The Red Hat Cluster Suite NFS Cookbook

Setting up a Load-Balanced NFS Cluster with Failover Capabilities

By Bob Peterson
rpeterso@redhat.com
Updated: 15 July 2010

---

15 July 2010: Changed the document to reflect the maximum of sixteen nodes supported by Red Hat.

23 March 2007: I added a section on clustering basics, a few notes about how RHEL5 is different and additional notes about two-node clusters and quorum disks.

Special thanks to Riaan van Niekerk (riaan@obsidian.co.za) for his suggestions and improvements.

NFS file serving lets you share valuable data with potentially hundreds of client computers on your company's network, but there are pitfalls. Suppose, for example, that your company has purchased fifty disk-less workstations for your employees. The fact that the workstations have no hard drives saved the company big money on purchasing, and the reliability is better because there are fewer moving parts to fail. Sounds good in theory, right? But there are two big problems: First, how do you distribute the NFS workload so that all the clients aren't hammering on the same server? Second problem: what if a server fails? How do you prevent it from locking up dozens of workstations and paralyzing your business?

The solution to the first problem is to distribute the work throughout several NFS servers. This is known as load balancing. Rather than having one NFS server for fifty clients, you can have five NFS servers, each of which provide data to ten clients, thus distributing the workload.

The solution to the second problem is to configure your NFS servers with Red Hat Cluster Suite NFS failover service. Basically, all five servers have a stethoscope on each of the other four servers, listening for a heartbeat. If a server fails, one of the remaining servers automatically takes on its duties seamlessly and the client won't even know about the failure. This is known as failover and it is accomplished by the Red Hat Cluster Suite's (RHCS's) Resource Group Manager, rgmanager.

This document will cover the basics of setting up and running the Red Hat Cluster Suite and NFS so that your network is somewhat load-balanced and provides failover service. It assumes the reader has a basic understanding of NFS. I'm going to use lots of specific examples that will be pretty basic. In other words, I'm going to use baby-steps. I'm not going to expound on all the details and options available to you, but I will explain what I'm doing along the way. To keep things simple, I'm going to stick with my five-server example, although it should easily scale to sixteen servers.

The goal here is to show you how to do it by giving examples, rather than providing you with a cryptic manual and forcing you to figure it out yourself. If you're looking for a detailed manual on setting up RHCS, it may be found here:

http://www.redhat.com/docs/manuals/csgfs/browse/rh-cs-en/

Step 1 – Set your Goals and Design your Cluster

Take some time to carefully design your cluster before you invest time or money implementing it. Design the network so that it provides enough redundancy to guarantee no interruption of service. Make sure you buy hardware that provides scalability to allow your cluster to grow as your company's needs grow. Work from your goals. Your goals should dictate how complex your configuration gets.

Decide how many nodes to use

The first question you should ask is how big to make your cluster. In other words, how many computers ("nodes") to have in your cluster. There are two main issues to consider: fault tolerance and workload.

The first issue, fault tolerance, relates to how much you can tolerate downtime. At the heart of the issue is a special cluster requirement called quorum. Quorum is a rule that protects your data. Let me explain with an example.

Suppose you have two network switches tied together with a single Ethernet cable and each switch has half of your nodes. If someone accidentally unplugs that cable, you have a big problem: instead of one big cluster, you suddenly have two smaller clusters. If these nodes have access to the same data on shared storage, each of these two rogue clusters, without knowledge of the other, would quickly corrupt the file system, unless they have some rule to prevent it. To make sure that never happens, we only allow writing to the file system if there is a majority of nodes communicating. Any rogue clusters that are cut off from the majority refuse to operate because they don't have a majority of votes; they don't have quorum. In other words, at least (half + 1) of your nodes need to be communicating normally.

Since a majority of nodes is needed for normal operation, if you have too few nodes in the cluster, you're more likely to have down time. If your company rarely gets orders through the Internet, a three node cluster will continue serving your data if one node fails, but if two out of three go down, your cluster is down, and that may be fine for your business.

On the other hand, if every minute of downtime costs your company a million dollars, you may want a cluster of sixteen-nodes so that your only down time would be in the rare event that eight or more nodes fail. Right now, Red Hat only supports up to sixteen nodes in a cluster, although the software can probably handle more.

By the way, we made a special exception for two-node clusters so they can continue running if one node fails, but that has to be specially configured. There are also ways you can specify a tie-breaker for two-node clusters.

The second issue to consider when choosing the size of your cluster is workload. Workload relates to how much work is required of the cluster. If you have more nodes in your cluster, you can distribute the workload between the additional servers and handle a heavier workload. For example, if you're using your cluster for web serving, you probably want enough nodes to ensure that your web servers don't get swamped by lots of Internet traffic when your company is "discovered"; customers don't have much patience.

Load Balancing Issues

You should also think about load balancing issues. If you're planning to serve NFS, ask yourself why you want load-balanced NFS capabilities in the first place. If it's only to serve data to web servers, that might be more efficiently done by leaving NFS out of the picture altogether. Since the rgmanager service can manage web servers as well as NFS servers, it may be better to forget NFS and use a managed apache/httpd service on your cluster.

Next, determine if you need true load-balancing for your situation. With true load-balancing, all requests are distributed equally between the servers, and all servers do the same amount of work at any given time. Requests are distributed to the servers by a request “traffic-cop” known as a Linux Virtual Server (LVS) router. This is known as NAT routing. The router must stand between your client machines and the servers and this is known as a tier. Multiple tiers may be used reduce the workload on the LVS router, and multiple LVS routers may be introduced for failover, but each tier adds a layer of complexity. For more information, see the section on **True Load Balancing** below.

If you don't care about occasional server congestion, you can use a simple technique I call round-robin pseudo load-balancing. With pseudo load-balancing, clients are split up equally between the servers, and any client's individual requests are all handled by the same server. Individual servers may still occasionally get swamped with requests since the requests themselves are not load-balanced. For simple NFS applications, this might be acceptable.

For the sake of simplicity, I'll use pseudo load-balancing in this document.

Scalability

Design your cluster with scalability in mind. Setup your IP addresses and Virtual IP addresses so there is room to add more.

Because of the way I've chosen my IP addresses, this example allows you to add new servers to the cluster as desired. I've named my servers trin-10 through trin-14 and assigned them IP addresses 192.168.0.10 through 14. You can easily add new servers, starting with trin-15 and assign them IPs starting with 192.168.0.15 and have them working in the cluster with minimal effort.

