Linux High-Availability
An overview

Presented by Kevin Inscoe
http://kevininscoe.com/papers/linuxha/

May 15th 2008

Linux Enthusiasts and Professionals
Central Florida (LEAP-CF)

What is High-Availability?

Repair from single point of failure

If a system experiences a failure, it can continue to operate with or without interruption (transition) during the repair process.

IEEE defines High Availability as:

"... availability of resources in a computer system, in the wake of component failures in the system. This can be achieved in a variety of ways, spanning the entire spectrum ranging at the one end from solutions that utilize custom and redundant hardware to ensure availability, to the other end to solutions that provide software solutions using off-the-shelf hardware components. The former class of solutions provide a higher degree of availability, but are significantly more expensive, than the latter class. ... Typically, these products survive single points of failure in the system..."

http://www.ieeetfcc.org/high-availability.html

Wikipedia says:

"High availability is a system design protocol and associated implementation that ensures a certain absolute degree of operational continuity during a given measurement period."

The dreaded nines:

Table 1: Uptime and Maximum Downtime

Uptime	**Uptime**	**Maximum Downtime per Year**
Six nines	99.9999%	31.5 seconds
Five nines	99.999%	5 minutes 35 seconds
Four nines	99.99%	52 minutes 33 seconds
Three nines	99.9%	8 hours 46 minutes
Two nines	99.0%	87 hours 36 minutes
One nine	90.0%	36 days 12 hours

If you really want to throw a spanner in the works, change the definition of availability to: "The percentage of scheduled uptime per year." But let's not go there. We are talking absolutes.

High-Availability is not *Clustering (per se')*

Just because two or more systems are clustered together do not automatically make them highly-available. Clustering may and often is a solution to making an application or system highly-available sometimes but it is **NOT** the complete solution.

Remember you must be able to recover or repair **any** single point of failure this includes networks, front-ends and back-ends (think database).

High-Availability is not Fault-tolerance

Fault tolerance is defined as continuous operation despite multiple failures or interruptions within a given sphere of operation. For instance a system might be consider fault tolerant within a facility but may not consider infrastructure or connectivity failure outside the facility.

High Availability is not Self-Healing.

Self-healing describes any device or system that has the ability to perceive that it is not operating correctly and, without human intervention, make the necessary adjustments to restore itself to normal operation. Because users of a product may find the cost of servicing it too expensive (in some cases, far more than the cost of the product itself), some product developers are trying to build products that fix themselves. Examples of self-healing include Sun's Service Management Facility. Self-healing implies repair procedures are full defined.

High-Availability is not Load balancing or scalability

Although load balancing can improve High-Availability the two concepts are not the completely the same. HA does not imply scalability. Most traditional HA is redundant and not scaled or aggregate.

High Availability is not Parallel processing or check pointing.

Software like Beowulf clusters or Symmetric multiprocessing (SMP) do wonderful at scaling large applications but unless your applications can survive node failures by parallelization alone you still want HA.

So why use High Availability then?

Usually because you have a (legacy) application that cannot horizontally scale, is non-threaded and has no parallelization. Or it cannot be load balanced. In the case of databases cannot replicate or the application cannot take advantage of replication (connection pooling).

Network latency and performance across a network can be significant reason to choose a cluster over parallelization.

Cost. It may be more cost effective to cluster two hosts with attached storage rather than roll out a parallel storage network.

Assumptions:

You are looking at holistic and distro-agnostic solutions. I will not be covering specific HA distros or clusters packages such as Beowulf.org, Rocks or Red Hat Cluster Suite.

In this presentation we will be looking at SPOF as components such that would work in any GNU/Linux distributions.

Infrastructure availability will not be covered.

High-Availability is the repair of a single point of failure (SPOF) what are the failure points and what exists to make them highly available?

Network(s), storage, server hardware (motherboard, disk controllers, nic cards, memory, planar and daughter boards, bus. A whole host of things (pun intended) that go wrong.

Most of the time it is more cost effective to simply duplicate the server to make parts redundant. Many time it simply is not possible to make parts redundant outside of a separate server box or frame. This is what HA is made for.

