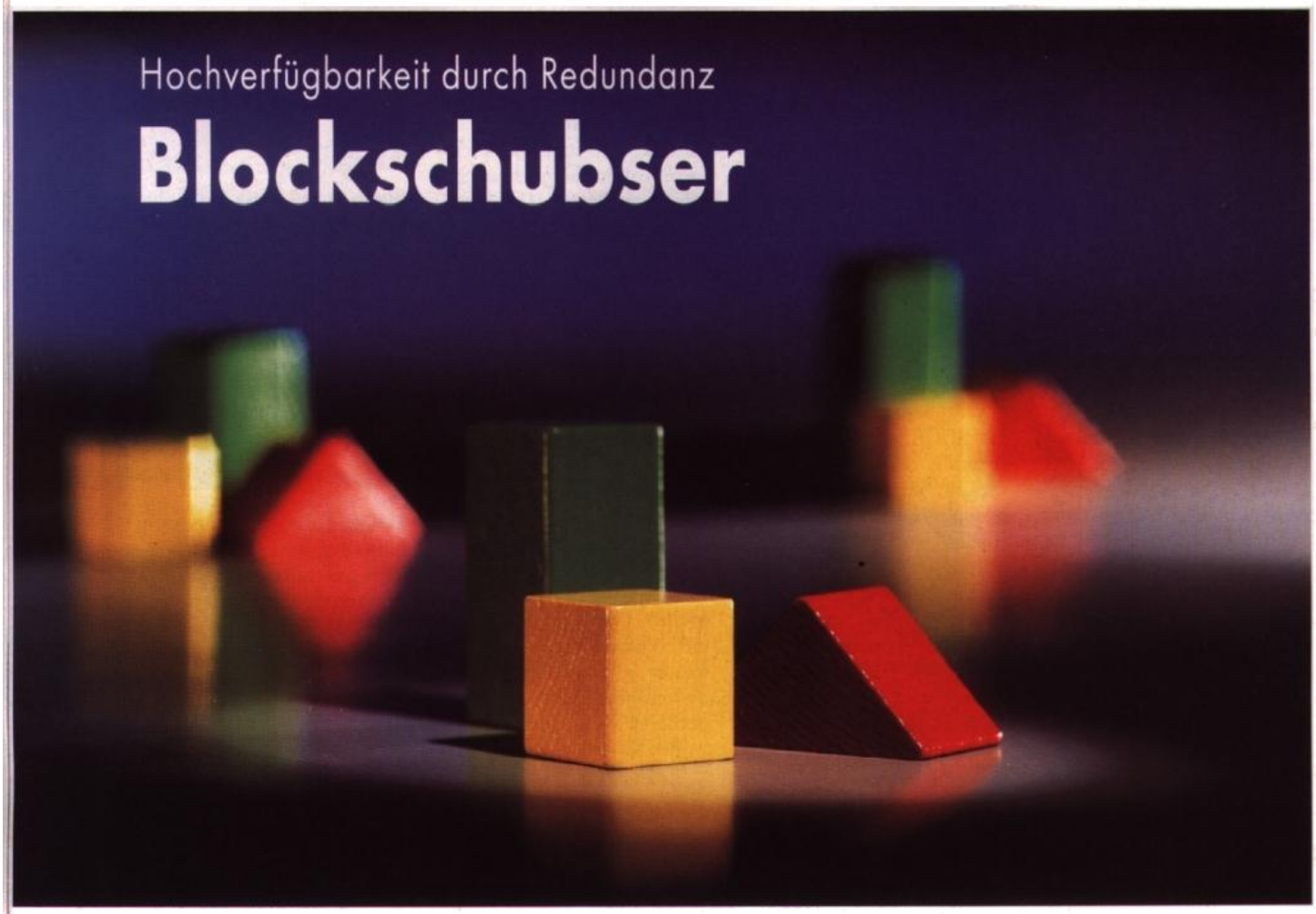


Hochverfügbarkeit durch Redundanz

Blockschubser

Philipp Reisner

Festplattenspiegelung ist eine eingeführte Methode zur Erhöhung der Verfügbarkeit von Daten – Clustering beugt einem einzelnen Systemausfall vor. Mit DRBD lassen sich unter Linux so genannte 'Shared Nothing'-Cluster realisieren, deren Knotenabstand um Größenordnungen über denen liegt, die mit Shared Discs arbeiten, bei um ebenfalls Größenordnungen geringeren Kosten.