For this document, the goal is only to provide basic NFS service to several client computers with pseudo load-balancing and failover. I'll assume:

- Fifty disk-less NFS clients.
- Five NFS servers (called trin-10, 11, 12, 13, and 14) running Red Hat Enterprise Linux 4 (RHEL4) with static IP addresses 192.168.0.10 through 192.168.0.14.
- Basic NFS file serving through five virtual IP addresses 10.0.0.1 through 10.0.0.5.
- Pseudo load-balancing. Each of the five servers will be assigned one of the virtual IPs. DHCP service can be set up on one of the servers to dynamically assign client IPs and distribute them equally between the five Virtual IPs. Virtual IP 10.0.0.1 will service clients 1, 6, 11, 16, 21 and so on. Address 10.0.0.2 will service clients 2, 7, 12, 17, 22, etc.

  We'll manage our Virtual IPs with round robin DNS, which ordinarily may have some disadvantages. In this case, it becomes more palatable because the Virtual IPs will be managed and made highly available by Cluster Suite, so even if a single node goes down, its

Virtual IP will be relocated to another node, and in theory, the VIPs should be always available.
- For failover, the Virtual IP of the failing server may be moved to any of the others. We can also set up failover for the DHCP service.

Step 2 – Design and Set up the Cluster Hardware

The first step is to design and set up your cluster of servers. For that you're going to need some hardware. For my example, I'm using a configuration that consists of:

- Five NFS servers, each of which has an IDE boot drive (hda), at least one Network Interface Card (NIC) providing device eth0 and a Fibre Channel Host Bus Adapter: QLogic QLA2310.
- A fibre-channel switch.
- Hard disk storage consisting of three LUNs (which look like SCSI drives sda, sdb, sdc) connected to the fibre switch.
- Network switch or hub. In my example, any good high speed Ethernet switch will do.
- A Network Power Switch. In my example, I'll use an APC MasterSwitch Plus.

Network Connections

Each server is attached to a network switch with an Ethernet cable. The Ethernet connections will be assigned static IPs which are shown in blue on the diagram. Later, we will assign virtual IP addresses to them as well, shown in red.

The client connections are shown in red to indicate they're getting their NFS data from the Virtual IP, even though they do have a physical connection. The diagram shows them connected directly to the network switch, but in real life the network would probably be more complex.

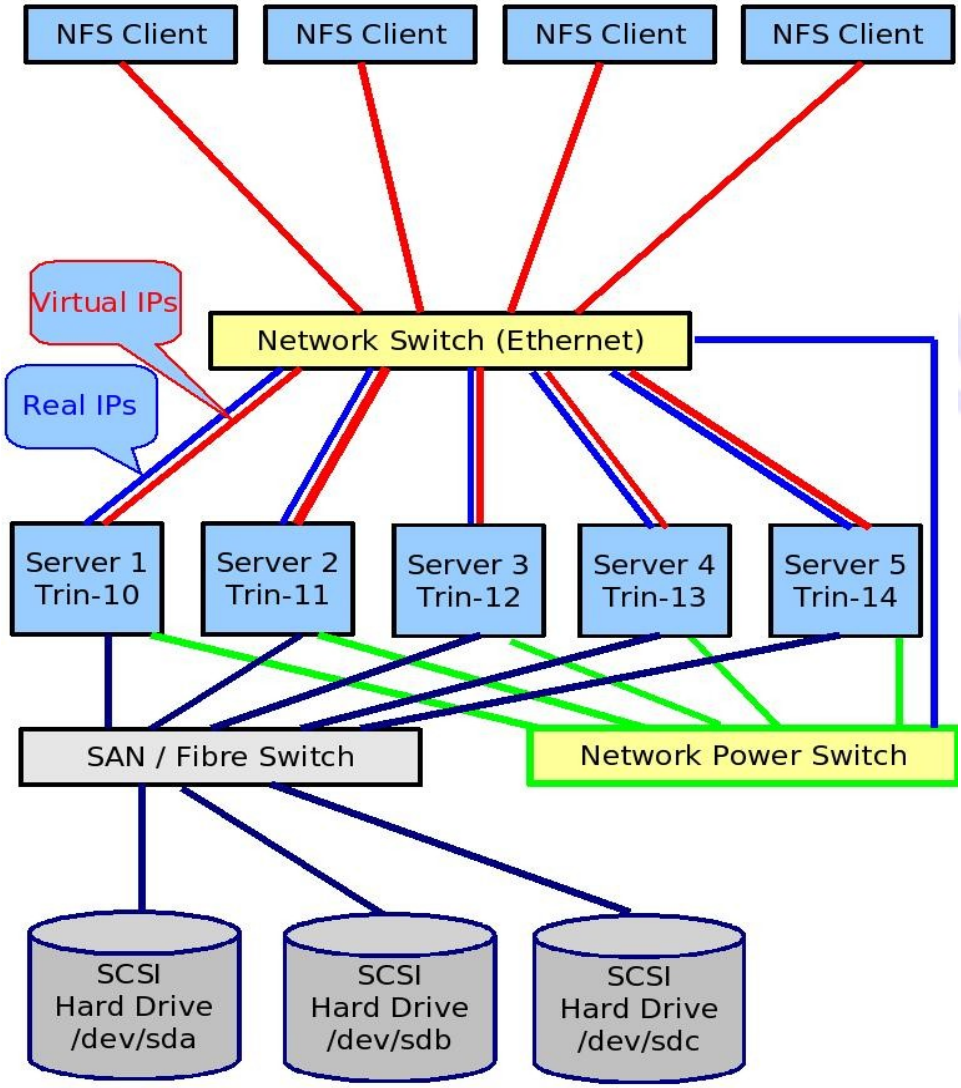
Power Connections

All servers are plugged into a Network Power Switch (NPS) that also has an Ethernet connection to the network switch. Power cables are shown in green. The NPS serves an important role: if a node fails, the other nodes must be able to power-cycle (turn off and back on) the failed one to prevent it from potentially damaging the shared storage.

Storage Connections

Each server has access to the same physical storage through the Fibre Channel switch. The hard drives and servers plug into the switch with fiber-optic cables (shown in dark blue).

Here is a diagram of the hardware:



Step 3 – Set up Red Hat Cluster Suite and Red Hat Global File System

In order to set up rgmanager failover service, you must first set up your servers in a clustered environment. For that, we install Red Hat Cluster Suite (RHCS) 4.0. and Red Hat Global File System (GFS) 6.1. Here are the basics:

Step 3a. Apply the necessary RPMs on all five servers

Your NFS servers should have the latest Red Hat Cluster Suite, Device Mapper, LVM2 and Linux kernel RPMs installed. These RPMs should be added to all NFS servers to be clustered (called nodes). Here is an example:

[root@trin-10 ~]# cd /usr/src/redhat/RPMS/i686/

[root@trin-10 i686]# rpm -i kernel-2.6.9-42.EL.i686.rpm

Repeat this step for all five servers. After the new kernel is installed, reboot your servers to pick up the new kernel. Next, we apply the RPMs for device-mapper, lvm2, and the Red Hat Cluster Suite.