Node (or server) fail-over.

Sometimes you have to.

Expect time to perform the fail-over! Realistically I always estimate 10-15 minutes for complete roll-over. Maybe longer if performing a node take-over from a hung server. Deadman detection is key.

Note however it is always better to handle faults locally. If you can afford dual NICs and power supplies use them! Most hardware however cannot be faulted.

Good time to remind you to use redundant power legs from different power distribution units. Know your power loading on each circuit.

Faults that we should check for that would trigger a node failure:

Memory, bus errors, power supply failure (if more than one), possibly disk errors is locally attached storage or storage path errors and generally any kind of hardware errors.

Most importantly we must know if our partner is hung this is called dead-man detection AKA Node Fencing. In Linux-HA this is known as Shoot The Other Node In The Head (STONITH). This is accomplished with Heartbeat and multiple networks (serial cable).

You want the kernel to panic on a bad.

/etc/sysctl.conf:

1. kernel.panic_on_oops = 1
2. kernel.panic = 1

Heartbeat

Typical /etc/ha.d/ha.cf looks like this:

```
serial  /dev/ttyS0
auto_failback off (active-active change to on for active-passive)
node hosta
node hostb
keepalive 5
deadtime 15
warntime 10
initdead 30
```

We then specify that "keepalive" heartbeat messages should be sent every 5 seconds, that a host is declared dead when it isn't heard from for 15 seconds, and that a warning is logged if a host takes 10 seconds or more to answer (this might help flag potential problems). We then specify than on first boot, heartbeat sits quietly for at least 30 seconds before deciding who is live and who is dead. This is important as some networks can take a little while before hosts appear properly.

Watchdog Timeout (WDT)

Intel architecture

/etc/ha.d/ha.cf

watchdog /dev/watchdog
Optional. The watchdog function provides a way to have a system that is still minimally functioning, but not providing a heartbeat, reboot itself after a minute of being sick. This could help to avoid a scenario where the machine recovers its heartbeat after being pronounced dead. If that happened and a disk mount failed over, you could have two nodes mounting a disk simultaneously. If you wish to use this feature, then in addition to this line, you will need to load the "softdog" kernel module and create the actual device file. To do this, first type "insmod softdog" to load the module. Then, type "grep misc /proc/devices" and note the number it reports (should be 10).
Next, type "cat /proc/misc | grep watchdog" and note that number (should be 130). Now you can create the device file with that info typing, "mknod /dev/watchdog c 10 130".

/kernel/Documentation/nmi_watchdog.txt

In order to use the NMI watchdog, you need to have APIC (Advanced Programmable Interrupt Controller) support in your kernel.

For SMP kernels, APIC support gets compiled in automatically. For UP, enable either CONFIG_X86_UP_APIC (Processor type and features -> Local APIC support on uniprocessors) or CONFIG_X86_UP_IOAPIC (Processor type and features -> IO-APIC support on uniprocessors) in your kernel config. CONFIG_X86_UP_APIC is for uniprocessor machines without an IO-APIC. CONFIG_X86_UP_IOAPIC is for uniprocessor with an IO-APIC. [Note: certain kernel debugging options, such as Kernel Stack Meter or Kernel Tracer, may implicitly disable the NMI watchdog.]

Other deadman and hardware failure detection methods:

LM Sensors - http://www.lm-sensors.org/
SMARTmon - http://smartmontools.sourceforge.net/
Intelligent Platform Management Interface (IPMI) -
    /kernel/Documentation/IPMI.txt
    http://openipmi.sourceforge.net/
    http://www.samag.com/documents/s=9559/sam0503e/
Watch Dog boards (old legacy hardware) – Berkprod.com

Forcing reboot on hung:

rICMP
Magic SysReq (if you are at the keyboard or have access to a KVM)
Intregated Lights Out (iLO) – HP/Compaq servers
Integrated Lights Out Management (ILOM) – Sun and others
Sun Advanced Lights Out Manager (ALOM) – Sun Sparc servers
Remote power off – Advocent and intelligent power strips
Control Alt Del – Hey this ain't windows!