RAID1 ist eine weithin bekannte Methode zum Erhöhen der Verfügbarkeit eines Servers. Es verhindert den Datenverlust beim Ausfall einer Festplatte. Aber das Spiegeln innerhalb einer Maschine nutzt nichts, wenn andere Teile als die Festplatte ausfallen. Der geringe Abstand zwischen den beiden Festplatten schützt die Daten auch nicht vor Unfällen wie Feuer.

Mit HA-Clustern [1] lässt sich der Ausfall von Server(komponente)n weitgehend abfangen. In der Regel verfügen solche Systeme über Shared Discs, bei denen die Platten meist in einer separaten RAID-Box untergebracht sind – mit nach wie vor geringem Abstand zueinander.

DRBD ist ein Gerätetreiber für Linux, der es erlaubt, Shared-Nothing-Cluster mit verteilten Plattenspiegeln aufzubauen. Den Anwendungen gegenüber stellt sich das Modul als normales Block Device dar. Dieser Ansatz hat nicht nur den Vorteil, dass der Abstand zwischen den beiden Kopien der Daten um Größenordnungen über dem mit Shared-Disc-Konfigurationen realisierbaren liegen kann. Obendrein sind die Kosten um ebenfalls Größenordnungen geringer.

Für den Aufbau eines einfachen Clusters benötigt man einen Synchronisierungsmechanismus, eine Überwachungskomponente, ein Dateisystem und letztlich einen Daemon, der den Dienst zur Verfügung stellt.

Bausteine des Systems

DRBD beschränkt sich auf den Bereich der Synchronisierung der Block-Devices der Cluster-Knoten. Folglich ist das so genannte Split-Brain-Problem – die Verbindung zwischen den Nodes fällt aus und beide nehmen an, der jeweils andere sei ausgefallen – nicht Bestandteil dieses Artikels, dieses sollte die Cluster-Monitor-Software berücksichtigen. Abhilfe verspricht das im CVS bereits implementierte Metadatenmanagement, es verwaltet genug Information, um die Split-Brain-Situation zu erkennen. Was für eine automatische Rückkehr in den opera-

tiven Zustand derzeit noch fehlt, ist die notwendige Unterstützung durch den Cluster-Manager.

Letzterer überwacht das Arbeiten des primären Knotens und stößt bei dessen Ausfall den Failover-Prozess an. Da der Zeitpunkt für eine Übernahme nicht vorhersehbar ist, muss das Dateisystem in jedem denkbaren Zustand online gehen können. Nach einem Failover muss es sich dem Benutzer im selben Zustand wie davor präsentieren. Eine wichtige Vorbedingung, die Journaling Dateisysteme unter anderem erfüllen.

Schließlich muss der Service mit einem Failover umgehen können. Dazu sollte er entweder auf einem zustandslosen Netzprotokoll basieren oder über eingebaute Wiederholungsmechanismen verfügen. Greift der Service auf Dateien zu, darf das Vorhandensein von bei einem Ausfall zurückgebliebenen inkonsistenten Files nicht die Verfügbarkeit des Dienstes gefährden.

