High availability and DRBD

One of the most exciting topics in system admininstration that has come up in the last few years is a methodology of doing fairly unintelligent remote data replication via logical devices.

Simple local data replication is usually done with mirroring. RAID-5 allows you to do local data replication more efficiently in terms of space since it yields far more space for the same amount of disk

required. Backups are also remote data replication, but they are not high availability.

Previously, high availability remote data replication was done with some sort of hardware setup configured as some sort of "storage area network." Vendors such as EMC sell very large SAN solutions, usually accessed over a specialized network such as FibreChannel. (Other vendors sell smaller solutions which provide their data service via NFS over IP over Ethernet called "Network Attached Storage", but these typically are using simple local data replication such

RAID-5.)

Other ideas, such as iSCSI and AoE, can give you the ability to transfer data to virtual storage hardware over a network. iSCSI works over long distances by defining a SCSI implementation over TCP. AoE, interestingly enough, is an Ethernet proctocol, like the old LAT, and is not routeable. Each has its own place – you are even seeing hardware being developed to directly support both of these – and offer very cost-effective solutions to problems that have traditionally had expensive solutions.

Another category of solution, DRBD ("Data Replication By DRBD" or "Distributed Replicated Block Device") has come along recently which also takes the idea of using your ordinary network for data replication, but instead of accessing the data via NFS, iSCSI, or AoE, you mount a logical block device that both writes to local storage and writes the same data to another machine. Reads are done locally since the data is there locally and it is certainly faster to read locally.

To be effective, of course, you need applications that are using the block device.

Such applications could be MySQL, Postgresql, NFS, Apache, or most any application that depends on some sort of data storage. This can give inexpensive and simple replication for systems such as MySQL and PostgreSQL, which otherwise can have much more complex replication setups. (PostgreSQL is still working on coming up with a “standard” replication system, with people using several different schemes such as SLONY, PostgreSQL-r, and Bucardo.)

NFS over a DRBD logical device

As an example, an interesting applications for DRBD would be to provide highly available NFS.

Basically, to start out, you need to create simple configuration file (this one is from http://www.linux-ha.org):

```
resource drbd-resource-0 {
  protocol C;                    # fully synchronous mode
  incon-degr-cmd ``halt -f'';    # killall heartbeat
                                 #  would be a good alternative :->
```

```
disk {
  on-io-error panic;
}

syncer {
  rate 100M; # Note: 'M' is MegaBytes, not MegaBits
}

on host-a {
  device    /dev/drbd0;
  disk      /dev/hda8;
  address   192.168.1.10:7789;
  meta-disk  internal;
}

on host-b {
  device    /dev/drbd0;
  disk      /dev/hda8;
  address   192.168.1.20:7789;
```

```
    meta-disk  internal;
  }
}
```

Once you have both machines configured, start DRDB with **drbdadm** and then choose the initial primary.

You can with newer versions of DRBD actually monitor the device the `/proc/drbd` with a simple `cat`.

Create your file system

Now you have your “raw” logical device ready for a file system. You can use **mke2fs -j** to create a ext3 filesystem on that device.

```
mk2efs -j -v /dev/drdb0
```

All of the superblocks and inodes that are you writing will be written to both machines.

NFS

Then mount the new filesystems:

```
mount /dev/drdb0 /mnt/nfs0
mkdir /mnt/nfs0/share0 /mnt/nfs0/share1
```

and add to your `/etc/exports` file lines defining what is to be exported and to whom:

```
/mnt/nfs0/share0    192.168.0.0/255.255.0.0(rw)
/mnt/nfs0/share1    192.168.10.0/255.255.255.0(rw)
```

There is also some variable state for NFS that should be available to both machines in the event of failover. Make a new directory for `/var/lib/nfs` on the new partition at, say, **mkdir /mnt/nfs0/varlibnfs**, and copy the data over. Then remove the old `/var/lib/nfs` and make a soft link from **ln -s /var/lib/nfs /mnt/nfs0/varlibnfs**.

NFS, one at a time

Your basic configuration is finished; however, we need to make sure that NFS is only being served by one machine at a time. This means putting NFS under some sort of controlling system, such as **heartbeat**. Then when either is started, the service should use a virtual ip on its network device; that virtual ip should be available to both machines, which will have to use **ifconfig** to turn that virtual interface on and off when when switches occur. Let's say that we use

192.168.1.100 for that.

Then we need to create a logical name (say, “nfs-server”) for the new NFS cluster should also be added to your DNS that points to this ip number:

```
./add-alias nfs-server.cs.fsu.edu 192.168.
```