THE EXPERT'S VOICE® IN LINUX

Pro Linux High Availability Clustering

Sander van Vugt

Apress®

For your convenience Apress has placed some of the front matter material after the index. Please use the Bookmarks and Contents at a Glance links to access them.

Apress®

Contents at a Glance

Introduction

This book is about high availability (HA) clustering on Linux, a subject that can be overwhelming to administrators who are new to the subject. Although much documentation is already available on the subject, I felt a need to write this book anyway. The most important reason is that I feel there is a lack of integral documentation that focuses on tasks that have to be accomplished by cluster administrators. With this book, I have tried to provide insight into accomplishing all of the tasks that a cluster administrator typically has to deal with.

This means that I'm not only focusing on the clustering software itself but also on setting up the network for redundancy and configuring storage for use in a clustered environment. In an attempt to make this book as useful as possible, I have also included three chapters with use cases, at the end of this book.

When working with HA on Linux, administrators will encounter different challenges. One of these is that even if the core components Corosync and Pacemaker are used on nearly all recent Linux distributions, there are many subtle differences.

Instead of using the same solutions, the two most important enterprise Linux distributions that are offering commercially supported HA also want to guarantee a maximum of compatibility with their previous solutions, to make the transition for their customers as easy as possible, and that is revealed by slight differences. For example, Red Hat uses fencing and SUSE uses STONITH, and even if both do the same thing, they are doing it in a slightly different way. For a cluster administrator, it is important to be acutely aware of these differences, because they may cause many practical problems, most of which I have tried to describe in this book.

It has, however, never been my intention to summarize all solutions. I wanted to write a practical field guide that helps people build real clusters. The difference between these two approaches is that it has never been my intention to provide a complete overview of all available options, commands, resource types, and so on. There is already excellent documentation doing this available on the Web. In this book, I have made choices with the purpose of making cluster configuration as easy as possible for cluster administrators.

An important choice is my preference for the `crm` shell as a configuration interface. This shell is the default management environment on SUSE Linux clusters and is not included in the Red Hat repositories. It is, however, relatively easy to install this shell by adding one additional repository, and, therefore, I felt no need to cover everything I'm doing in this book from both the `crm` shell as well as the `pcmk` shell. This would only make the book twice as long and the price twice at high, without serving a specific purpose.

I hope this book meets your expectations. I have tried to make it as thorough as possible, but I'm always open to feedback. Based on the feedback provided, I will make updates available through my web site: `www.sandervanvugt.com`. If you have purchased this book, I recommend checking my web site, to see if errata and additions are available. If you encounter anything in this book that requires further explanation, I would much appreciate receiving your comments. Please address these to `mail@sandervanvugt.nl`. and I will share them with the readership of this book.

I am dedicated to providing you, the reader, with the best possible information, but in a dynamic environment such as Linux clustering, things may change, and different approaches may become available. Please share your feedback with me, and I will do my best to provide all the readers of this book with the most accurate and up-to-date information!

—Sander van Vugt

CHAPTER 1

High Availability Clustering and Its Architecture

In this chapter, you'll learn how high availability (HA) clustering relates to other types of clustering. You'll also read about some typical use cases for HA clustering. After a discussion on the general concepts of HA clustering, you'll read about its different components and implementations on Linux.

Different Kinds of Clustering

Roughly speaking, three different kinds of cluster can be distinguished, and all of these three types can be installed on Linux servers.

- *High performance*: Different computers work together to host one or more tasks that require lots of computing resources.
- *Load balancing*: A load balancer serves as a front end and receives requests from end users. The load balancer distributes the request to different servers.
- *High availability*: Different servers work together to make sure that the downtime of critical resources is reduced to a minimum.

High Performance Clusters

A high performance cluster is used in environments that have heavy computing needs. Think of large rendering jobs or complicated scientific calculations that are too big to be handled by one single server. In such a situation, the work can be handled by multiple servers, to make sure it is handled smoothly and in a timely manner.