Blockgerätetreiber unter Linux

Zunächst ein Blick auf die Eigenschaften eines Block Device. Unter Linux erfolgen alle Zugriffe auf blockorientierte Geräte über den Buffer Cache des Kernels. Dadurch kann es passieren, dass der schreibende Aufruf *ll_rw_block()* einen Rückgabewert liefert, bevor die Daten physikalisch auf der Platte gelandet sind. Erschwerend kommt hinzu, dass das Block Device aus Optimierungsgründen die Reihenfolge der Schreiboperationen ändern kann. Per *buffer_uptodate()* lässt sich prüfen, ob ein Block wirklich seinen Weg auf die Platte gefunden hat, via *wait_on_buffer()* kann die Anwendung warten, bis das Gerät den Block physikalisch geschrieben hat.

Aus Sicht des Geräts stellt sich die Situation folgendermaßen dar: Mit einem *do_request()* kommen die Datenblöcke, ein Aufruf von *end_request()* bestätigt das Ende des Schreibvorgangs. Das Device darf die Blöcke dabei zu Optimierungszwecken umsortieren. Beim Aufbau eines über das Netz gespiegelten Block Device stehen daher auf der einen Seite die korrekte Signalisierung des Abschlusses von I/O-Operationen und auf der anderen der maximale Datendurchsatz. DRBD bietet je nach Anwendungsfall drei Protokolle (A, B, C), um den goldenen Mittelweg zu finden.

Protokoll A

Über Protokoll A signalisiert der Treiber die Beendigung einer Schreibanforderung, sobald der Block sowohl auf der lokalen Platte steht als auch über das Netzinterface den Rechner verlassen hat. Das heißt, das System meldet die Beendigung der Operation, ohne zu wissen, ob der Block auf der Platte des Spiegels angekommen ist respektive ankommen wird.

Verarbeitung von I/O-Aufrufen (Abb. 1).

Da TCP ein Peer-to-Peer-Protokoll ist, funktioniert dieser Kommunikationskanal nur, wenn beide Teilnehmer arbeiten. Fällt nun der sendende Knoten unmittelbar nach dem Abschicken eines Datenpaketes aus, kann TCP nicht sicherstellen, dass es auch beim Empfänger ankommt. Übernimmt jetzt der zweite Knoten den Dienst, kann es sein, dass er nicht alle Daten des ersten Node bekommen hat.

Für transaktionsbasierte Anwendungen ist dieses Verhalten nicht akzeptabel, Protokoll A eignet sich daher nur, wenn es auf hohen Datendurchsatz ankommt und nicht auf Transaktionseigenschaften. Ein Anwendungsfall wäre der Betrieb über so genannte Long Links, das heißt, Verbindungen mit hoher Bandbreite und großen Latenzzeiten, die praktisch größere Datenmengen in sich speichern.

Protokoll B

Protokoll B betrachtet einen Schreibvorgang als abgeschlossen, wenn die Daten lokal physikalisch auf der Platte sind und vom zweiten Knoten die Empfangsbestätigung vorliegt. Im Gegensatz zur Variante A bleiben beim Ausfall eines Node die Transaktionseigenschaften erhalten. Lediglich ein Sonderfall kann zur Verletzung der Transaktionssicherheit führen: Beide Knoten fallen kurz nacheinander beispielsweise durch Stromausfall aus. Der primäre teilt einem Client mit, dass eine Transaktion abgeschlossen ist, danach fällt er aus. Der zweite Node hat zwar den Datenblock empfangen und dies dem Master bestätigt, fällt aber aus, bevor er die Daten auf die Platte schreiben kann. Entscheidet der Cluster-Manager nach dem Stromausfall, das der bisherige Fallback-Knoten den Dienst übernehmen soll, so ist die Transaktion für ihn unvollständig, obwohl der Client seine Bestätigung bekam. Diese Schwierigkeit ließe sich umgehen, wenn der Cluster-Manager nach einem doppelten Ausfall den Service wieder auf dem vorher aktiven Rechner startet – eine Funktion, die im Zusammenspiel mit Heatbeat gerade in der aktuellen Version der CVS-Tree entsteht.

