Ideen für ausfallsichere Datenbanken

Datenbank Cluster Konzepte

Linux, DRBD, MySQL und PostgreSQL

Felix J. Ogris (fjo@dts.de)

Version 1.0
2009-03-25

DTS Systeme GmbH

1 Prolog

Weder MySQL noch PostgreSQL besitzen native Funktionen, um sie in einem hochverfügbaren Cluster zu betreiben, bei dem der Ausfall eines Knotens innerhalb weniger Sekunden erkannt und durch einen zweiten Knoten kompensiert wird (wie z.B. bei Oracle RAC). Zwangsläufig müssen in einem Cluster alle Knoten Zugriff auf den Datenbestand besitzen bzw. die Möglichkeit hierzu haben. MySQL ist mit der Storage Engine *NDB* zwar clusterfähig, unterliegt jedoch zwei Einschränkungen. Zum einen hält die NDB im Betrieb sämtliche Daten im Arbeitsspeicher vor, zum anderen werden nicht alle Indizierungsarten und Fremdschlüsseltypen unterstützt, so dass gegenüber anderen Engines wie MyISAM oder InnoDB clientseitige Anwendungen angepasst werden müssen. In den kommenden Versionen 6.x von MySQL soll zumindest erstgenannte Beschränkung aufgehoben werden. Sowohl PostgreSQL als auch MySQL ermöglichen Datenreplikation im Master/Slave-Betrieb. Bei MySQL können die Slaves als read only-Kopien dienen und somit zur Performanceverbesserung in leseintensiven Anwendungen beitragen. PostgreSQL schreibt jede Veränderung der Daten in serielle *Write Ahead Logs* (WALs), die per Skript auf einen Slave kopiert werden können. Dieser Slave befindet sich bis zum Failover im ständigen Recovery-Modus, in dem die WALs eingelesen werden. Der Slave kann im Gegensatz zu MySQL nicht als read only-Kopie verwendet werden. Derartige Replikationssetups unterliegen jedoch einem zeitlichen Versatz zwischen Master und Slave(s). Auch Transaktionen, die auf dem Master committet wurden, sind nicht zeitgleich auf dem Slave sichtbar. Dies kann beim Absturz des Masters zu (unbemerktem) Datenverlust führen kann. MySQL gestattet sogar die Angabe einer Zeitdauer, die ein Slave nicht mit seinem Master verbunden sein muss, um dennoch beim Reconnect alle bis dato aufgelaufenen Änderungen zu erhalten. Für eine hochverfügbare Datenbank ist daher ein Master/Slave-Szenario ungeeignet.
Alternativ kann ein Clusterfilesystem wie OCFS2 oder GFS verwendet werden, um den Datenbestand zwischen allen Knoten gleich zu halten. Allerdings hielten nach unseren Erfahrungen beide Dateisystemen in einfacheren Setups als einem Datenbankcluster keinem Dauerbetrieb stand. GFS benötigt Tools wie CMAN, CLVM und GNDB aus der RedHat-Clustersuite. Schon ohne den Einsatz von GFS konnten wir mit diesen Tools keinen stabilen Betrieb in unserem Xen-Cluster herstellen, der lediglich Blockdevices clusterweit zur Verfügung stellen musste. OCFS2 implementiert die `mmap()`-Funktion nur rudimentär. So blockierten Webserverprozesse (lighttpd) ohne Fehlermeldung beim Ausliefern von wenigen, grossen Dateien. Ferner benötigen Clusterfilesysteme ein gemeinsames, zentrales Speichermedium wie z.B. ein SAN. Dieses muss mit zusätzlichem Aufwand per iSCSI oder Fibre Channel allen Knoten bereitgestellt werden.