An approach to high performance clustering is the use of a Single System Image (SSI). Using that approach, multiple machines are treated by the cluster as one, and the cluster just allocates and claims the resources where they are available (Figure 1-1). High performance clustering is used in specific environments, and it is not as widespread as high availability clustering.

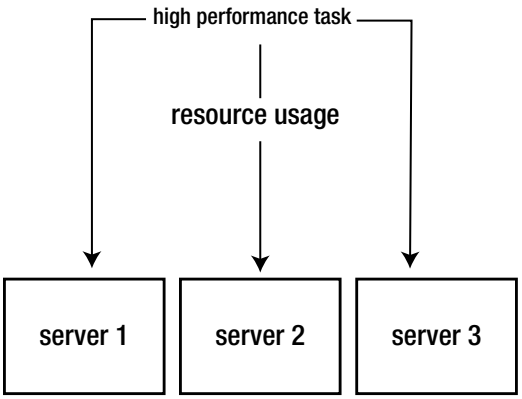

Figure 1-1. *Overview of high performance clustering*

Load Balancing Clusters

Load balancing clusters are typically used in heavy-demand environments, such as very popular web sites. The purpose of a load balancing cluster is to redistribute a task to a server that has resources to handle the task. That seems a bit like high performance clustering, but the difference is that in high performance clusters, typically, all servers are working on the same task, where load balancing clusters take care of load distribution, to get an optimal efficiency in task-handling.

A load balancing cluster consists of two entities: the load balancer and the server farm behind it. The load balancer receives requests from end users and redistributes them to one of the servers that is available in the server farm (Figure 1-2). On Linux, the Linux Virtual Server (LVS) project implements load balancing clusters. HAProxy is another Linux-based load balancer. The load balancers also monitor the availability of servers in the server farm, to decide where resources can be placed. It is also very common to use hardware for load balancing clusters. Vendors like Cisco make hardware devices that are optimized to handle the load as fast and efficiently as possible.

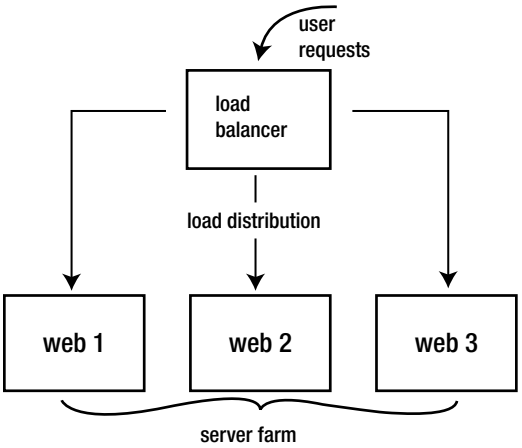

Figure 1-2. *Overview of load balancing clusters*

High Availability Clusters

The goal of a high availability cluster is to make sure that critical resources reach the maximum possible availability. This goal is accomplished by installing cluster software on multiple servers (Figure 1-3). This software monitors the availability of the cluster nodes, and it monitors the availability of the services that are managed by the cluster (in this book, these services are referred to as *resources*). If a server goes down, or if the resource stops, the HA cluster will notice and make sure that the resource is restarted somewhere else in the cluster, so that it can be used again after a minimal interruption. This book is exclusively about HA clusters.