Für die komplette Spiegelung eines Rechenzentrums über den Atlantik eignet sich Protokoll B nicht besonders, weil es das sendende System bei solchen Verbindungen mit großen Latenzzeiten ausbremst. Dies liegt an der auf 128 Einträge beschränkten Request Queue des 2.2er-Kernels, von denen DRBD für das Schreiben nur 42 Slots benutzen kann. Ab Version 2.4.x sieht die Situation etwas besser aus, da hier der Kernel eine Request Queue pro Gerät verwaltet. Ist kein Slot verfügbar, blockiert er die schreibbereite Anwendung, bis ein neuer frei wird respektive ein älterer Auftrag zum Abschluss kommt.

Protokoll C

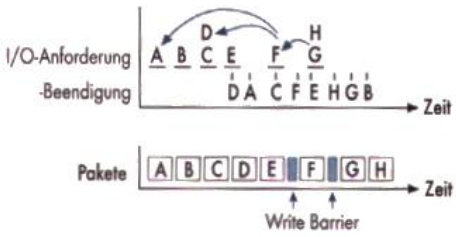
Mit Protokoll C gilt eine Schreiboperation als komplett, wenn der Sender vom Standby-System die Meldung bekommt, dass dieses den Block physikalisch geschrieben hat. Es garantiert die Transaktionssemantik in allen Ausfallszenarien.

Bei Protokoll B und C muss man bei Ausfall des verbindenden Netzes oder des zweiten Knotens den Empfang der noch ausstehenden

iX-TRACT

- DRBD ist ein Kernel-Modul zum Aufbau eines 2-Node-HA-Clusters unter Linux.
- Da sich DRDB als Block Device darstellt, ist der Einsatz für Anwendungen völlig transparent.
- In der Theorie beschränkt nur die Netztopologie den Abstand zwischen den Cluster-Knoten.
- In der Praxis erreicht DRBD zwischen 50 und 98 % des theoretisch maximal möglichen Datendurchsatzes.

Bestätigungen simulieren. Sonst würden die Slots der Request Queue nicht mehr freigegeben und das System blockiert.



Aus I/O-Anforderungen resultierende Paketreihenfolge im Netz (Abb. 2)

Auf die Reihenfolge kommt es an

Für viele Dateisysteme spielt die Reihenfolge, in der die Datenblöcke auf der Platte landen, eine entscheidende Rolle. Beispielsweise muss ein Journaling Filesystem vor jeder Aktualisierung der Metadaten eine Transaktion in den Journal-Bereich schreiben, wobei der 'Commit Record' der letzte in der Reihe sein muss.

Linux wartet derzeit explizit auf den Abschluss jener Schreiboperationen, die vor dem nächsten Schreiben auf der Platte sein müssen. Damit DRBD Schreiboperationen auf dem sekundären Knoten in der gleichen Reihenfolge durchführt, ergäbe sich folgender Algorithmus:

Empfange den Block vom Netz, schreibe diesen auf die Platte und warte, bis die Operation abgeschlossen ist.

Es ist offensichtlich, dass dies sowohl den primären Knoten bremsen als auch das lokale I/O-System nicht auslasten kann. Verbesserung verspricht die Parallelisierung von Empfang über das Netz und Platten-I/O ohne die Anforderungen des Dateisystems zu verletzen. Hinweise dazu liefert eine Analyse der zeitlichen Abfolge der Schreibbefehle.

Bei der Darstellung entspricht die zeitliche Ausdehnung der Übergabe von Datenblöcken an das I/O-System einem Aufruf von *ll_rw_block()*. Mehrere Blöcke können gleichzeitig abgeschickt werden und die Bearbeitung kann unterschiedlich dauern, da der Scheduler die Bearbeitung blockieren beziehungsweise verzögern kann. I/O-Beendigung entspricht dem Zeitpunkt, an dem das 'uptodate'-Bit im korrespondierenden Buffer-Head gesetzt ist.

