Clusteren met MySQL (2)

Zeer beschikbaar: DRBD als solide basis

Rick van Rein

In de vorige aflevering van dit drieluik bespraken we hoe MySQL bovenop DRBD kan worden gedraaid om de beschikbaarheid te verhogen. Maar met DRBD kan nog veel meer.

Het algemene idee van DRBD is het verschaffen van een block device die oogt als een gewone, lokale harde schijf, maar die in wezen gerepliceerd wordt naar minstens één ander systeem. En omdat er van alles is wat kan werken op een block device, kun je dus ook van alles met DRBD. Het is dan ook de basis onder HA-Linux, en niet onder HA-MySQL of een andere specifieke oplossing.

Synchronisatie

Een belangrijk punt bij dit soort systemen is natuurlijk hoe het omgaat met durability; ofwel, het gegeven dat een toegezegde opslagpoging is gelukt, en vastligt tot er iets anders overheen wordt geschreven. Op het nivea van programma-interfaces werkt dit aan de hand van een aanroep van de sync() functie. Deze keert pas terug nadat alle tot dusverre naar opslag verstuurde gegevens vastliggen.

DRBD kan hier op een aantal manieren mee omgaan, afhankelijk van hoe strak de koppeling is gelegd tussen de systemen die het block device delen. In de meest strakke uitvoering moeten alle systemen lokaal een sync() hebben uitgevoerd, en bereik je dus durability op alle systemen. In een wat lossere variant keert sync() op een door DRBD beheerd block device terug zodra de lokale opslag geschied is, en de opdracht klaarstaat voor verzending naar de andere systemen. Deze lossere variant kan volstaan, en is soms noodzaak als de afstand tussen de systemen groot is. Eigenlijk is DRBD geschikt voor elke toepassing die gebruikmaakt van sync() om durability te bewerkstelligen – en toepassingen die dat niet doen hebben geen basis om zeker te weten dat de opslagpoging gelukt is, dus eigenlijk kunnen ook die ook goed werken op DRBD.

Wat eigenlijk dus vooral van belang is, is of de toepassing die je bovenop DRBD draait gebruik maakt van sync() en dus of die toepassing durability zeker stelt. Uiteraard hebben databases (zoals MySQL in de vorige aflevering) storage engines die dat doen. Maar ook veel andere toepassingen; bijvoorbeeld kun je even goed de Cyrus IMAP daemon op DRBD draaien. Want Cyrus zal geen "okay, ik heb het" terugmelden op een aangeboden e-mail voordat een sync() gelukt is.

Merk op dat hier een lijn door de software te volgen is om de opslag zeker te stellen. Het LMTP protocol (een eenvoudige variant op SMTP voor lokale aflevering) dat door Cyrus wordt ondersteund meldt terug of de opslag van een e-mail gelukt is of niet; op basis hiervan besluit de zendende mail server of de aanbieding nogmaals geprobeerd moet worden of dat hij als gelukt mag worden beschouwd. En dankzij de sync() in Cyrus IMAP, gecombineerd met de gewenste semantiek voor de afhandeling ervan in DRBD, is goed zeker te stellen dat de mail is aangekomen. Mail wordt transactioneel afgehandeld, dus je wilt in ieder geval de meest strakke semantiek hebben voor de afhandeling van sync() – en doordat het geen interactief protocol is blijft dat gelden voor systemen op grote afstand: er is geen haast bij de aflevering immers. Dit soort redenaties is nuttig om optimaal gebruik van DRBD te maken.

Kluwens van updates

De performance van allerlei systemen wordt er beter op als er werk in een wachtrij kan klaarstaan. Zo worden pieken in werkaanbod verspreid en er kan zelfs aan de volgorde worden gewijzigd om voordeliger te routeren. Dit compliceert echter wel de wijze waarop DRBD de updates en terugmeldingen beheert. Het meest algemene model mogelijk is een set van blokken die ooit eens naar de schijf moeten worden getransporteerd; als daar volgordes tussen vastliggen, dan wordt dat per tweetal blokken vastgelegd. De onderliggende datastructuur zou Partially Ordered Set kunnen zijn: een graaf met pijlen die volgorde-relaties vastlegt, maar vrij van gerichte cycles. Dit kan in allerlei volgordes daadwerkelijk worden opgeslagen. Dit algemene model is echter zo complex dat het weinig in de praktijk wordt toegepast.

Een praktische tussenvorm, die zowel concurrency mogelijk maakt alsook af en toe een stuk zekerheid biedt over afgerond werk, is de Write Barrier. Dit concept komt voor in SCSI en SATA, en bovendien in DRBD. De Linux-kernels van de 2.6 reeks introduceren dit concept ook in de rest van de blokafhandeling van het operating systeem.

Write Barriers scheiden sets van updates, zodanig dat de ene set wordt afgerond voordat aan de volgende set wordt begonnen, en kunnen dienen om een sync() op te leggen, bij de implementatie van journaling, enzovoort. Het voordeel van de aanwezigheid

van Write Barriers in alle niveaus van kernel en DRBD tot zelfs de hardware, is dat DRBD updates via het netwerk kan verzenden op een manier die uiteindelijk zelfs in de hardware wordt uitgevoerd zoals gewenst – er is dus geen noodzaak om allerlei modellen in elkaar om te zetten, zodat in principe veel van de kennis in de hardware zelf kan worden toegepast om het schrijven naar schijf optimaal te laten verlopen.