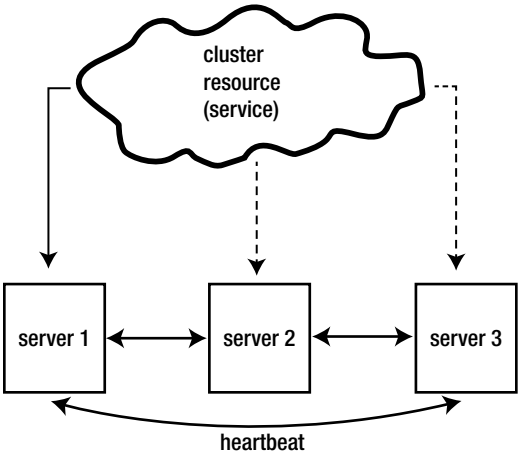

Figure 1-3. *Overview of high availability clusters*

What to Expect from High Availability Clusters

Before starting your own high availability cluster project, it is good to have the appropriate expectations. The most important is to realize that an HA cluster maximizes availability of resources. It cannot ensure that resources are available without interruption. A high availability cluster will act on a detected failure of the resource or the node that is currently hosting the resource. The cluster can be configured to make the resource available as soon as possible, but there will always be some interruption of services.

The topic of this book is HA clustering as it can be used on different Linux distributions. The functionality is often confused with HA functionality, as it is offered by virtualization solutions such as VMware vSphere. It is good to understand what the differences and similarities between these two are.

In VMware vSphere HA, the goal is to make sure that virtual machines are protected against hardware failure. vSphere monitors whether a host or a virtual machine running on a host is still available, and if something happens, it makes sure that the virtual machine is restarted somewhere else. This looks a lot like Linux HA Clustering. In fact, in Chapter 11, you'll even learn how to use Linux HA clustering to create such a solution for KVM Virtual machines.

There is a fundamental difference, though. The HA solution that is offered by your virtualization platform is agnostic on what happens in the virtual machine. That means that if a virtual machine hangs, it will appear as available to the virtualization layer, and the HA solution of your virtualization layer will do nothing. It also is incapable of monitoring the status of critical resources that are running on those virtual machines.

If you want to make sure that your company's vital resources have maximum protection and are restarted as soon as something goes wrong with them, you'll require high availability within the virtual machine. If the virtual machine runs the Windows operating system, you'll need Windows HA. In this book, you'll learn how to set up such an environment for the Linux operating system.

History of High Availability Clustering in Linux

High availability in Linux has a long history. It started in the 1990s as a very simple solution with the name Heartbeat. A Heartbeat cluster basically could do two things: it monitored two nodes (and not more than two), and it was configured to start one or more services on those two nodes. If the node that was currently hosting the resources went down, it restarted the cluster resources on the remaining node.

Heartbeat 2.0 and Red Hat Cluster Suite

There was no monitoring of the resources themselves in the early versions of Heartbeat, and there was no possibility to add more than two nodes to the cluster. This changed with the release of Heartbeat 2.0 in the early 2000s. The current state of Linux HA clustering is based in large part on Heartbeat 2.0.

Apart from Heartbeat, there was another solution for clustering: Red Hat Cluster Suite (now sold as the Red Hat High Availability Add On). The functionality of this solution looked a lot like the functionality of Heartbeat, but it was more sophisticated, especially in the early days of Linux HA clustering. Back in those days, it was a completely different solution, but later, the Red Hat clustering components merged more and more with the Heartbeat components, and in the current state, the differences are not so obvious.

Cluster Membership and Resource Management

An important step in the history of clustering was when Heartbeat 2.0 was split into two different projects. Clustering had become too complex, and therefore, a project was founded to take care of the cluster membership, and another project took care of resource management. This difference exists to the current day.

The main function of the cluster membership layer is to monitor the availability of nodes. This function was first performed by the OpenAIS project, which later merged into the Corosync project. In current Linux clustering, Corosync still is the dominant solution for managing and monitoring node membership. In Red Hat clustering, cman has always been used as the implementation of the cluster membership layer. Cman isn't used often in environments without Red Hat, but in Red Hat environments, it still plays a significant role, as you will learn in Chapter 3.

For resource management, Heartbeat evolved into Pacemaker, which, as its name suggests, was developed to fix everything that Heartbeat wasn't capable of. The core component of Pacemaker is the CRM, or cluster resource manager. This part of the cluster monitors the availability of resources, and if an action has to be performed on resources, it instructs the local resource manager (LRM) that runs on every cluster node to perform the local operation.