[root@trin-10 i686]# rpm -i device-mapper-1.02.07-4.0.RHEL4.i686.rpm
[root@trin-10 i686]# rpm -i lvm2-2.02.06-6.0.RHEL4
[root@trin-10 i686]# rpm -i ccs-1.0.7-0.i686.rpm cman-1.0.11-0.i686.rpm cman-kernel-smp-2.6.9-45.2.i686.rpm dlm-1.0.1-1.i686.rpm dlm-kernel-smp-2.6.9-42.10.i686.rpm fence-1.32.25-1.i686.rpm GFS-6.1.6-1.i686.rpm GFS-kernel-smp-2.6.9-58.0.i686.rpm gulm-1.0.7-0.i686.rpm iddev-2.0.0-3.i686.rpm lvm2-cluster-2.02.06-7.0.RHEL4.i686.rpm magma-1.0.6-0.i686.rpm magma-plugins-1.0.9-0.i686.rpm perl-Net-Telnet-3.03-3.noarch.rpm rgmanager-1.9.53-0.i686.rpm system-config-cluster-1.0.27-1.0.noarch.rpm

Repeat this step for all five servers. This is just an example; the versions for these RPMs are a moving target, so take the latest for your release.

NOTE: For RHEL5 and equivalent, you will need a different set of RPMs. For example, there is no gulm, or magma, ccs, fence and cman are all combined into one “cman,” and dlm is built into the kernel. For RHEL5, you'll need openais.

Step 3b. Create your /etc/cluster/cluster.conf configuration file.

If you're running RHEL5 or RHEL4.5, you're in luck because there's a new web-based cluster setup and management gui called “conga”. It will make your life a whole lot easier than before.

If you're not using conga, there is a graphical tool for first-time setup of a cluster called **cs-deploy-tool** that automates most of the setup. There's another graphical tool called **system-config-cluster** that allows you to configure your cluster even after it's been configured. To get to it from the GUI, follow this path:

Applications -> System Settings -> Server Settings -> Cluster Management.

If you want to configure your cluster manually, edit /etc/cluster/cluster.conf with your editor of choice. Here is the file I'll be using for my example:

```
<?xml version="1.0"?>
<cluster config_version="1" name="bob_cluster">
      <cman/>
      <clusternodes>
            <clusternode name="trin-10">
                  <fence>
                        <method name="single">
                              <device name="trin-apc" nodename="trin-10"/>
                        </method>
```

```
            </fence>
        </clusternode>
        <clusternode name="trin-11">
            <fence>
                <method name="single">
                    <device name="trin-apc" nodename="trin-11"/>
                </method>
            </fence>
        </clusternode>
        <clusternode name="trin-12">
            <fence>
                <method name="single">
                    <device name="trin-apc" nodename="trin-12"/>
                </method>
            </fence>
        </clusternode>
        <clusternode name="trin-13">
            <fence>
                <method name="single">
                    <device name="trin-apc" nodename="trin-13"/>
                </method>
            </fence>
        </clusternode>
        <clusternode name="trin-14">
            <fence>
                <method name="single">
                    <device name="trin-apc" nodename="trin-14"/>
                </method>
            </fence>
        </clusternode>
    </clusternodes>
    <fencedevices>
        <fencedevice agent="fence_apc" ipaddr="192.168.0.9" login="apc" name="trin-apc" passwd="apc"/>
    </fencedevices>
    <fence_daemon post_fail_delay="0" post_join_delay="3"/>
</cluster>
```

If you're only planning to have two nodes, you'll need to do some more. Two-node clusters are a special case that require some forethought. A bigger cluster will only run as long as it maintains quorum, which means that there has to be a majority of nodes working in the cluster. So for a seven-node cluster, there has to be at least four nodes or more running. With a two-node cluster, there is no majority, so you have to decide between two possible solutions: The first solution is to “bend” the quorum rules by adding expected_votes="1" two_node="1" to the <cman/> statement. The second solution is to use a “quorum disk” as a tie-breaker.

The first option, bending the rules, has a negative side-effect. If the two nodes lose contact with each other, for whatever reason, they will both realize it at the same time and each will try to fence the other. This can result in a shootout in which the fastest node kills the other to protect the data, and depending on the kind of fencing you use, they may both end up dying.

The second option is to use a tie breaker. If a node starts experiencing communication failures, there's no way to know whether the problem is with itself or with the other node, unless it has

some other shared resource like a a special partition on shared storage or a second communication channel. This is typically called a quorum disk, but that's a misnomer; you can set up a wide variety of heuristics to determine which node will break the tie. Unfortunately, setting up a quorum disk is not trivial; your best resource is to read the qdisk man page.

Step 3c. Send the cluster.conf to all nodes in the cluster.

Use some tool like scp to move the cluster.conf file to all of the servers. Once they have their initial cluster.conf file, changes to the file are normally propagated by using the **ccs_tool update** command, but that won't work until we have the cluster up and running. So for now, we'll do it by hand:

[root@trin-10 ~]# scp /etc/cluster/cluster.conf root@trin-11:/etc/cluster/
[root@trin-10 ~]# scp /etc/cluster/cluster.conf root@trin-12:/etc/cluster/
[root@trin-10 ~]# scp /etc/cluster/cluster.conf root@trin-13:/etc/cluster/
[root@trin-10 ~]# scp /etc/cluster/cluster.conf root@trin-14:/etc/cluster/

Step 4. Bring up the cluster

The next step is to get your servers thinking as a team. In other words, you need to bring up the cluster. The first time, we'll do this manually, but we'll also configure the systems to do this automatically on startup.

You can find some cool tools out on the Internet that allow you to type the same command simultaneously on all of your nodes. I use one called cssh (cluster-ssh). It's probably not the best and it's not part of the regular installation, so for now I'll assume you'll just use regular ssh and a for loop.

Step 4a. Tell all servers to start the ccsd service.

This is the service that shares cluster.conf information in the cluster. Not only do we start the service, we also set it up to start automatically every time the systems are rebooted.

[root@trin-10 ~]# service ccsd start
[root@trin-10 ~]# for i in `seq 11 14` ; do ssh trin-$i “service ccsd start” ; done

Step 4b. Tell all servers to join in the cluster.

Now we use the cluster manager tool to get the servers talking to each other. As before, we start the service and configure it to start automatically at startup.

