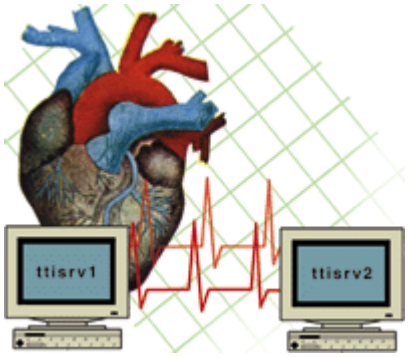
This paper describes a real solution,
implemented by OpenIntegra Ltd.

Case Memo:

Linux High-availability Fault-tolerance Cluster, based on Heartbeat and Distributed Redundant Block Device driver

Yovko Lambrev, <yovko@fsdl.org>
Stoian Mishinev, <mishinev@openintegra.com>



May-June 2004
Sofia - Plovdiv

OpenIntegra Ltd. &
Free Software Development Labs

www.openintegra.com
www.fsdl.org

Introduction

The term "cluster" is actually not very well defined and could mean different things to different people. According to Webopedia, cluster refers to a group of disk sectors. Most Windows users are probably familiar with lost clusters--something that can be rectified by running the defrag utility.

However, at a more advanced level in the computer industry, cluster usually refers to a group of computers connected together so that more computer power, e.g., more MIPS (millions instruction per second), can be achieved or higher availability (HA) can be obtained.

High Availability (HA) Cluster

As more and more critical commercial applications move on the Internet, providing highly available services becomes increasingly important. One of the advantages of a clustered system is that it has hardware and software redundancy. High availability can be provided by detecting node or daemon failures and reconfiguring the system appropriately so that the workload can be taken over by the remaining nodes in the cluster.

Clusters in this category use various technologies to gain an extra level of reliability for a service. Companies such as Red Hat, TurboLinux and PolyServe have cluster products that would allow a group of computers to monitor each other; when a master server (e.g., a web server) goes down, a secondary server will take over the services, similar to "disk mirroring" among servers.

In fact, high availability is a big field. An elegant highly available system may have a reliable group communication sub-system, membership management, concurrent control sub-system and so on. There is a lot of works. However, we can use some existing software to construct highly available Linux systems now.

Simple Theory

Because our client does not have access to more than one real (or public) IP address, we set up our two-node cluster in a private network environment. If you have access to three or more real/public IP addresses, you can certainly set up the Linux cluster with real IP addresses.

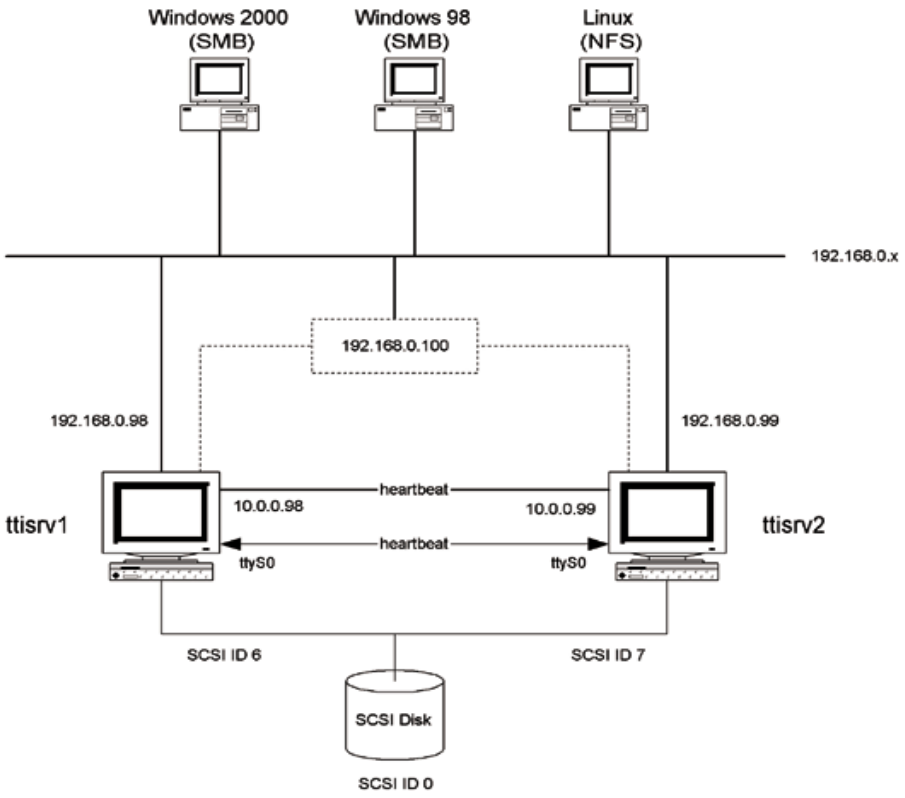
In this network a Cisco router was the gateway to the Internet, and it consists of two IP addresses. The real one was connected to a DSL modem for internet access.