In Red Hat, up to Red Hat 6, the resource group manager (rgmanager) was used for managing and placing resources. In Red Hat 6, however, Pacemaker was already offered as an alternative resource manager, and in Red Hat 7, Pacemaker has become the standard for managing resources in Red Hat as well.

The Components That Build a High Availability Cluster

To build a high availability cluster, you'll need more than just a few servers that are tied together. In this section, you'll get an overview of the different components that typically play a role when setting up the cluster. In later chapters, you'll learn in detail how to manage these different components. Typically, the following components are used in most clusters:

- Shared storage
- Different networks
- Bonded network devices
- Multipathing
- Fencing/STONITH devices

It is important to think about how you want to design your cluster and to find out which specific components are required to build the solution you need.

Shared Storage

In a cluster, it's the cluster that decides on which server the shared resources are going to be hosted. On that server, the data and configuration files have to be available. That is why most clusters need shared storage. There are exceptions, though.

Some services don't really have many files that have to be shared, or take care of synchronization of data internally. If your service works with static files only, you might as well copy these files over manually, or set up a file synchronization job that takes care of synchronizing the files in an automated way. But most clusters will have shared storage.

Roughly speaking, there are two approaches to taking care of shared storage. You can use a Network File System (NFS) or a storage area network (SAN). In an NFS, one or more directories are shared over the network. It's an easy way of setting up shared storage, but it doesn't give you the best possible flexibility. That is why many clusters are set up with an SAN.

A SAN is like a collection of external disks that is connected to your server. To access a SAN, you'll need a specific infrastructure. This infrastructure can be Fibre Channel or iSCSI.

Fibre Channel SANs typically are built for the best possible performance. They're using a dedicated SAN infrastructure, which is normally rather expensive. Typically, Fibre Channel SANs costs tens of thousands of dollars, but you get what you pay for: good quality with optimal performance and optimal reliability.

iSCSI SANs were developed to send SCSI commands over an IP network. That means that for iSCSI SAN, a normal Ethernet network can be used. This makes iSCSI a lot more accessible, as anyone can build an iSCSI SAN, based on standard networking hardware. This accessibility gives iSCSI SANs a reputation for being cheap and not so reliable. The contrary is true, though. There are some vendors on the market who develop high-level iSCSI SAN solutions, where everything is optimized for the best possible performance. So, in the end, it doesn't really matter, and both iSCSI and Fibre Channel SANs can be used to offer enterprise-level performance.

Different Networks

You could create a cluster and have all traffic go over the same network. That isn't really efficient, however, because a user who saturates bandwidth on the network would be capable of bringing the cluster down, as the saturated network cluster packets wouldn't come through. Therefore, a typical cluster has multiple network connections (Figure 1-4).

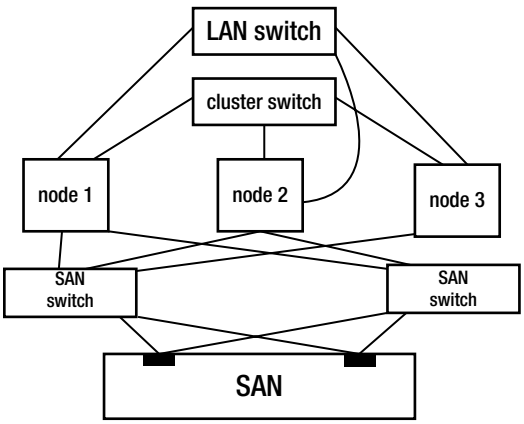

Figure 1-4. *Typical cluster network layout*

First, there is the user network, from which external users access the cluster resources. Next, you would normally have a dedicated network for the cluster protocol packets. This network is to offer the best possible redundancy and ensure that the cluster traffic can come through at all times.