[root@trin-10 ~]# service cman start
[root@trin-10 ~]# for i in `seq 11 14` ; do ssh trin-$i “service cman start” ; done

Now your systems are not just servers, they are nodes in a cluster.

FYI: The cman service essentially executes this command: cman_tool join -w

Step 4c. Tell all nodes to start the "fenced" service.

This is the service that detects node failures and power cycles servers that have failed (an act known as fencing). This is necessary to prevent damage to the filesystem.

[root@trin-10 ~]# service fenced start
[root@trin-10 ~]# for i in `seq 11 14` ; do ssh trin-$i "service fenced start" ; done

The fenced service essentially executes this command: fence_tool join -w

NOTE: for RHEL5 and equivalent, these three services are all combined into one called "cman." So for RHEL5, you don't need steps 4a or 4c.

Step 4d. Tell all nodes to start the clvmd service.

This is the cluster LVM (Logical Volume Manager) daemon. Its job is to distribute LVM metadata information between cluster nodes.

[root@trin-10 ~]# service clvmd start
[root@trin-10 ~]# for i in `seq 11 14` ; do ssh trin-$i "service clvmd start" ; done

Step 4e. Start the GFS service.

This enables cluster locking and the GFS filesystem within the Linux kernel. The necessary kernel modules will be loaded automatically at startup by the gfs service if you have a GFS filesystem to be mounted in your /etc/inittab.

[root@trin-10 ~]# service gfs start
[root@trin-10 ~]# for i in `seq 11 14` ; do ssh trin-$i "service gfs start" ; done

Step 5. Make sure your servers can see your shared storage

From one of the nodes, you need to prepare your volume group, physical volume, logical volume, then format your GFS filesystem.

Step 5a. Make sure all nodes have the host bus adapter device drivers loaded.

The first step is to make sure your fibre channel host bus adapter device driver is loaded. For my example, I'm using a QLogic QLA2310 host bus adapter, so I'll need the qla2300 and qla23xx device drivers.

```
[root@trin-10 ~]# lsmod | grep qla
qla2300               124481  0
qla2xxx               179297  8 qla2300
scsi_transport_fc       7873  1 qla2xxx
scsi_mod              121293  3 qla2xxx,scsi_transport_fc,sd_mod
```

If you don't see the device drivers, you may need to use “depmod -a” to resolve module dependencies for your kernel and modprobe to load them.

Step 5b. Make sure all nodes can properly see the physical drives.

Now that our device drivers are loaded, we need to find out if they can actually see the hard drives that are attached. The best way is by asking the Linux kernel. If the kernel can see the SCSI drives, they should show up as sdX partitions in /proc/partitions:

```
[root@trin-10 ~]# cat /proc/partitions
major minor  #blocks  name
   3     0   39082680 hda
   3     1     104391 hda1
   3     2   38973690 hda2
   8     0    8388608 sda
   8    16    8388608 sdb
   8    32    8388608 sdc
   8    48    8388608 sdd
```

I threw in a fourth SCSI device, sdd, just to make a point: make sure you know which hard drives are which. You don't want to accidentally destroy someone else's data by repartitioning a hard drive you assumed was yours. You can use the “pvs” and “pvdisplay” commands to see what physical volumes exist and whether they are assigned to volume groups.

If your SAN isn't in the list, you may want to use the “dmesg” command to see if there are any complaints from the Linux kernel. If there aren't any complaints, you may need to configure your SAN, which is sometimes done through a web interface.

Step 6. Create your GFS filesystem

Step 6a. Update the lvm configuration files on the servers.

If you switched from normal lvm to the clustered version, you may need to change your lvm configuration a bit.

```
[root@trin-10 gfs]# vi /etc/lvm/lvm.conf
```

Locate the “global” section and change locking_type and locking_library as follows:

locking_type = 2
locking_library = "liblvm2clusterlock.so"

NOTE: For RHEL5, you should use locking_type = 3.

The changes need to be propagated to all the servers.

[root@trin-10 gfs]# vgchange -aly
[root@trin-10 gfs]# vgscan
[root@trin-10 gfs]# for i in `seq 11 15` ; do scp /etc/lvm/lvm.conf root@trin-$i:/etc/lvm/ ; done

Step 6b. Prepare the physical volumes for use.

This takes the place of using fdisk to partition the drives.

[root@trin-10 ~]# pvcreate /dev/sda
[root@trin-10 ~]# pvcreate /dev/sdb
[root@trin-10 ~]# pvcreate /dev/sdc

Step 6c. Create a volume group for your GFS filesystem.

This groups all three physical volumes into a volume group.

[root@trin-10 ~]# vgcreate bobs_vg /dev/sda dev/sdb dev/sdc

Step 6d. Create a logical volume out of your volume group.

[root@trin-10 ~]# lvcreate -L 750G bobs_vg

Step 6e. Make your GFS filesystem within the logical volume.

[root@trin-10 ~]# gfs_mkfs -t bob_cluster:bobs_gfs -p lock_dlm -j 8 /dev/bobs_vg/lvol0

Here I specified eight (8) journals for the filesystem. There must be at least one journal per server in the cluster. By specifying eight, I'm leaving myself room to add more servers in the future.

Step 6f. Edit /etc/fstab so that it mounts the GFS filesystem to be exported.

Using your editor of choice, edit the /etc/fstab file to automatically mount the new GFS filesystem at system startup. In my fstab, I've got an entry that looks like this:

/dev/bob_vg/lvol0 /mnt/bob gfs defaults 0 0

Step 6g. Mount the new filesystem and propagate the fstab to the other nodes.

[root@trin-10 ~]# mount -a
[root@trin-10 ~]# for i in `seq 11 15` ; do scp /etc/fstab root@trin-$i:/etc/ ; done
[root@trin-10 ~]# for i in `seq 11 15` ; do ssh trin-$i “mount -a” ; done

Step 7 – Choose and Configure NFS Failover Service

There are two basic types of NFS failover: managed NFS service and Managed Virtual IPs with RHCS's Global File System, GFS. You need to decide which you're going to use and there are advantages and disadvantages to both. In either scenario, NFS clients continue to read and write files on the NFS-mounted path without even detecting when a failure occurs.

With managed NFS service, rgmanager manages the entire NFS service, automatically monitoring the service and relocating it to other nodes when node failures are detected.

With Managed Virtual IPs, NFS service is running on all of the nodes and rgmanager monitors a virtual IP address. When node failures are detected, the Virtual IP address is moved from node to node. This method is currently designed to work only in conjunction with GFS.

Which should you choose? If you need to export file-systems other than GFS, for example, EXT3, on your shared storage, then you should use Managed NFS Service. If you need to maintain your own /etc/exports file and export a mixture of filesystems where some are on shared storage and some are on local storage, use Managed Virtual IPs.