In Abbildung 1 kann zwischen den Datenblöcken A, B, C, D und E keine Abhängigkeit vorliegen, da vor deren Schreibbefehlen keine Schreiboperation beendet war. Da der Auftrag für Block F erst nach der Beendigung von Block A und D kam, kann dieser von den beiden anderen abhängen, ebenso kann eine Beziehung von G und H zu A, C, und D bestehen. Ob diese Abhängigkeiten tatsächlich bestehen, weiß nur das Dateisystem, aber ein Festplattentreiber – und so etwas stellt DRBD dar – darf keine potenzielle Abhängigkeit verletzen.

Auf Basis der Analyse lässt sich ein Algorithmus angeben, der jeden Block einer Menge, einem so genannten Epoch Set, zuordnet. Signalisiert einer der Blöcke eines Epoch Set das Ende des Schreibvorgangs, schreibt DRBD eine Schranke (Write Barrier) auf das Netz und leert die Menge.

Auf der Empfängerseite kann der Node sein Vorgehen anpassen. Wenn er ein Paket empfängt, muss er prüfen, ob es sich um ein Datenpaket handelt. Wenn ja, kann er den Befehl zum Schreiben geben, sich den Datenblock in einer Menge merken und den nächsten Block empfangen. Nur wenn das Paket eine Write Barrier ist, muss er auf den Abschluss aller Schreiboperationen der Menge warten und diese anschließend leeren. Abbildung 2 zeigt den Zusammenhang zwischen Blockabhängigkeiten und den Paketen auf dem Netz.

Neben der zu einer normalen Schreiboperation gehörenden Replikation von Datenblöcken muss ein Mechanismus existieren, den Inhalt der gespiegelten Festplatten zu synchronisieren. Dies kann beispielsweise nach dem Ausfall eines Node und dem Einbinden eines neuen Fallback-Knotens nötig sein.

Abstimmung ist alles

DRDB implementiert die Synchronisierung so, dass sie den normalen Betrieb des aktiven Knotens nicht beeinträchtigt. Sie läuft ähnlich wie die normale Spiegelung und nutzt nur einen Teil der zur Verfügung stehenden Bandbreite des Netzes.

Üblicherweise erfolgt die Aktualisierung der Daten durch blockweises Kopieren vom aktiven Knoten zum Standby-Node. Diesen Full Synchronisation genannten Modus benutzt DRBD beispielsweise, wenn es einen Knoten zum ersten Mal in den Cluster einbindet.

Fehlt einem Node nur kurzzeitig die Verbindung zum Cluster, reicht es jedoch, nur die in der Zwischenzeit geänderten Blöcke zu übertragen – mit der 'Quick Synchronisation'. Bei dieser markiert DRBD in einer im Arbeitsspeicher befindlichen Bitmap alle seit dem letzten Acknowledgement-Paket modifizierten Blöcke. Weil der ausgefallene Node keine Bestätigungen mehr schickt, bestätigt der Treiber alle Schreibbarrieren der Protokolle A und B und markiert alle Blöcke vom noch nicht bestätigten Epoch Set als 'Not In Sync'.

Da die Blockliste nur im RAM vorliegt, muss als einzige Bedingung der aktive Knoten zwischenzeitlich kontinuierlich laufen. Er setzt die 'Not In Sync'-Bits erst zurück, wenn der Standby-Node bestätigt, dass er den Block auf die Platte geschrieben hat. Daher stellen weitere Störungen des zweiten Knotens während der Synchronisation keine Gefahr für die Datenkonsistenz dar.

Vor der Synchronisation enthält das gespiegelte System einen veralteten aber konsistenten Datenbestand. Während des Abgleichs ist das Dateisystem nicht konsistent. Fällt zu diesem Zeitpunkt der aktive Node aus, hilft nur noch ein Backup weiter.

Leistung durchaus praxistauglich

In der Praxis hängt der realisierbare Datendurchsatz von dem der jeweiligen Festplatten sowie der Latenzzeit des verbindenden Netzes ab. Messungen im Rahmen der Diplomarbeit des Autors [2] liefern hierzu konkrete Zahlen. Die Latenzzeiten ermittelten die Tester, indem sie über mehrere Messreihen die für 50 Ping-Pakete benötigte Zeitspanne ermittelten, die verfügbare Netzbandbreite lieferte ein Skript, das per *rsh* auf dem entfernten Rechner Daten aus */dev/zero* in das lokale */dev/null* schaufelte.