2 DRBD

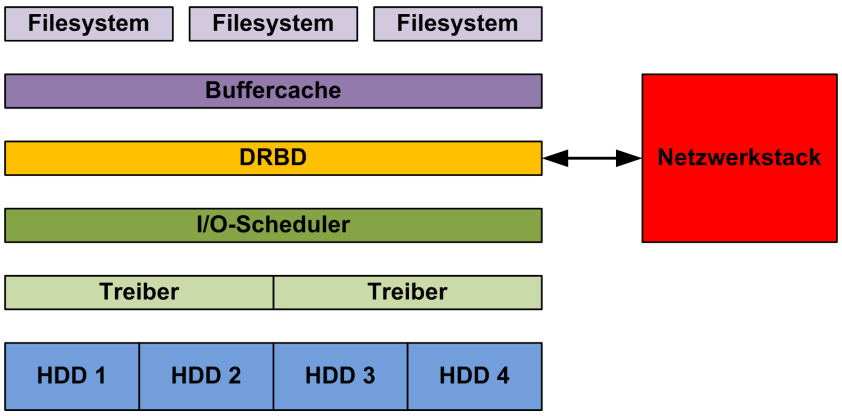

Abbildung 1: DRBD im Linux-Kernel

Das *Distributed Replicated Block Device* (DRBD) ist eine Erweiterung für den Linux-Kernel. Es wird für Blockdevices wie Festplatten oder Logical Disks eines Volume Managers konfiguriert. DRBD sitzt zwischen Dateisystem und Blockdevice (genauer: zwischen Buffercache und Diskscheduler) und überträgt alle lokalen Schreibzugriffe an einen zweiten Rechner (s. Abbildung 1). Dieser, analog mit DRBD aufgesetzte Slave speichert alle Änderungen ebenfalls auf seinen Festplatten. Es kann daher als RAID1 auf Rechnerebene über ein Netzwerk betrachtet werden. Im Regelbetrieb kann nur der Master auf die Daten zugreifen. Der Slave kann weder schreibend noch lesend auf seine Kopie zugreifen. Die Masterrolle wird ausschliesslich manuell durch eine Clustermanagementsoftware oder den Administrator gesetzt. Nach einem Ausfall eines Knotens wird dieser automatisch zum Slave und aktualisiert seine Daten von einem eventuell noch lebendem Master. Für ein Master-Slave-Szenario bedeutet dies:

- Fällt der Slave temporär aus (Hardwareschaden, Reboot, etc.), muss nicht eingegriffen werden, da dem Slave nach dem Reconnect wieder die Slaverolle zugewiesen wird und angefallene Änderungen vom Master nachzieht[1]
- Fällt der Master für eine kurze Zeitspanne aus, muss ihm nach dem Reconnect wieder die Masterrolle zugewiesen werde. Bei einem längeren Ausfall muss hingegen der Slave zum Master promoted werden.

DRBD kann je nach Anforderung in 3 verschiedenen Modi betrieben werden:

Protocol A Der Schreibvorgang wird auf dem Master als abgeschlossen betrachtet, sobald die Daten auf das lokale Blockdevice (sprich: Festplatte) geschrieben und dem Netzwerkstack übergeben wurden

Protocol B Der Schreibvorgang gilt als beendet, wenn die Daten auf die Festplatte geschrieben wurden und der Slave den Empfang bestätigt hat

Protocol C Der Schreibvorgang ist erst dann beendet, wenn sowohl Master als auch Slave die Daten auf ihre Festplatten gesichert haben und der Slave dies bestätigt hat.

[1] Dies stellt einen kritischen Zeitraum dar, da bei einem Ausfall des Masters während der Resynchronisation kein Knoten mehr über gültige, aktuelle Daten verfügt und somit der Cluster nicht betriebsbereit ist