If you're only going to be exporting GFS on shared storage you can use either method. However, Managed NFS Service has two advantages: First, it's easier to use since rgmanager takes care of the exports for you, so you don't need to manage the /etc/exports file. Second, if for some reason the NFS service stops on one of your servers—which admittedly isn't likely—the rgmanager will detect and handle the problem. With Managed Virtual IPs, if the NFS service stops, you may never know it unless you check it periodically.

Managed NFS Service

This method of managing NFS failover is more powerful than using Managed Virtual IPs because it may be used to manage any file-system NFS can export—provided the storage is shared among the nodes in the cluster—whereas Virtual IPs should only be used with GFS. It is relatively easy to set up because you can use the system-config-cluster GUI tool to manager your resources. It is also easier to maintain because it automates the monitoring, starting and stopping of the NFS service on the servers. However, it is not as flexible as using Managed Virtual IPs because rgmanager manages all of your NFS shares through the cluster configuration file (cluster.conf) rather than letting you pick and choose with the /etc/exports file.

There are two ways to set up Managed NFS Service with rgmanager: manually or with the system-config-cluster GUI tool.

Setting up Managed NFS Service using system-config-cluster:

Step 7a. Create our failover domains for the servers

1. Start the system-config-cluster tool. From the Start menu:
   Applications -> System Settings -> Server Settings -> Cluster Management
2. Under "**Managed Resources**" highlight "**Failover Domains**."
3. Click the "**Create a Failover Domain**" button.
4. Next, you want to build a list of nodes that may run the service. To do that, use the "**Available Cluster Nodes**" pulldown to add nodes to the "member node" box.
5. If you want the service to be prioritized, check the "**Prioritized List**" check-box. You may then highlight the nodes listed in the "member nodes" box and click the "**Adjust Priority**" buttons to select the order in which the failover should occur. The node with the highest priority (priority 1) will be the initial node for running the service. If that node fails, the service will be relocated to the node with the next higher priority (priority 2) and so forth. If multiple nodes have the same priority, rgmanager will choose among them as it sees fit. Hit the **Close** button when you are done.

Create five failover domains, one for each server. For each domain, set the priority to "1" for the "home" server, and "2" for the other servers. That way, each server will be responsible for a group of clients, and if that server fails, the service will be relocated to another server.

Step 7b. Create our resources

Next, for each server, we create our resources, the NFS exports.

6. Under "Managed Resources" Highlight "**Resources**."
7. Click the "**Create a Resource**" button.
8. Select resource type "**GFS**" and fill in Name, Mount Point, Device and Options
9. Click the "**Create a Resource**" button.
10. Select "**NFS Export**" and fill in the name.
11. Click the "**Create a Resource**" button.
12. Select "**NFS Client**" and fill in the name, target, read-write or read-only and options.

Step 7c. Set up the NFS services

Now we have to set up the NFS services to be run on each of the servers.

13. Under "Managed Resources" highlight "**Services**."
14. Click the "**Create a Service**" button and give the service a name. For example, "nfssvc." A "Service Management" pop-up window will appear.
15. Use the "**Failover Domain**" pulldown to select the domain you specified from step 4.
16. Under "Recovery Policy" select "**Restart**" or "**Relocate**."
17. Click the button labeled "**Create a new resource for this service**."

18. Use pull-down to select Resource Type “GFS.”
19. Enter Name (“bobfs1”), Mount Point (“/mnt/bob”), Device (“/dev/bob_vg/lvol0”), and Options (“acl”).
20. Put in all of your exports this way.

Step 7d. Save your changes and propagate to the cluster

21. When you are finished, save your changes with **File->Save** or by pressing **<Ctrl+S>**
22. Click the “**Send To Cluster**” button to send your new configuration to the nodes in the cluster.
23. You may then click on the “**Cluster Management**” tab and start the service.

When you're done, your /etc/cluster/cluster.conf file should be similar to the one shown in the next section.

Setting up Managed NFS Service manually:

Step 7a. Add resource manager domains, resources and services to cluster.conf

Edit the /etc/cluster/cluster.conf file using your editor of choice and add a <rm> section that defines what rgmanager should manage and how. For this example, we add the following lines to the cluster.conf near the bottom, before the <fence_daemon> statement:

```
<rm>
        <failoverdomains>
                <failoverdomain name="nfsdomain1" ordered="1" restricted="1">
                        <failoverdomainnode name="trin-10" priority="1"/>
                        <failoverdomainnode name="trin-11" priority="2"/>
                        <failoverdomainnode name="trin-12" priority="2"/>
                        <failoverdomainnode name="trin-13" priority="2"/>
                        <failoverdomainnode name="trin-14" priority="2"/>
                </failoverdomain>
                <failoverdomain name="nfsdomain2" ordered="1" restricted="1">
                        <failoverdomainnode name="trin-10" priority="2"/>
                        <failoverdomainnode name="trin-11" priority="1"/>
                        <failoverdomainnode name="trin-12" priority="2"/>
                        <failoverdomainnode name="trin-13" priority="2"/>
                        <failoverdomainnode name="trin-14" priority="2"/>
                </failoverdomain>
                <failoverdomain name="nfsdomain3" ordered="1" restricted="1">
                        <failoverdomainnode name="trin-10" priority="2"/>
                        <failoverdomainnode name="trin-11" priority="2"/>
                        <failoverdomainnode name="trin-12" priority="1"/>
                        <failoverdomainnode name="trin-13" priority="2"/>
                        <failoverdomainnode name="trin-14" priority="2"/>
                </failoverdomain>
                <failoverdomain name="nfsdomain4" ordered="1" restricted="1">
                        <failoverdomainnode name="trin-10" priority="2"/>
                        <failoverdomainnode name="trin-11" priority="2"/>
```

```
            <failoverdomainnode name="trin-12" priority="2"/>
            <failoverdomainnode name="trin-13" priority="1"/>
            <failoverdomainnode name="trin-14" priority="2"/>
        </failoverdomain>
        <failoverdomain name="nfsdomain5" ordered="1" restricted="1">
            <failoverdomainnode name="trin-10" priority="2"/>
            <failoverdomainnode name="trin-11" priority="2"/>
            <failoverdomainnode name="trin-12" priority="2"/>
            <failoverdomainnode name="trin-13" priority="2"/>
            <failoverdomainnode name="trin-14" priority="1"/>
        </failoverdomain>
    </failoverdomains>
    <resources>
        <clusterfs device="/dev/bob_vg/lvol0" force_unmount="0" fstype="gfs"
                                                mountpoint="/mnt/bob" name="bobfs1” options="acl"/>
        <nfsexport name="NFSexports1"/>
        <nfsclient name="client-1" options="rw" target="client-1.mycompany.com"/>
        <clusterfs device="/dev/bob_vg/lvol0" force_unmount="0" fstype="gfs"
                                                mountpoint="/mnt/bob" name="bobfs2" options="acl"/>
        <nfsexport name="NFSexports2"/>
        <nfsclient name="client-2" options="rw" target="client-2.mycompany.com"/>
        <clusterfs device="/dev/bob_vg/lvol0" force_unmount="0" fstype="gfs"
                                                mountpoint="/mnt/bob" name="bobfs3" options="acl"/>
        <nfsexport name="NFSexports3"/>
        <nfsclient name="client-3" options="rw" target="client-3.mycompany.com"/>
        <clusterfs device="/dev/bob_vg/lvol0" force_unmount="0" fstype="gfs"
                                                mountpoint="/mnt/bob" name="bobfs4" options="acl"/>
        <nfsexport name="NFSexports4"/>
        <nfsclient name="client-4" options="rw" target="client-4.mycompany.com"/>
        <clusterfs device="/dev/bob_vg/lvol0" force_unmount="0" fstype="gfs"
                                                mountpoint="/mnt/bob" name="bobfs5" options="acl"/>
        <nfsexport name="NFSexports5"/>
        <nfsclient name="client-5" options="rw" target="client-5.mycompany.com"/>
    </resources>
    <service autostart="1" domain="nfsdomain1" name="nfssvc1">
        <ip address="10.0.0.1" monitor_link="1"/>
        <clusterfs ref="bobfs1">
            <nfsexport name="bobfs1">
                <nfsclient ref="client-1"/>
            </nfsexport>
        </clusterfs>
    </service>
    <service autostart="1" domain="nfsdomain2" name="nfssvc2">
        <ip address="10.0.0.2" monitor_link="1"/>
        <clusterfs ref="bobfs2">
            <nfsexport name="bobfs2">
                <nfsclient ref="client-2"/>
            </nfsexport>
        </clusterfs>
    </service>
    <service autostart="1" domain="nfsdomain3" name="nfssvc3">
        <ip address="10.0.0.3" monitor_link="1"/>
        <clusterfs ref="bobfs3">
```

```
                <nfsexport name="bobfs3">
                    <nfsclient ref="client-3"/>
                </nfsexport>
            </clusterfs>
        </service>
        <service autostart="1" domain="nfsdomain4" name="nfssvc4">
            <ip address="10.0.0.4" monitor_link="1"/>
            <clusterfs ref="bobfs4">
                <nfsexport name="bobfs4">
                    <nfsclient ref="client-4"/>
                </nfsexport>
            </clusterfs>
        </service>
        <service autostart="1" domain="nfsdomain5" name="nfssvc5">
            <ip address="10.0.0.5" monitor_link="1"/>
            <clusterfs ref="bobfs5">
                <nfsexport name="bobfs5">
                    <nfsclient ref="client-5"/>
                </nfsexport>
            </clusterfs>
        </service>
    </rm>
```

This example essentially defines the managed NFS service.

Notice that the priorities for each failover domain are distributed between the servers. I'm setting up pseudo load-balancing by making each failover domain an ordered (prioritized) list and using the priority setting to distribute the workload between the servers.

The shared filesystem defined as path /mnt/bob is exported through NFS to clients named “client-1” through “client-50” although they're not all shown here. The preferred node for running the NFS service “nfssvc1” is “trin-10” since trin-10's priority is set higher than the other nodes in the failover domain. If trin-10 fails, the rgmanager of the other nodes will detect the failure and relocate the NFS service to one of the other nodes depending on circumstances. Likewise, the preferred node for the second NFS export is trin-11.

Step 7b. Copy the cluster.conf file to all nodes in the cluster:

When the cluster was down, we had to use scp to copy the cluster.conf file to the servers. Now that the cluster is running, we should do it the right way. This is accomplished with the use of two tools, ccs_tool and cman_tool as shown below.

First, propagate your changed cluster.conf file to the cluster:

```
[root@trin-10 ~]# ccs_tool update /etc/cluster/cluster.conf
```

Next, determine the in-memory configuration version number.

```
[root@trin-10 ~]# cman_tool status | grep "Config version"
Config version: 1
```

Update the version number to your newer version. Since the old version was 1, we tell it to use version 2.

[root@trin-10 ~]# cman_tool version -r 2

Step 7c. Delete the /etc/exports file

If you want rgmanager to manage your NFS exports, rgmanager should manage them all. In other words, you shouldn't have other NFS exports in /etc/exports or NFS will get confused.

If you need a /etc/exports file to manage NFS exports that are not part of the clustered filesystem, you should read further and use Managed Virtual IP Service instead.

Otherwise you can skip the next section and proceed to step 8.

Managed Virtual IP Service

This method of managing NFS failover is more flexible than using Managed NFS Service because it allows you to hand-craft your NFS exports as you see fit. However, it is only designed to work on GFS file systems. Using it for non-GFS file systems is not recommended.

In this configuration, you must ensure that /etc/exports file is in sync on all nodes, and that nfsd is always running on all nodes (it's not monitored by the cluster; the service doesn't use <nfsexport> in this case), and that the GFS file system is mounted before NFS is started. Therefore, it requires more planning, maintenance and monitoring.

