The Sysadmin's Daily Grind: DRBD

High-Availability Linux

In this month's SysAdmin section of Linux Magazine we cover high availability and software RAID systems. This goes to show how wide ranging this topic can be. However, as usual, we will be kicking off with Charly's column. This time it is the turn of DRBD to reduce downtime.

BY CHARLY KÜHNAST

Linux continues to make inroads into the market for "large-scale" server based UNIX systems and this necessitates keeping pace with its current and future contenders with regard to availability. There are several approaches to avoiding downtime, such as clustering or hot and cold standby. But genuine high availability, HA, can only be achieved by ensuring that *every single* component is redundant – that is, by eliminating SPOFs (Single Points of Failure). Hard disk storage is particularly challenge.

A short while ago, I was looking into setting up a highly available hot standby mail server, and opted for two servers with identical hardware configurations for the task in hand. The secondary server is basically idle and only activates if the primary disappears without trace. The heartbeat monitors the connection.

I somehow needed to mirror the primary configuration and user data on the secondary server. Production systems often use a separate NFS server with a shared data pool, but again this would constitute a SPOF and is out of the question.

The solution I came up with was DRBD [1]. The tool provides a block device where I was able to create a filesystem. Write operations to the device are pushed across the wire to the secondary server and executed on the secondary, and this results in a kind of distributed RAID 1.

Decisions

DRBD uses three alternative protocols, referred to as A, B, and C in the documentation.

Protocol A considers a write operation as completed as soon as the information to be written has reached the hard disk on the primary server and has started to cross the wire en route to the secondary. Although this approach offers maximum throughput, it could mean a lot of trouble if things go wrong.

The DRBD instance talk to each other over TCP, and this means that data can be lost if a failover occurs while information is still on its way to the secondary server.

Protocol B resolves this issue. In this case a transaction is regarded as complete when the primary server has written the data and the secondary has confirmed receipt. The protocol guarantees atomic transactions with one possible exception: the secondary server confirms receipt, but both servers die before they can complete the write op. Data loss occurs if the secondary server is promoted to primary when the system comes up again.

If you want to avoid even this residual risk, you can opt for **Protocol C**. In this case a transaction is not regarded as complete until the secondary server has confirmed the physical write operation. The performance should be adequate for two adjacent servers that use a crossover cable for DRBD. However, if the secondary server is on a high latency network, writing large amounts of information might take a lot of patience.

To keep a long story short, I selected Protocol B for my mail server, and am quite happy with that choice. Write performance is about 75 percent of the value achieved without DRBD. I can live with those figures considering the fact that the read performance, which DRBD does not impact at all, is far more important for a mail server. ■

INFO

[1] DRBD: *http://drbd.cubit.at*

THE AUTHOR

Charly Kühnast is a Unix System Manager at the data-center in Moers, near Germany's famous River Rhine. His tasks include ensuring firewall security and availability and taking care of the DMZ (demilitarized zone). Although Charly started out on IBM mainframes, he has been working almost exclusively with Linux since 1995.