Third, there would typically be a storage network as well. How this storage network is configured depends on the kind of storage that you're using. In a Fibre Channel SAN, the nodes in the cluster would have host bus adapters (HBAs) to connect to the Fibre Channel SAN. On an iSCSI network, the SAN traffic goes over an Ethernet network, and nothing specific is required for the storage network except a dedicated storage network infrastructure.

Bonded Network Devices

To connect cluster nodes to their different networks, you could, of course, use just one network interface. If that interface goes down, the node would lose connection on that network, and the cluster would react. As a cluster is all about high availability, this is not what you typically want to accomplish with your cluster.

The solution is to use network bonding. A network bond is an aggregate of multiple network interfaces. In most configurations, there are two interfaces in a bond. The purpose of network bonding is redundancy: a bond makes sure that if one interface goes down, the other interface will take over. In Chapter 3, you will learn how to set up bonded network interfaces.

Multipathing

When a cluster node is connected to a SAN, there are typically multiple paths the node can follow to see the LUN (logical unit number) on the SAN. This results in the node seeing multiple devices, instead of just one. So, for every path the node has to the LUN, it would receive a device.

In a configuration where a node is connected to two different SAN switches, which, in turn, are connected to two different SAN controllers, there would be four different paths. The result would be that your node wouldn't see only one iSCSI disk, but four. As each of these disks is connected to a specific path, it's not a good idea to use any single one of them. That is why multipath is important.

The multipath driver will detect that the four different disks are, in fact, all just one and the same disk. It offers a specific device, on top of the four different disks, that is going to be used instead. Typically, this device would have a name such as mpatha. The result is that the administrator can connect to mpatha instead of all of the underlying devices, and if one of the paths in the configuration goes down, that wouldn't really matter, as the multipath layer would take care of routing traffic to an interface that still is available. In Chapter 2, you will learn how to set up multipathing.

Fencing/STONITH Devices and Quorum

In a cluster, a situation called split brain needs to be avoided. Split brain means that the cluster is split in two (or more) parts, but both parts think they are the only remaining part of the cluster. This can lead to very bad situations when both parts of the cluster try to host the resources that are offered by the cluster. If the resource is a file system, and multiple nodes try to write to the file system simultaneously and without coordination, it may lead to corruption of the file system and the loss of data. As it is the purpose of a high availability cluster to avoid situations where data could be lost, this must be prevented no matter what.

To offer a solution for split-brain situations, there are two important approaches. First, there is quorum. *Quorum* means "majority," and the idea behind quorum is easy to understand: if the cluster doesn't have quorum, no actions will be taken in the cluster. This by itself would offer a good solution to avoid the problem described previously, but to make sure that it can never happen that multiple nodes activate the same resources in the cluster, another mechanism is used as well. This mechanism is known as STONITH (which stands for "shoot the other node in the head"), or fencing. Both the terms *STONITH* and *fencing* refer to the same solution.

In STONITH, specific hardware is used to terminate a node that is no longer responsive to the cluster. The idea behind STONITH is that before migrating resources to another node in the cluster, the cluster has to confirm that the node in question really is down. To do this, the cluster will send a shutdown action to the STONITH device, which will, in turn, terminate the nonresponsive node. This may sound like a drastic approach, but as it guarantees that no data corruption can ever occur and can clean up certain transient errors (such as a kernel crash), it's not that bad.

When setting up a cluster, you must decide which type of STONITH device you want to use. This is a mandatory decision, as STONITH is mandatory and not optional in Linux HA clusters. The following different types of STONITH devices are available:

- Integrated management boards, such as HP ILO, Dell DRAC ,and IBM RSA
- Power switches that can be managed, such as the APC master device
- Disk-based STONITH, which uses a shared disk device to effectuate the STONITH operation
- Hypervisor-based STONITH, which talks to the hypervisor in a virtualization environment
- Software and demo STONITH solutions (which, in fact, should be avoided at all times)

In Chapter 5, you'll learn how to configure different STONITH and fencing solutions.