Step 7a. Add resource manager domains, resources and services to cluster.conf

The cluster configuration is much simpler than before. The failover domains are the same, but the resources and services are simpler. Using your favorite editor, change the “rm” section of the /etc/cluster/cluster.conf file to look something like this:

```
<rm>
    <failoverdomains>
        <failoverdomain name="igridnodes1" ordered="1" restricted="1">
            <failoverdomainnode name="trin-10" priority="1"/>
            <failoverdomainnode name="trin-11" priority="2"/>
            <failoverdomainnode name="trin-12" priority="2"/>
            <failoverdomainnode name="trin-13" priority="2"/>
            <failoverdomainnode name="trin-14" priority="2"/>
        </failoverdomain>
        <failoverdomain name="igridnodes2" ordered="1" restricted="1">
            <failoverdomainnode name="trin-10" priority="2"/>
            <failoverdomainnode name="trin-11" priority="1"/>
            <failoverdomainnode name="trin-12" priority="2"/>
            <failoverdomainnode name="trin-13" priority="2"/>
            <failoverdomainnode name="trin-14" priority="2"/>
        </failoverdomain>
        <failoverdomain name="igridnodes3" ordered="1" restricted="1">
```

```
                <failoverdomainnode name="trin-10" priority="2"/>
                <failoverdomainnode name="trin-11" priority="2"/>
                <failoverdomainnode name="trin-12" priority="1"/>
                <failoverdomainnode name="trin-13" priority="2"/>
                <failoverdomainnode name="trin-14" priority="2"/>
            </failoverdomain>
            <failoverdomain name="igridnodes4" ordered="1" restricted="1">
                <failoverdomainnode name="trin-10" priority="2"/>
                <failoverdomainnode name="trin-11" priority="2"/>
                <failoverdomainnode name="trin-12" priority="2"/>
                <failoverdomainnode name="trin-13" priority="1"/>
                <failoverdomainnode name="trin-14" priority="2"/>
            </failoverdomain>
            <failoverdomain name="igridnodes5" ordered="1" restricted="1">
                <failoverdomainnode name="trin-10" priority="2"/>
                <failoverdomainnode name="trin-11" priority="2"/>
                <failoverdomainnode name="trin-12" priority="2"/>
                <failoverdomainnode name="trin-13" priority="2"/>
                <failoverdomainnode name="trin-14" priority="1"/>
            </failoverdomain>
        </failoverdomains>
        <resources>
            <ip address="10.0.0.1" monitor_link="1"/>
            <ip address="10.0.0.2" monitor_link="1"/>
            <ip address="10.0.0.3" monitor_link="1"/>
            <ip address="10.0.0.4" monitor_link="1"/>
            <ip address="10.0.0.5" monitor_link="1"/>
        </resources>
        <service autostart="1" domain="igridnodes1" name="10.0.0.1">
            <ip ref="10.0.0.1"/>
        </service>
        <service autostart="1" domain="igridnodes2" name="10.0.0.2">
            <ip ref="10.0.0.2"/>
        </service>
        <service autostart="1" domain="igridnodes3" name="10.0.0.3">
            <ip ref="10.0.0.3"/>
        </service>
        <service autostart="1" domain="igridnodes4" name="10.0.0.4">
            <ip ref="10.0.0.4"/>
        </service>
        <service autostart="1" domain="igridnodes5" name="10.0.0.5">
            <ip ref="10.0.0.5"/>
        </service>
    </rm>
```

Again, notice that the priorities for each failover domain are distributed between the servers. I'm setting up pseudo load-balancing by making each failover domain an ordered (prioritized) list and using the priority setting to distribute the workload between the servers. The first Virtual IP service is set to run on the first server, trin-10, at startup because its priority is set higher (1) than the priorities of the other servers (2). The second Virtual IP service is set to run on the second server, trin-11, and so forth.

Step 7b. Concoct your NFS exports file

First, you need to decide your NFS exports and put them in your /etc/exports file. For example:

/mnt/bob/bin *(ro,insecure,no_root_squash)
/mnt/bob/sbin *(ro,insecure,no_root_squash)
/mnt/bob/usr/bin *(ro,insecure,no_root_squash)

Carefully choose your NFS export options with your company's security in mind. Exporting to everyone (*) is generally not a good idea. Likewise, options like "insecure" and "no_root_squash" might be okay if you're exporting directories like /bin or /usr/bin, but it's definitely not a good idea for valuable company data. Remember, this is just an example.

Step 7c. Next, send your /etc/exports file to all servers in the cluster.

Since the NFS exports file is not managed by the cluster software in this configuration, you need to ensure that it's the same for all of your NFS servers.

[root@trin-10 ~]# for i in `seq 11 14` ; do scp /etc/exports root@trin-$i:/etc/ ; done

Step 7d. Start your NFS service and set it up to start at boot time on all servers:

[root@trin-10 ~]# service nfs start
[root@trin-10 ~]# for i in `seq 11 14` ; do ssh trin-$i "service nfs start" ; done

Step 8 – Start the Resource Manager Service

Next, we need to start the Resource Manager service, rgmanager, on all servers. Like before, we start the service and set it up to start at boot time.

[root@trin-10 ~]# service rgmanager start
[root@trin-10 ~]# for i in `seq 11 14` ; do ssh trin-$i "service rgmanager start" ; done

If we've done our jobs correctly, the rgmanager should start the NFS service or Virtual IP service we set up earlier on the appropriate servers. If it doesn't work for some reason, we can look in /var/log/messages to find out why.

Step 9 – Set up DHCP

It may be out of the scope of this document to set up DHCP, but somehow you need to get your fifty NFS clients to use the five virtual IP addresses in a pseudo load-balanced way. One way you

can do that is to configure DHCP on one of the servers to assign IP addresses to your disk-less NFS clients and export their root directories.

For example, you can set up root partitions for each of your clients on your GFS filesystem. You might have fifty entries in /etc/dhcpd.conf that look something like this:

```
host client1 {
        option root-path "/mnt/bob/client1/";
        filename "/mnt/bob/bootImage";
        hardware ethernet 00:60:18:3F:00:01;
        fixed-address 10.0.1.1;
}
host client2 {
        option root-path "/mnt/bob/client2/";
        filename "/mnt/bob/bootImage";
        hardware ethernet 00:60:18:3F:00:02;
        fixed-address 10.0.1.2;
}
(and so forth for every client)
```

When the clients are booted, they request their IP address from the DHCP server. Once assigned, they can use some mechanism like tftp to load their kernel and NFS to mount their own root directories Read/Write.

For static directories that can be read-only, all of your clients could access the same physical directories through NFS and distribute the workload throughout the five servers by using the five Virtual IPs. For example, /bin, /usr/bin, and /usr/lib can all be NFS mounts. In client1's fstab (file /mnt/bob/client1/etc/fstab on the servers) we have something like:

```
/dev/nfs              /             nfs     defaults       0 0
none                 /proc          proc    defaults       0 0
10.0.0.1:/mnt/bob/bin.i686    /bin      nfs     defaults,ro     0 0
10.0.0.1:/mnt/bob/usrbin.i686    /usr/bin      nfs     defaults,ro     0 0
10.0.0.1:/mnt/bob/usrlib.i686    /usr/lib      nfs     defaults,ro     0 0
```

The fstab file would be the same for clients 6, 11, 16, 21, 26, and so forth, so every fifth client would be using the same Virtual IP and therefore, the same physical server.