Arbeitet DRBD ohne Verbindung (es muss Schreibaufforderungen nur an die lokale Platte durchreichen), erreicht es durchschnittlich 89,5 % des Plattendurchsatzes, bei einer sehr niedrigen Standardabweichung von 1,41 %. Im verbundenen Zustand erreicht Protokoll A im Durchschnitt 66,9 % des theoretischen Maximalwertes, dem Minimum von Platten- und Netzdurchsatz – bei einer Standardabweichung von fast 12 % allerdings mit deutlich größerer Streuung. Die beiden anderen Protokolle stehen erwartungs-

gemäß etwas schlechter da: Protokoll B erreicht mit gut 99 % fast die Leistung von A, während C immerhin noch gut 97 % des Durchsatzes von Protokoll B schafft.

Zusammenspiel im Clustereinsatz

DRBD stellt im System eine virtuelle Platte mit gemeinsamen Datenzugriff bereit. Die Steuerung der Cluster-Knoten muss über geeignete Monitorsoftware wie Heartbeat erfolgen. Da DRBD auf allen Knoten im Cluster läuft, muss dessen Management sicherstellen, dass die Anwendungen auf den Knoten laufen, die über den aktuellen Datenbestand verfügen und dass eine Synchronisation die anderen so schnell wie möglich auf den aktuellen Stand bringt. Diese Anforderungen kann Heartbeat allein nicht erfüllen. Als Entscheidungshilfe verwaltet DRBD einige wenige Metadaten. Zum einen zeigt ein Inkonsistenz-Flag an, dass der Knoten Ziel eines Synchronisierungsprozesses ist und zur Zeit inkonsistente Daten enthält. Zum anderen gibt ein Generation Counter Auskunft über den Systemstatus. Dieser besteht aus vier Teilen:

human-intervention-count: Diesen zählt DRBD hoch, wenn Einwirkungen des Administrators den Status des Clusters verändern. Er besitzt Priorität vor den anderen beiden Zählern.

connected-count: Inkrement, wenn der Status des laufenden DRBD-Clusters sich ändert oder der sekundäre Knoten aus dem Cluster wegfällt.

arbitrary-count: nimmt zu, wenn die Managementsoftware den Status des Clusters ändert und der Knoten nicht Teil eines laufenden Clusters ist.

primary-indicator: Beim aktiven Node beträgt der Wert '1', sonst '0'.

Diese Daten verwalten die Nodes in nicht-flüchtigen Speicherbereichen. Existiert nur ein als aktiv markiertes System im Cluster und die Kommuniaktion zwischen den Knoten funktioniert, so gilt dessen Generation Counter vor allen anderen. Kann ein Rechner beim Systemstart nicht mit seinem Partner kommunizieren, so muss der Administrator die Entscheidung treffen.