Things to consider when performing a node fail-over:

Can your data survive such an event for **any** reason?

Is your data check-pointed, journaled, 3-Phase Commit, etc..

Are you using replication:
Distributed Replicated Block Device (DRBD), MySQL replication, Hadoop, Network Area Storage (NAS), NFS or cellular file systems like AFS? Resilient file systems like ZFS.

Session management. Are you web sessions session safe across servers (session storage)? Customer does not like losing their shopping cart during a node fail-over.

Multi-step jobs. If they begin running on another host are they check pointed?

Message buses and IPC do they have backing store?

System time should always be in sync on all hosts. Ntp, clockspeed, etc

Logging of events: syslog should be going off cluster.

Monitoring: How do you know an event has occurred?

Backups: How are backups performed when the node has failed-over?

Think how will I start operating again automatically and seamlessly if this server fails and then TEST, TEST and TEST SOME MORE!

I walk behind servers regularly and pull plugs to see what happens. Hint: I already know what will happen. :-)

Clustering:

Active-Active (Load balancing)
Passive-Active (Redundancy)

Cold (off), Hot (active) and Warm (passive) hosts in Linux-HA.

Resource management:

Linux-HA requires resources be defined:

V1: haresoruces file
V2: Cluster Resource Manager (CRM)

http://www.linux-ha.org/ClusterResourceManager

It can get complicated quick.
KISS – Keep It Simple Silly

Cluster needs to have a consistent view of which nodes are valid members.

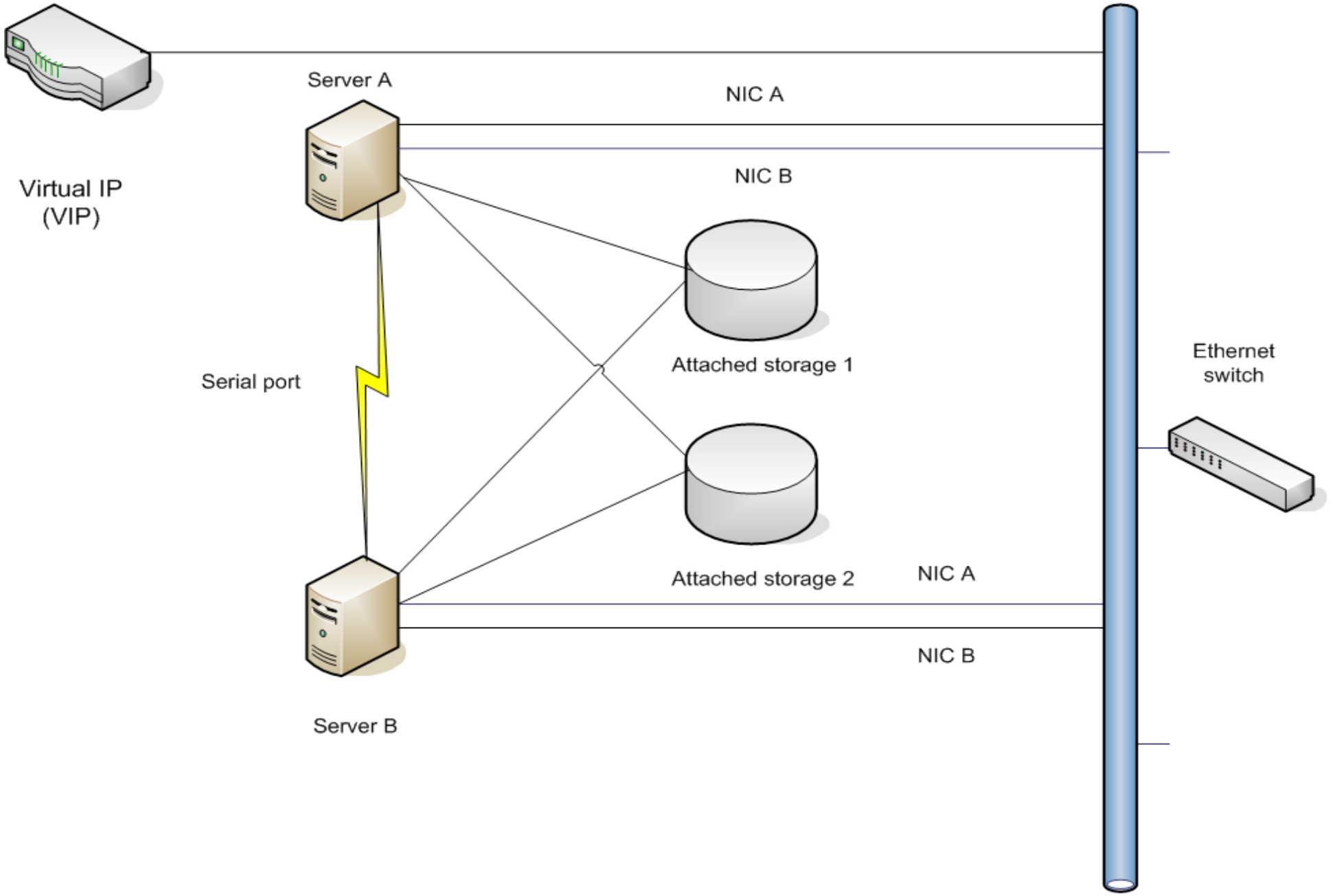
Lack of heartbeats received will lead to assumption of a node failure.