The fstab for client2 (/mnt/bob/client2/etc/fstab) is similar, but accomplishes pseudo load-balancing by using a different Virtual IP which accesses the same data from a different server. Same for clients 17, 22, 27, 32, etc. We have:

```
/dev/nfs              /             nfs     defaults       0 0
none                 /proc          proc    defaults       0 0
10.0.0.2:/mnt/bob/bin.i686    /bin      nfs     defaults,ro     0 0
10.0.0.2:/mnt/bob/usrbin.i686    /usr/bin      nfs     defaults,ro     0 0
10.0.0.2:/mnt/bob/usrlib.i686    /usr/lib      nfs     defaults,ro     0 0
```

And of course, clients 3, 8, 13, 18, 23, etc., would mount from 10.0.0.3. Clients 4, 9, 14, 19, 24 would all mount from 10.0.0.4 and 5, 10, 15, 20, 25 would all mount from 10.0.0.5. All fifty of the clients would have similar configurations to accomplish the pseudo load-balancing.

I could have also specified clients 1 through 10 to use server1, 11 through 20 to use server

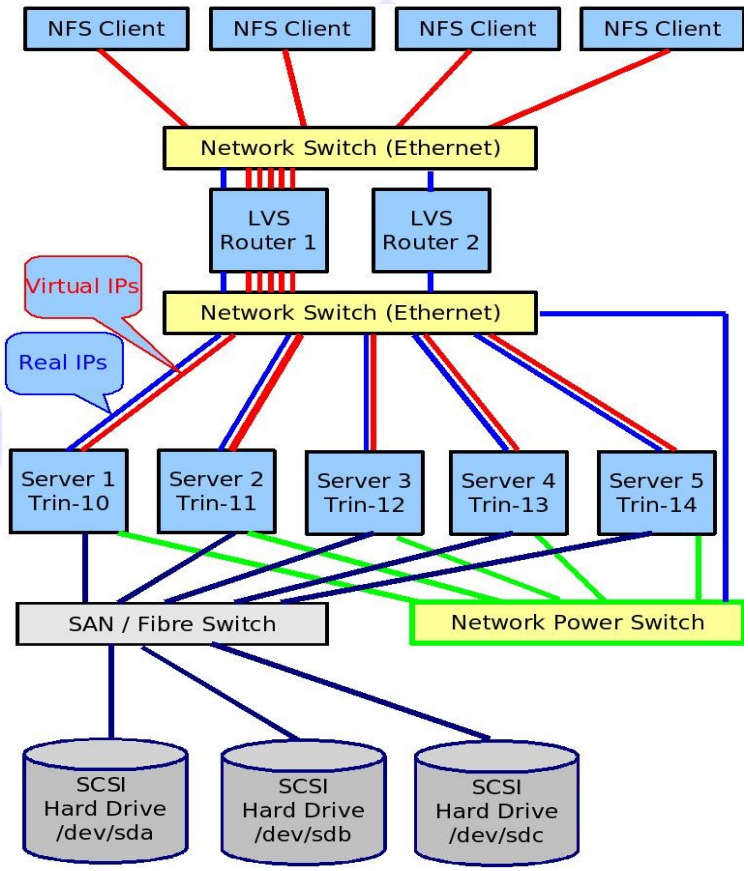
2, and so forth. I purposely chose to stagger them instead, and that goes back to the design of the network. If you assign clients 1 through 10 to one department and 11 through 20 to another department (thereby making it easier to find hardware problems down the road), our server workload may become unbalanced due to the demands of the different departments. You don't want one department to hammer a server hard while another department's server does nothing but sit and eat bonbons. It's better for all departments to tax the servers equally. Which brings us to our next topic.

True Load-Balancing

What we've done so far is not true load balancing. We distributed the clients equally across the NFS servers, but some of your servers may still be pounded by requests while others sit idle. If your fifty workstations are divided among different departments, you might have one department hammering one of the servers on the weekend (for example, production) while another server is completely idle (for example, because the shipping department is closed for the weekend.)

If you really need to distribute the workload equally among the servers, you need true load-balancing, and that adds another layer of complexity. The best way to achieve that is by using Linux Virtual Server (LVS). LVS uses a mechanism called Network Address Translation (NAT) to route requests from one IP address to another and that's what achieves true load-balancing.

With LVS, you have a second set of servers (called LVS routers) whose job is to equally distribute the requests. Only one router is active at a time; additional routers are needed to provide failover capabilities in case the first router fails. Our example diagram might look like this:



With LVS, all requests come in to a central LVS server known as the Active Router. The router decides which server to give the request to, based on one of several selectable methods. A second router (called the backup router) monitors the network and takes over if the active router fails.

Red Hat Cluster Suite comes with a graphical LVS configuration tool called Piranha. It has a web interface and requires php to run.

If you really want to get fancy, you can add a second tier of servers, one tier doing the NFS file serving and the second tier serving that NFS data to the web. Of course, it gets harder to configure and maintain these networks with each level of complexity.

Minimizing Downtime on Mission-Critical Servers

I've shown you how Red Hat Cluster Suite can provide failover capabilities in case a server fails, but that's not the end of the story. If you look at the diagram of our cluster above, there are still several points of failure. If your NFS servers are mission-critical, you may want to do everything in your power to make sure there is no downtime.

There are several steps you can take to ensure your systems stay up barring a city-wide power failure. It's out of the scope of this document to show you how to set those up, but I'd be remiss if I didn't at least mention them. Most of them are hardware related.

- **Redundant Power Supplies** – Instead of having one power supply in a server, you have two. This provides redundant power if one power supply fails.
- **Uninterruptible Power Supply (UPS)** – Provides temporary continuity of power if the electricity goes out.
- **Multipathing** – The basic idea here is to have two SAN / Fibre switches with redundant cables. That allows the data on the hard drives to be read even if one of the switches has a problem.
- **RAID, Mirroring, etc.** – The data on the hard drives is copied (mirroring) or arranged on the disks to allow for data recovery in case a hard drive fails. RAID stands for Redundant Array of Independent Disks.
- **Network Redundancy** – The idea here is to have more than one network switch or hub and redundant NIC connections. That way, if a switch or hub fails, the cluster can still reach the network through the other switch or hub. Also, if a NIC or a network cable fails in a server, there is still a path from that server to the network through a second NIC and cable.
- **Site Redundancy** – The idea here is to have two different locations for your servers, maybe even in different cities. That way, even if the entire site is taken down by some disaster or loss of power, you'll have another site to back them up. This is sometimes known as Site Mirroring.

Where to go for help

At first this can all seem overwhelming, but once you get into it, it's not that hard. Fortunately, there are places where you can go for help and to get questions answered. You can:

- Read the Cluster Project Page:
  http://sources.redhat.com/cluster/wiki/

- Search the Cluster Suite Frequently Asked Questions:
  http://sources.redhat.com/cluster/wiki/FAQ
- Subscribe to the Linux Cluster Mailing List and ask there. For more information visit:
  https://www.redhat.com/mailman/listinfo/linux-cluster