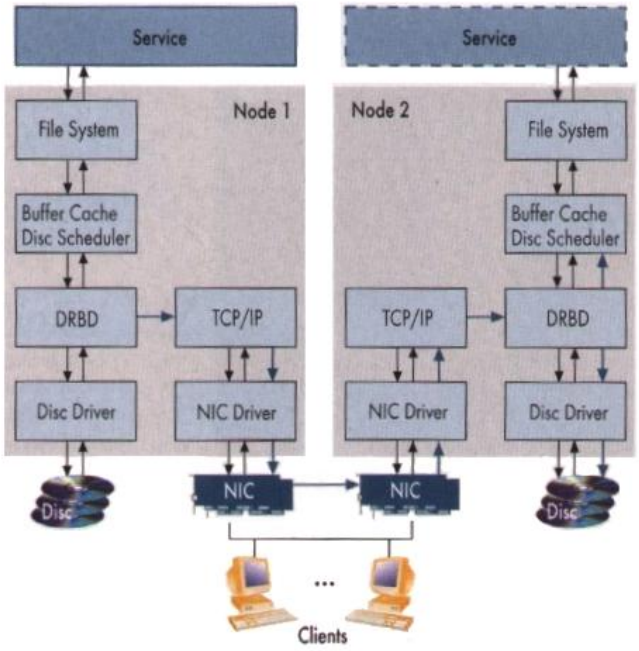
Bei der Entscheidung, welcher Knoten die konsistenten Daten hat und den Dienst bekommen soll, arbeitet das System die Einträge in der oben beschriebenen Reihenfolge ab. Existiert ein Knoten mit gesetztem Inkonsistenz-Flag, besitzt der andere die aktuellen Daten. Sind die ersten Einträge des Generation Counter verschieden, fällt die Entscheidung für den Rechner mit dem höheren Wert. Eine Auswertung der nachrangigen Zähler erfolgt nur, wenn bei den vorhergehenden gleiche Werte vorliegen.

Glossar

Shared Disc: Festplatten, auf die alle Knoten eines Clusters zugreifen können.

Shared Nothing: Genau ein Knoten eines Clusters hat Zugriff auf die gemeinsamen Platten-Devices.

Split Brain Problem: Zerfällt ein Cluster durch Netzausfall in zwei Teile, müssen Strategien implementiert sein, wer welche Dienste übernimmt.



Beim Einsatz in einem Cluster sorgt der DRBD-Treiber für eine transparente Spiegelung der Daten über das Ethernet (Abb. 3).

Praktische Erfahrungen mit DRBD

Bei der österreichischen CUBiT IT Solutions befindet sich DRBD auf zwei Linux-Clustern seit neun Monaten im produktiven Einsatz. Während in dieser Zeit das eine System – ein eher wenig ausgelasteter Server mit einer PostgreSQL-Datenbank und einem Messaging-System – störungsfrei lief, traten beim zweiten – einem starker Last ausgesetzen NFS- und MySQL-Server – erhebliche, allerdings nicht DRBD-bedingte Schwierigkeiten auf.

Neben Hardwarefehlern im Bereich von CPU und Mainboard führten SCSI-Probleme bei den RAID-Controllern sowie Kernel-Bugs, die erst bei hoher Speicherauslastung auftraten, zu Ausfällen des primären Servers. In allen Absturzfällen verhielt sich die Lösung aus DRDB-basiertem, via Heatbeat überwachtem Cluster wie erwartet – die Kundenapplikation liefen nur durch die Failover-Zeitspanne unterbrochen ohne Datenverluste oder -inkonsistenzen weiter.

Lediglich bei der Synchronisierung nach einem Ausfall eine Knotens besteht noch Optimierungsbedarf, sie dauert momentan noch zu lange. So kommt die stabile Version 0.5.7 nicht über 0,7 MByte/s heraus, während die Entwicklerversion in ersten Tests bis zu 5 MByte/s erreichte. (avr)

PHILIPP REISNER

ist Dipl.-Informatiker und arbeitet als Leiter der Entwicklungsabteilung bei der CUBiT IT Solutions GmbH, Wien.

Literatur

[1] Kai Dupke, Ralph Hülsenbusch; Hochverfügbarkeit; Getreue Diener; Der Weg zu Hochverfügbarkeitslösungen; *iX* 4/2001, S. 89

[2] Philipp Reisner; Festplattenspiegelung übers Netzwerk für die Realisierung hochverfügbarer Server unter Linux; www.complang.tuwien.ac.at/Diplomarbeiten/reisner00.ps.gz

[3] Gregory F. Pfister; In search of Clusters; Prentice-Hall PTR, Upper Saddle River, 1998

[4] Alan Robertson; Linux-HA Heartbeat System Design; www.usenix.org/publications/library/proceedings/als2000/robertson.html