Difficult to distinguish between link failures, node failures if all links are cut – leads to Splitbrain.

Splitbrain is the term when both sides of the cluster have become confused and can no longer work as a cluster. This is incredibly bad because depending on setting of auto_fallback both may try to take over. This is why using serial cable is best (but make sure it is good!)

Because failures can only be detected asynchronously, the cluster state is never certain.

Typical two node HA configuration



Network adapters

Media: Ethernet and fiber. For sake of time dial-up, token ring, wireless and other non-prolific network media will not be considered here.

Two schools of thought: Heartbeat or Keep-alive.

Linux High Availability Project and Heartbeat

http://www.linux-ha.org/

Virtual Router Redundancy Protocol (VRRP) and Keepalived

http://www.keepalived.org/
http://sourceforge.net/projects/vrrpd/

Heartbeat requires cooperation through a reliable secondary network. This can be either another Ethernet network or some other kind of media but most frequently is a serial port network since serial connections do not frequently break.

Keep-alive requires the assistance of external gear to maintain connectivity to a single host. A keep-alive signal is often sent at predefined intervals, and plays an important role on the Internet. After a signal is sent, if no reply is received the link is assumed to be down and future data will be routed via another path until the link is up again. Keep-alive packet contains null data. VRRP provides information on the state of a router, not the routes processed and exchanged by that router. Each VRRP instance is limited, in scope, to a single subnet. VRRP is not routable.

VRRP works well with Linux Virtual Servers (LVS) (http://www.linux-vs.org/).

VRRP provides gateway redundancy by allowing each router within the redundant router topology to share a virtual Ethernet MAC address and a virtual IP address. When the virtual addresses are active on a particular router, the router is said to be the master. Routers without control of the virtual addresses are referred to as backups. VRRPd is an implementation of Virtual Router Redundancy Protocol as specified in rfc2338 (obsoleted by rfc3768).

Which one is better? It depends on where you want your HA fail over decisions to be made. VRRP requires adapter configurations and router and switch support for it but little else needs to be done to the host for network fail-over. You also do not require a second network. On the other hand if the infrastructure does not support VRRP you should use Heartbeat.

Essentially VRRP is router based fail-over and Heartbeat is host based fail-over.

Heartbeat requirements:

- Serial cable and dedicated serial ports on each host.
- Heartbeat software http://www.linux-ha.org/Heartbeat
- If you want IP fail-over in a single box you will need the Linux Channel Bonding Driver (IpFailoverChannelBonding) and dual NIC's. (http://www.linux-ha.org/IpFailoverChannelBonding)

VRRP requirements:

rfc3768 capable router and switch gear (Nokia, Cisco and Juniper all on board with latest firmware).