Protocol A ist die unsicherste Variante, da beim Ausfall des Masters nicht sichergestellt ist, dass der Slave alle Daten erhalten hat. Es ist jedoch die schnellste Option, da die Replikation nicht auf Bestätigungen warten muss. Protocol B kann im Fall eines zeitnahen Absturzes beider Knoten zu Datenverlust führen, da ggf. die Daten auf dem Slave noch nicht auf die Festplatte geschrieben, dem Master aber schon der Empfang quitiert wurde. Protocol C verlangt hingegen, dass auf beiden Knoten die Daten nicht flüchtig gespeichert werden, eh dies dem Dateisystem und somit Anwendungen bestätigt werden. Es kann somit als *Two Phase Commit-Protocol* (2PC) betrachtet werden, das der letzten Anforderung einer ACID-konformen Datenbank, nämlich der Dauerhaftigkeit (durability), genügt. Allerdings ist es die langsamste Alternative, vor allem wegen der hohen Latenz der Festplatten und wegen der notwendigen Netzwerkkommunikation.

Da DRBD zwischen Buffercache und Festplattenscheduler sitzt (s. Abbildung 1), ist nicht sicher gestellt, dass die von einer Anwendung geschriebenen Daten unmittelbar an DRBD und somit an den zweiten Rechner weitergereicht werden. Dieses asynchrone Verhalten ist in den meisten Fällen gewünscht und bedeutet einen grossen Geschwindigkeitsvorteil, da häufig gelesene Blöcke im Hauptspeicher zwischengespeichert und durch Schreibzugriffe veränderte Bereiche optimiert (z.B. gebatched mit nur einem Festplattenzugriff) geschrieben werden. Bei einem Absturz des Rechners könnten daher trotz Einsatz von DRBD und eines Zweitrechners Daten verloren gehen oder ein defektes Dateisystem entstehen. Um den Buffercache für Schreibzugriffe effektiv zum umgehen, bieten sich 3 Möglichkeiten an:

- Das Dateisystem wird mit der Option `sync` gemountet
- Die Anwendung öffnet Dateien mit dem Flag `O_SYNC` bzw. `O_FSYNC`
- Die Anwendung ruft nach jedem Schreibvorgang die Funktion `fsync()` für die jeweilige Datei auf.

Ersteres bietet sich bei (Closed-Source-)Programmen an, die kein synchrones Schreiben ihrer Dateien unterstützen, oder bei alten Dateisystemen ohne Journaling wie ext2 an. Allerdings werden somit alle Dateien direkt und ohne Buffercache geschrieben, was für Logdateien, temporäre Files, etc. einen unnötigen Schreibrequest gegen die Festplatte darstellt. Öffnet die Anwendung hingegen alle Dateien expliziet mit der Sync-Option, werden alle Schreibzugriffe unmittelbar an die Hardware (und somit an DRBD) weitergeleitet. Effizienter ist es, wenn die Anwendung jede logische Operation mit einem Aufruf der Betriebssystemfunktion `fsync()` für die beteiligten Dateien beendet. So muss eine ACID-konforme Datenbank ohnehin nach jedem Commit einer Transaktion `fsync()` aufrufen, um sicherzustellen, das veränderte Daten dauerhaft gespeichert werden.