The two-node Linux cluster consists of node1 (192.168.1.1) and

node2 (192.168.1.2) Linux servers. Depending on your setup, either node1 or node2 can be your primary server, and the other will be your backup server. Once the cluster is set, with IP aliasing (read IP aliasing from the Linux Mini HOWTO for more detail), the primary server will be running with an extra IP address (192.168.1.5). As long as the primary server is up and running, services (e.g., DHCP, DNS, HTTP, FTP, etc.) on node1 can be accessed by either 192.168.1.1 or 192.168.1.5. In fact, IP aliasing is the key concept for setting up this two-node Linux cluster.

When node1 (the primary server) goes down, node2 will be take over all services from node1 by starting the same IP alias (192.168.1.5) and all subsequent services. In fact, some services can co-exist between node1 and node2 (e.g., FTP, HTTP, Samba, etc.), however, a service such as DCHP can have only one single running copy on the same physical segment. Likewise, we can never have two identical IP addresses running on two different nodes in the same network.

In fact, the underlining principle of a two-node, high-availability cluster is quite simple, and people with some basic shell programming techniques could probably write a shell script to build the cluster. We can set up an infinite loop within which the backup server (node2) simply keeps pinging the primary server, if the result is unsuccessful, and then start the floating IP (192.168.1.4) as well as the necessary dæmons (programs running at the background).



Hardware and Software Components

Hardware

To get started, you will need two Linux systems with at least one network interface each (preferably two), or an available serial port.

Software

We need **heartbeat** software (http://www.linux-ha.org/download).

Heartbeat is a publicly available package written by Alan Robertson. It provides the basic functions required by any HA system such as starting and stopping resources, monitoring the availability of the systems in the cluster, and transferring ownership of a shared IP address between nodes in the cluster. Heartbeat is a software solution that monitors the health of a particular service (or services) through either a serial line or Ethernet interface or both. It is a vital component of the whole Linux-HA package.

DRBD stands for Distributed Remote Block Device which is produced by LinBit Information Technologies GmbH, provides data mirroring between two servers on a LAN or WAN. (www.linbit.com)

Data replication with DRBD

DRBD is constantly under development, with the last major release 0.7. This release brings numerous enhancements (for example faster resyncs and support for GFS.)

DRBD currently supports one to one replication (i.e. two node clusters, where one node acts as a standby). It is anticipated that DRBD will support one to many replication in the near future.

DRBD can be used to provide data replication between nodes for nearly any application (E.g. Apache, Samba, NFS, Oracle and so on), as well as being used as just an offsite data backup.

DRBD allows for a mirror to be paused, so allowing for backups of a standby node to take place, although a full resynchronisation would be required afterwards.

DRBD is available under the GNU General Public License (GPL), and can be downloaded from drbd.org

DRBD with LifeKeeper

Individual Resource Monitoring ?	Yes
Maximum Cluster Size ?	2 with DRBD. With Shared storage up to 32
Suitable for WAN or LAN environments ?	Yes
Local recovery possible ?	Yes

DRBD with HeartBeat

Individual Resource Monitoring ?	No, 3rd party applications required (e.g. Mon)
Maximum cluster size ?	2 with, or without DRBD
Suitable for WAN or LAN environments ?	Yes
Local recovery possible ?	No, unless using 3rd party software (e.g. Mon)

Configuration files:

/etc/hosts

```
127.0.0.1             node1 localhost.localdomain localhost
10.0.0.1              node1
10.0.0.2              node2
193.68.121.200        cluster
```

/etc/fstab

```
LABEL=/                 /                       ext3    defaults        1 1
/dev/nb0                /mnt/disk               ext3    noauto          0 0
/dev/sda3               swap                    swap    defaults        0 0
```

ha.cf

```
debugfile /var/log/ha-debug
logfile /var/log/ha-log
logfacility    local0
keepalive 2
deadtime 30
warntime 10
initdead 120
udpport 694
#
#       Baud rate for serial ports...
#
#baud   19200
#       serial  serialportname ...
#serial /dev/ttyS0      # Linux
#serial /dev/cuaa0      # FreeBSD
#serial /dev/cua/a      # Solaris
```

```
#       What interfaces to broadcast heartbeats over?
#
bcast   eth0            # Linux

auto_failback on
watchdog /dev/watchdog


node    node1
node    node2
ping 193.68.121.1
respawn hacluster /usr/lib/heartbeat/ipfail
```

drbd.conf

```
resource drbd0 {
  protocol = C
  fsckcmd  = /bin/true


  disk {
    do-panic
    disk-size = 10317828k
  }


  net {
    sync-nice  = -18  # if synchronization is high priority for you
    sync-min    = 500k
    sync-max    = 100M    # maximal average syncer bandwidth
    tl-size     = 5000  # transfer log size, ensures strict write ordering
    timeout     = 60    # unit: 0.1 seconds
    connect-int = 10    # unit: seconds
    ping-int    = 10    # unit: seconds
    ko-count    = 4     # if some block send times out this many times,
                        # the peer is considered dead, even if it still
                        # answeres ping requests
  }


  on node1 {
    device  = /dev/nb0
    disk    = /dev/sda2
    address = 10.0.0.1
    port    = 7788
  }


  on node2 {
    device  = /dev/nb0
    disk    = /dev/sda6
    address = 10.0.0.2
    port    = 7788
  }
}
```

haresources

```
node1  193.68.121.200 datadisk::drbd0 httpd vsftpd
```

authkeys

```
auth 1
1 crc
#2 sha1 HI!
#3 md5 Hello!
```