Keepalived (http://www.keepalived.org/)

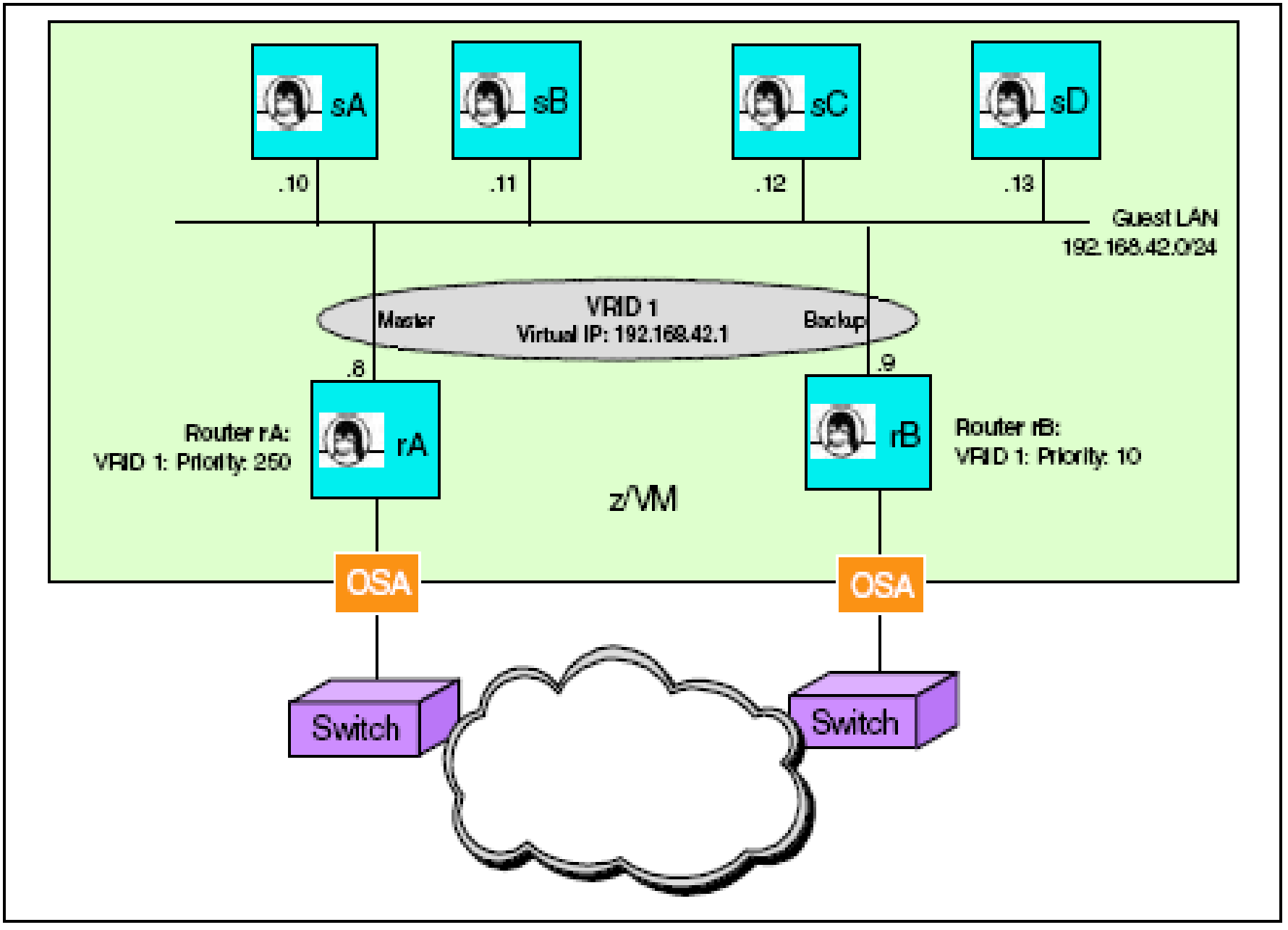


Figure 4 Linux routing with basic VRRP

In the configuration of the Linux guests, we specify the VIP as the default gateway. Now, VRRP will provide a continuous router service across the two routers.