3 Anwendung

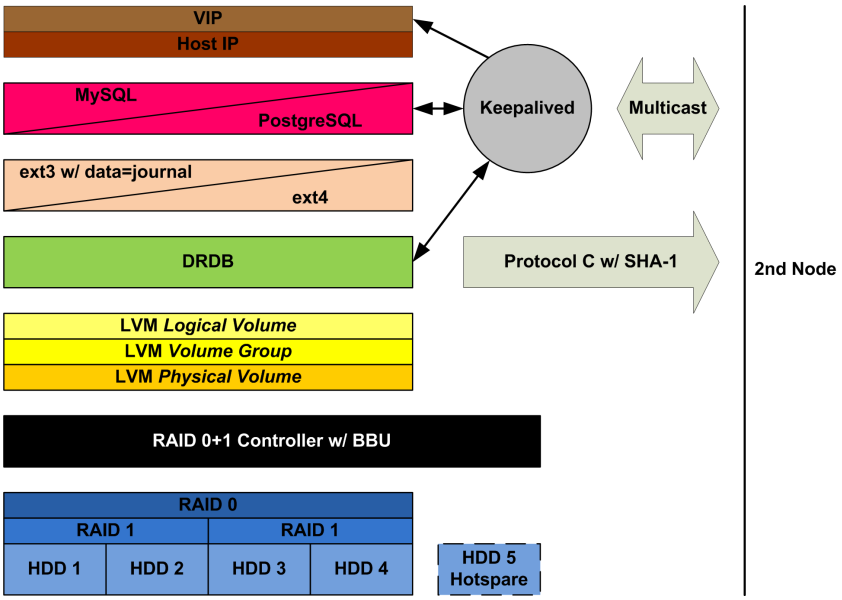

Abbildung 2: DRBD im Einsatz

Abbildung 2 zeigt den Protokollstack in einem typischen Active-Passive-Cluster mit DRBD. Für Datenbanken, auf die viele Benutzer(-sessions) mit unterschiedlichen Queries zugreifen, sollte ein RAID 1+0 verwendet werden, da RAID 5 für Random-I/O ungeeignet ist. Die von DRBD bereitgestellten Blockdevices können im laufenden Betrieb vergrössert und verkleinert werden. Daher sollte ein Logical Volume als Grundlage dienen, da LVM-Container ebenfalls online verändert werden können. Als Dateisysteme kommen ext3 bzw. zukünftig ext4 sowie xfs in Frage. Letzters kann jedoch nur vergrössert werden. Logical Volumes, die mit ext3 formatiert sind, sollten mit der Option `data=journal` gemountet werden, um nach einem Absturz oder Failover die Zeit für den Filesystemcheck kurz zu halten. MySQL wird wie folgt konfiguriert, um nach jeder Transaktionen ein `fsync()` zu erzwingen:

```
flush=1
sync_binlog=1
sync_frm=1
innodb_flush_log_at_trx_commit=1
```

Das Gegenstück zur `my.cnf` heisst bei PostgreSQL `postgres.conf`, die man entsprechend erweitert:

```
fsync=on
synchronous_commit=on
full_page_writes=on
```

Eine zentrale Rolle spielt keepalived. Der Dienst wird auf beiden Knoten gestartet und sorgt dafür, dass der jeweilige Master die Virtual IP erhält, über den Clients den Datenbankcluster ansprechen. Um den Datenbankdienst und DRBD anzusteuern, müssen individuelle Shellskripte o.ä. erstellt werden. Neben Start und Stop von MySQL bzw. PostgreSQL muss das Skript auch den Status des Datenbankservers ausgeben, damit

keepalived ggf. einen Failover einleiten kann. Ein weiteres Skript muss in der Lage sein, DRBD in den Master- oder Slavemodus zu setzen und den Status des zugrunde liegenden Blockdevices an keepalived zu melden, um auch bei einem Hardwarefehler auf den zweiten Node umschalten zu können.

4 Stonith

Wenn beide Clusterknoten Instanzen in einer Virtualisierungsumgebung wie VMware ESX(i), Xen, KVM, etc. sind, kann ein Stonith-Konzept (*Shoot the other node in the head*) ohne zusätzlichen Hardwareaufwand realisiert werden. Hierbei wird der virtuelle Server, der sich im Fehlerzustand befindet, per Funktionsaufruf über die Virtualisierungssoftware vom verbleibenden Master rebootet. Damit sich beide Knoten im Fehlerfall nicht gegenseitig die virtuelle Stromzufuhr nehmen, sollten die keepalived-Prozesse über eine zusätzliche Netzwerkverbindung (Heartbeat) kommunizieren.

5 Literatur

- http://www.drbd.org
- http://www.mysql.org
- http://www.postgresql.org
- http://www.keepalived.org/
- http://en.wikipedia.org/wiki/RAID
- http://en.wikipedia.org/wiki/ACID
- http://en.wikipedia.org/wiki/2PC