DRBD is gemaakt om te overleven in tijden van crashes. Daarbij is het ideaal om zo snel mogelijk van fouten te herstellen, bij voorkeur zonder hele schijven over te moeten zenden, maar in elk geval nooit te weinig van de data – want een verschil mag na herstel nooit blijven bestaan. Er is dus informatie nodig over de blokken die mogelijk veranderd zijn. Deze blokken worden door DRBD bijgehouden in een Activity Log. Alle blokken die niet in die log voorkomen zijn gegarandeerd niet veranderd.
Het ergste dat kan gebeuren is een toestand waarin het netwerkverkeer tussen hosts zodanig is ontwricht dat beide denken dat ze de enige host in leven zijn, en als master moeten opereren. In zo'n situatie wordt aan beide zijden geschreven. Door het bijhouden van een Activity Log is alles wat niet strijdig is (dezelfde blokken beschrijft) daardoor herintegreerbaar.

Cluster file-systemen

De opslagstructuur van een block device is nogal rauw, en meestal wordt er dan ook een file-systeem op gedraaid. In het vorige deel gaven we als voorbeeld een journaling file-systeem zoals ext3, dat snel kan worden opgestart op een slave na een crash van de master. Prima als je slechts één systeem tegelijk actief wilt hebben, maar niet als je de netwerklast wilt verdelen over meerdere systemen.
Als meerdere systemen tegelijk actief moeten zijn, dan kan een gewoon file-systeem niet langer gebruikt worden. Het is dan tijd voor cluster file-systemen. Deze zijn speciaal ontworpen om te draaien bovenop gedeelde block devices zoals DRBD. Ze weten dat er andere locaties zijn die dezelfde blokken kunnen grijpen, en lossen dat op door slim met de structuren om te springen. DRBD zelf is prima in staat om meerdere actief schrijvende partijen op het gedeelde block device te ondersteunen, tenzij meerdere partijen dezelfde blokken op de schijf 'claimen'. Wanneer meerdere partijen dezelfde blokken proberen te beschrijven dan zorgt DRBD dat alle partijen dezelfde blokken terugvinden, maar wat gevonden wordt is een onvoorspelbare selectie uit hetgeen geschreven wordt. Dit is vergelijkbaar met de situatie waarin meerdere processen ongecoördineerd parallel draaien, en naar dezelfde blokken op een lokale harde schijf schrijven. Ook dan is het niet bekend welke blokken uiteindelijk overblijven, alleen dat de keuze consistent is. (Als je Niels Bohr zo'n systeem zou laten beschrijven dan zou hij beweren dat de blokken in twee toestanden verkeren tot het moment dat je de eerste waarneming doet, en dat de waarde op dat moment vast komt te liggen.) Een hogere laag kan veel efficiënter omgaan met blokkenverdeling over schrijvers dan DRBD, door de extra kennis. Zo kan de verzameling aan vrije blokken (een concept dat DRBD niet eens kent) bijvoorbeeld worden gepartitioneerd over verschillende nodes, zodat het cluster file-systeem een lokale set van beschrijfbare blokken heeft. Als per blok een gedistribueerde lock zou moeten worden aangevraagd dan zou het een stuk minder soepel lopen!

De botte bijl

Het moge duidelijk zijn dat DRBD tamelijk verfijnd met synchronisatie werkt. Wat dat betreft loopt het soms zelfs voor op Linux zelf, want dat werkt met een relatief 'botte bijl'. Ga maar na: Een proces doet een sync() om alles, alles, alles weg te schrijven. Dus niet slechts dat ene e-mailtje, maar ook een andere die in een afzonderlijke thread binnenkomt maar die nog maar half op schijf gezet is.
De sync() kan geen kwaad voor dat halve mailtje, maar het is natuurlijk wel zonde dat er ook tijd nodig is om die inhoud *durable* te maken. Er is geen mechanisme om de werking van sync() te verfijnen, tenzij natuurlijk afzonderlijke processen worden gebruikt. Maar een nieuw proces voor elk binnenkomend mailtje is ook niet altijd handig, vanwege de efficiency van een proces in vergelijking met een thread. Ofwel, er is nog voordeel te behalen met een verfijnder proces voor de synchronisatie met de blokkenopslag. DRBD geeft al een beetje aan hoe. En de genoemde partieel geordende verzamelingen van blokupdates zouden het nog fijnmaziger kunnen maken.

Hogerop zoeken

Het heeft er veel van weg dat DRBD slechts een startpunt is voor het zeer beschikbaar krijgen van data en toepassingen; er zijn diverse zaken die naar hogere lagen wijzen, die nu ook geoptimaliseerd zouden kunnen worden. En dat kan een tijdje zo doorgaan – als MySQL bijvoorbeeld op een cluster file-systeem draait, kun je je gaan afvragen of MySQL zelf niet cluster-geschikt kan worden gemaakt. Het ontwerpen van een applicatie op deze vorm van clustering is niet gemakkelijk. Het is alsof een kasteel wordt gebouwd bovenop een moeras; immers, de onderliggende blokken kunnen voortdurend veranderen maar toch moet er een solide opslag-service worden verleend. Concrete problemen die hier ook bij horen zijn caches die invalide kunnen worden zonder dat dat expliciet gemeld wordt. Vrijwel elke applicatie bouwt in het geheugen een lijst of index op met verwijzingen naar stukken data die daarmee snel gevonden kunnen worden. Al deze datastructuren zijn aan spontane verandering onderhevig als het clusteren wordt doorgevoerd. Pas wanneer dergelijke caches (die meestal niet eens als cache worden ervaren door de programmeurs) geïntegeerd worden met een veranderingen-notificatie vanuit DRBD kan er serieus naar een clusterende database worden gestreefd.

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