Summary

This chapter has given an overview of Linux HA clustering. You've read how HA clustering relates to other types of clustering, and you've learned about the different software components that are used in a typical HA environment. You also have learned about the different parts that are used in high availability clustering, which allows you to properly prepare your high availability environment. In the next chapter, you'll learn how to configure and connect to storage in high availability environments.

CHAPTER 2

Configuring Storage

Almost all clusters are using shared storage in some way. This chapter is about connecting your cluster to shared storage. Apart from connecting to shared storage, you'll also learn how to set up an iSCSI storage area network (SAN) in your own environment, a subject that is even further explored in Chapter 10. You'll also learn the differences between network attached storage (NAS) and SAN and when to use which. The following topics are covered in this chapter:

- Why most clusters need shared storage
- NAS or SAN?
- iSCSI or Fibre Channel?
- Configuring the LIO iSCSI target
- Connecting to an iSCSI SAN
- Setting up multipathing

Why Most Clusters Need Shared Storage

In an HA cluster, you will make sure that vital resources will be available as much as possible. That means that at one time, your resource may be running on one node, while at another time, the resource may be running on another node. On the other node, the resource will need access to the exact same files, however. That is why shared storage may come in handy.

If your resource only deals with static files, you could do without shared storage. If modifications to the files are only applied infrequently, you could just manually copy the files over, or use a solution such as rsync to synchronize the files to the other nodes in the cluster. If, however, the data is dynamic and changes are frequent, you'll need shared storage.

Typically, a resource uses three different kinds of files. First, there are the binaries that make up the program or service that is offered by the resource. It is best to install these binaries locally on each host. That ensures that every single host in its update procedures will update the required binaries, and it will make sure that the host can still run the application if very bad things are happening to the cluster and you're forced to run everything stand-alone.

The second part of data that is typically used are configuration files. Even if many applications store configuration files by default in the local `/etc` directory, most applications do have an option to store the configuration files somewhere else. It often is a good idea to put these configuration files on the shared storage. This ensures that your cluster application always has access to the same configuration. In theory, manual synchronization of configuration files between hosts would work as well, but in real life, something always goes wrong, and you risk ending up with two different versions of the same application. So, make sure to put the configuration files on the shared storage and configure your application to access the files from the shared storage and not locally.

The third and most important type of files that applications typically work with is the data files. These normally are a valuable asset for the company, and also, they have to be available from all nodes at all times. That is why the nodes in the cluster are configured to access an SAN disk and the data files are stored on the SAN disk. This ensures that all hosts at all times can access the files from the SAN. The SAN itself is set up in a redundant way, to ensure that the files are highly protected and no interruption of services could occur because of bad configuration. See Figure 2-1 for an overview of this setup.

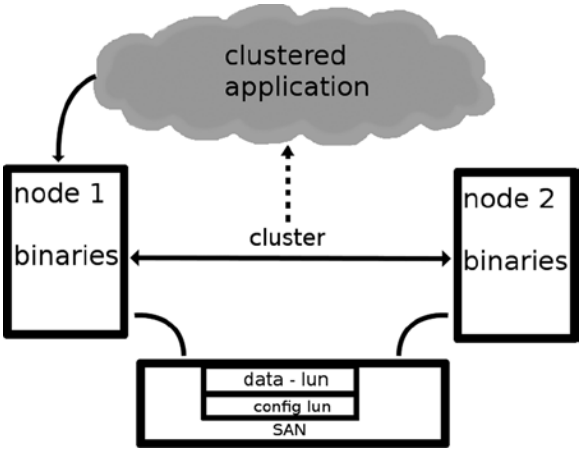

Figure 2-1. *Cluster application file access overview*

NAS or SAN?

When choosing the right solution for shared storage, you must select between network attached storage (NAS) and storage area networks (SAN). Let's discuss some differences and advantages between these two.