Storage

Lots of choices here but they fall into three general categories:

Local, Network and Adapter

Physical layers:

Local:

IDE, EIDE, ATA, SATA, PATA, SCSI, USB, etc...

Fail-over methods include mirroring, replication, multipathing (Device Mapper) and twin-tailing in some SCSI scenarios. Twin-tail must be able to “vary on and off” the LUN called SCSI reservations - fence_scsi(8)

Peripheral sharing between two co-located hosts might be possible with USB.

Disconnect failed nodes from access to storage is called Fencing.

Improve hardware redundancy with RAID and multipathing. Some disk controllers can perform RAID internally.

Software redundancy: JFS, LVM (mirror) and replication.

Network:

NFS, NAS (NetApp) – Simplest to implement, performance can be a problem. Linux-HA has HaNFS.

AFS, OpenAFS, Arla – More difficult to set up. Requires user authentication with Kerberos and time must be in sync.

ISCSI – Native to host but still can be a performance issue

Global file systems - Lustre - Lustre is a scalable, secure, robust, highly-available cluster file system. It is designed, developed and maintained by Sun Microsystems, Inc. (lustre.org)
GFS - http://www.redhat.com/gfs/
IBM GPFS

Scalable file systems – Hadoop (HDFS)

“ Hadoop implements MapReduce, using the Hadoop Distributed File System (HDFS) (see figure below.) MapReduce divides applications into many small blocks of work. HDFS creates multiple replicas of data blocks for reliability, placing them on compute nodes around the cluster. MapReduce can then process the data where it is located.

Hadoop has been demonstrated on clusters with 2000 nodes. The current design target is 10,000 node clusters.”

Supposedly unbreakable file systems – Sun ZFS (not yet natively available for Linux)

Replication – rsync (not real time)
- DBRD (real time and good recovery)

Adapters:

Host Bus Adapters, Serial Channel (SSA) and Fiber Channel

Unique identifiers by volser and/or World Wide Name (WWN) and World Wide Identifier (WWID).

Dual cards managed by LVM and/or Device Mapper (DM) [identified by WWID] or mounted as SCSI emulation (danger is device order of SCSI bus if it changes).

DM is recommended especially if using multiple HBAs.

LVM is only needed for partitioning LUN's

Application fail-over:

OpenAIS API – Built in to Linux-HA but programs must interface.

Open Clustering Framework Resource Agent API – opencf.org
Linux-HA has OCF Resource Agents that use these.

Linux Virtual Services, Linux-Director, Ldirectord and Ultra Monkey.

http://www.ultramonkey.org/3/ldirectord.html

Replicated High Availability Manager - http://www.linuxha.net/

Some applications have good clustering support or scalability built in: MySQL.

Web servers put content switches or load balancers ahead of them like Arrowpoint (now Cisco CSS), CoyotePoint or F5. Make use of Internet Cache Protocol (ICP).

Commercial cluster products: HP ServiceGuard, EMC AutoStart, UpSuite HA, Veritas Cluster Server (VCS), Oracle TAF and Netra High Availability (HA) Suite.

The most common method of start/stopping applications is via init.d (LSB Resource Agents with OCF Resource Agent) and shell scripts.

Init scripts need to be Linux Standard Base Core Specification 3.0 compatible

http://www.linux-ha.org/LSBResourceAgent

Need to be extra cognizant of exit status and making sure processes die and start like you expect. Also the order of jobs and processes may be important. Jobs steps may have dependencies.

Failure detection:

Linux-HA: linux-ha.org

Other HA software:

OSCAR - http://xcr.cenit.latech.edu/ha-oscar/

HACMP now available for Linux

Questions?

Kevin Inscoe - http://kevininscoe.com
email: kevin@inscoe.org