeft erkend dat een update is aangekomen.

> De kracht van het systeem zit in de betrouwbare uptime

Protocol C werkt uitstekend als systemen naast elkaar staan, protocol A kan dienen om zo goed mogelijk een verafgelegen systeem bij te werken, zonder dat een schrijfactie op de master er te erg door wordt vertraagd. MySQL biedt overigens een master/slave redundantiemechanisme dat handiger is dan het gebruik van protocol A.

Ontwerp van een MySQL cluster

Hoe ziet een twee-machine cluster met MySQL op DRBD er uit? Wat maakt het mogelijk en wat niet? Allereerst kan er maar op één computer tegelijk een MySQL server draaien, dus van load balancing kan geen sprake zijn. De rekenkracht wordt dus geheel waargenomen door een enkele computer in het cluster, de master. Om in noodsituaties nog op vol tempo te werken zou de slave dezelfde rekenkracht moeten hebben, dus we spreken over twee krachtige server machines.
De kracht van het systeem zit in de betrouwbare uptime. Als ze volledig onafhankelijk falen is de faalkans van twee systemen

het kwadraat van dat van één systeem. Een systeem dat 99 procent uptime heeft, faalt met een kans van 1 procent – een tweetal machines faalt dan met een kans van 0,01^2 = 0,0001, dus 0,01 procent. De uptime van het cluster is dan dus 99,99 procent. Met een derde computer zou het een derde macht zijn en eindig je met 99,9999 procent uptime. Dit wordt alleen gehaald als alles, van voeding tot harde schijven, per systeem onafhankelijk faalt.
Maar terug naar het voorbeeld met twee computers. Op master en slave wordt een even grote partitie aangemaakt voor de opslag van data. Daarop wordt DRBD gebouwd, met een verwijzing naar de onderliggende partitie en naar de andere computer in het cluster. Redundantie is al zinvol bij systemen die vlakbij elkaar staan, dus kiezen we daarvoor, ook al moeten de computers dan dicht bij elkaar staan. Omdat DRBD ongecodeerde datablokken uitwisselt, is het een goed idee om een directe link te leggen tussen de clustercomputers, die toegewijd is aan DRBD en eventueel de heartbeats. Dit voorkomt bijvoorbeeld problemen door een falende switch.

> De configuratie van heartbeat is zodanig dat het cluster als één geheel naar buiten treedt

Vervolgens zijn er scripts nodig op – met name – de slave. De slave draait normaal gesproken niets bovenop de DRBD-partitie, niet eens een file systeem. Pas wanneer blijkt dat de master problemen heeft, begint de slave iets te doen met deze partitie. Allereerst wordt het file systeem gereconstrueerd zoals dat gebruikelijk is na een plotselinge onderbreking van de voedingsspanning, want daarmee laat een afgekapte reeks updates zich goed vergelijken. Dit gaat natuurlijk het snelst met een journaling file systeem. Vervolgens wordt de MySQL server opgestart. Ook die zal beginnen met het herstel van eventuele halve datastructuren, zodat een stabiele database-toestand wordt bereikt. We zijn dan enige seconden verder, en dan al is de slave database server beschikbaar als backup. Mocht de master weer opkomen, dan kan de slave weer terug worden gebracht in de secundaire modus door de MySQL server en het file systeem inactief te maken. Master en slave wisselen de wijzigingen via DRBD uit en zijn daarna weer klaar voor actie. Uiteraard zal ook de master wel de MySQL server moeten stoppen en starten om de bijgewerkte data te vinden.
Deze werkwijze wordt door MySQL AB (de makers van MySQL) uitgedragen als een goede opbouw van redundantie met MySQL. De werkwijze met losse, generieke componenten is ook typerend voor Unix. Dus hoewel het wat onsamenhangender lijkt dan een geïntegreerde oplossing van bijvoorbeeld Oracle, is het wel degelijk een solide, betrouwbare oplossing voor het vraagstuk van een redundante database.

Conclusie

Het is mogelijk om MySQL bovenop DRBD te draaien om een redundante database te verkrijgen. Daarbij is geen load sharing mogelijk, maar wel een backup systeem dat in luttele seconden paraat is. De oplossing met DRBD lijkt vooral geschikt voor zo'n opzet met servers op korte afstand. Voor de langere afstand biedt MySQL zelf al een redundantiemechanisme dat afdoende werkt.

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