NAS

Network attached storage (NAS) is basically a network share that could be offered by any server on the network. In Linux clusters, NAS is typically provided by means of Network File System (NFS) shares, but a Common Internet File System (CIFS) is also a valid option to provide NAS functionality. The advantage of an NAS is that it is simple to set up. There are some other considerations, though.

Typically, NAS services are provided by a server in the network. Now, when setting up a cluster environment, it is of greatest importance to avoid having a single point of failure in the network. So, if you were planning to set up an NFS server to provide for shared storage in your cluster environment, you would have to cluster that as well, to make sure that the shared storage was still available if the primary NFS server dropped. So, you would have to cluster the NFS or CIFS server and make sure that no matter where the services itself were running, it had access to the same files. HA NAS servers that are using NFS or CIFS are commonly applied in HA cluster environments.

A common reason why NAS solutions are used in HA environments is because an NAS gives concurrent file system access, which an SAN won't, unless it is set up with OCFS2 or GFS2 at the client side.

SAN

A storage area network (SAN) is tailored to offer the best possible redundancy, as well as performance to access storage (Figure 2-2). It typically consists of disk arrays. To access these disks, a dedicated network infrastructure is used.

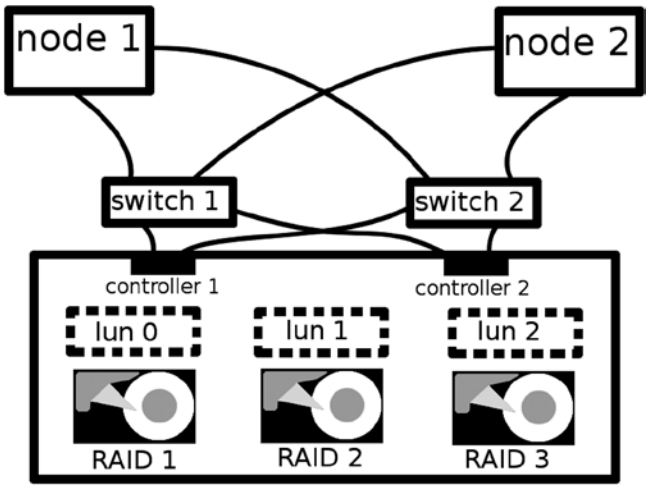

Figure 2-2. *SAN overview*

The disks in the SAN filer are normally set up using RAID. Typically, different RAID arrays are configured to make sure the SAN can survive a crash of several disks simultaneously. On top of those RAID arrays, the logical unit numbers (LUNs) are created. Nodes in the cluster can be authorized to access specific LUNs, which to them will appear as new local disks.

To access the SAN filer, a redundant network infrastructure is normally used. In this infrastructure, most items are double, which means that the nodes have two SAN interfaces that are connected to two SAN switches, which are, in turn, connected to two different controllers on the SAN. All this is to make sure that if something fails, the end user won't notice anything.

iSCSI or Fibre Channel?

Once you have decided to use a storage area network (SAN), the next question arises: is it going to be Fibre Channel or iSCSI? The first SANs that came on the market were Fibre Channel SANs. These were filers that were optimized for the best possible performance and in which a dedicated SAN network infrastructure was used as well. That is because in the time the first SAN solutions appeared, 100 Mbit/s was about the fastest speed available on LAN networks, and compared to the throughput on a local SCSI bus, that was way too slow. Also, networks in those days were using hubs most of the time, which meant that network traffic was dealt with in a rather inefficient way.

However, times have changed, and LAN networks became faster and faster. Gigabit is the minimum standard in current networks, and nearly all hubs have been replaced with switches. In the context of these improved networks, a new standard was created: iSCSI. The idea behind iSCSI is simple: the SCSI packets that are generated and sent on a local disk infrastructure are encapsulated in an IP header to address the SAN.

Fibre Channel SAN has the reputation of being more reliable and faster than iSCSI. This doesn't have to be true, though. Some high-end iSCSI solutions are offered on the market, and if a dedicated iSCSI network is used, where traffic is optimized to handle storage, iSCSI can be as fast and as reliable as Fibre Channel SAN. iSCSI does have an advantage that Fibre Channel SANs don't offer, and that is the relatively easy way that iSCSI SAN solutions can be created. In this chapter, for example, you will learn how to set up an iSCSI SAN yourself.

Another alternative to implement Fibre Channel technology without the need to purchase expensive Fibre Channel hardware is to use Fibre Channel over Ethernet (FCoE). This solution allows Fibre Channel to use 10 Gigabit Ethernet (or faster), while preserving the Fibre Channel protocol. FCoE solutions are available in the major Linux distributions.

Understanding iSCSI

In an iSCSI configuration, you're dealing with two different parts: the iSCSI target and the iSCSI initiator (Figure 2-3). The iSCSI target is the storage area network (SAN). It runs specific software that is available on TCP port 3260 of the SAN and that provides access to the logical unit numbers (LUNs) that are offered on the SAN. The iSCSI initiator is software that runs on the nodes in the cluster and connects to the iSCSI target.

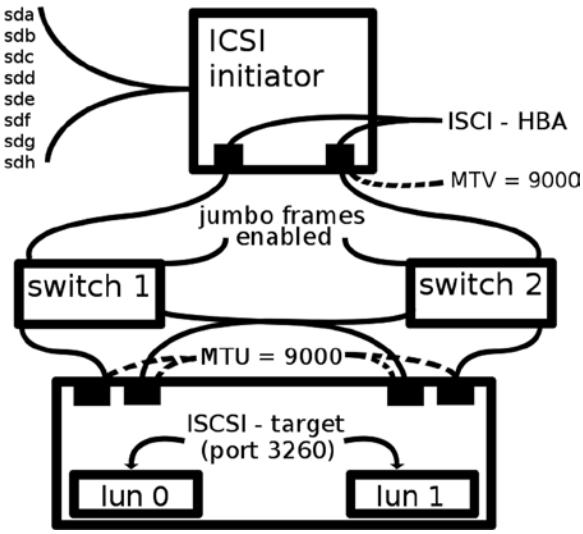

Figure 2-3. *iSCSI overview*

To connect to the iSCSI target, a dedicated SAN network is used. It normally is a regular Ethernet network, but configured in a specific way. To start with, the network is redundant. That means that two different network interfaces on the cluster nodes connect to two different switches, which in turn connect to two different controllers on the SAN that each are accessible by two different network interfaces as well. That means that no less than four different paths exist to access the LUNs that are shared on the SAN. That leads to a situation where every LUN risks being seen four times by the cluster nodes. You'll read more about this in the section about multipathing later in this chapter.

On the SAN network, some specific optimizations can be applied as well. Optimization begins on the cluster nodes, where the administrator can choose to select, not ordinary network cards, but iSCSI host bus adapters (HBAs). These are smart network cards that have been produced to handle iSCSI traffic in the most optimal way. They have their maximum packet size on the Ethernet level set to an MTU of 9000 bytes, to make sure the traffic is handled as fast as possible, and they often use an iSCSI offload engine to handle the iSCSI traffic even more efficiently. However, iSCSI HBAs have become less popular and tend to be replaced by fast network interface cards (NICs).

Configuring the LIO iSCSI Target

There are many different vendors on the market that make iSCSI solutions, but you can also set up iSCSI on Linux. The Linux-IO (LIO) Target is the most common iSCSI target for Linux on recent distributions (Figure 2-4). You will find it on all recent distributions. On SUSE Linux Enterprise Server 12, for instance, you can easily set it up from the YaST management utility. On other distributions, you might find the `targetcli` utility to configure the iSCSI target. Of course, when setting up a single iSCSI target, you must realize that this can be a single point of failure. Later in this chapter, you'll learn how to set up iSCSI targets in a redundant way.

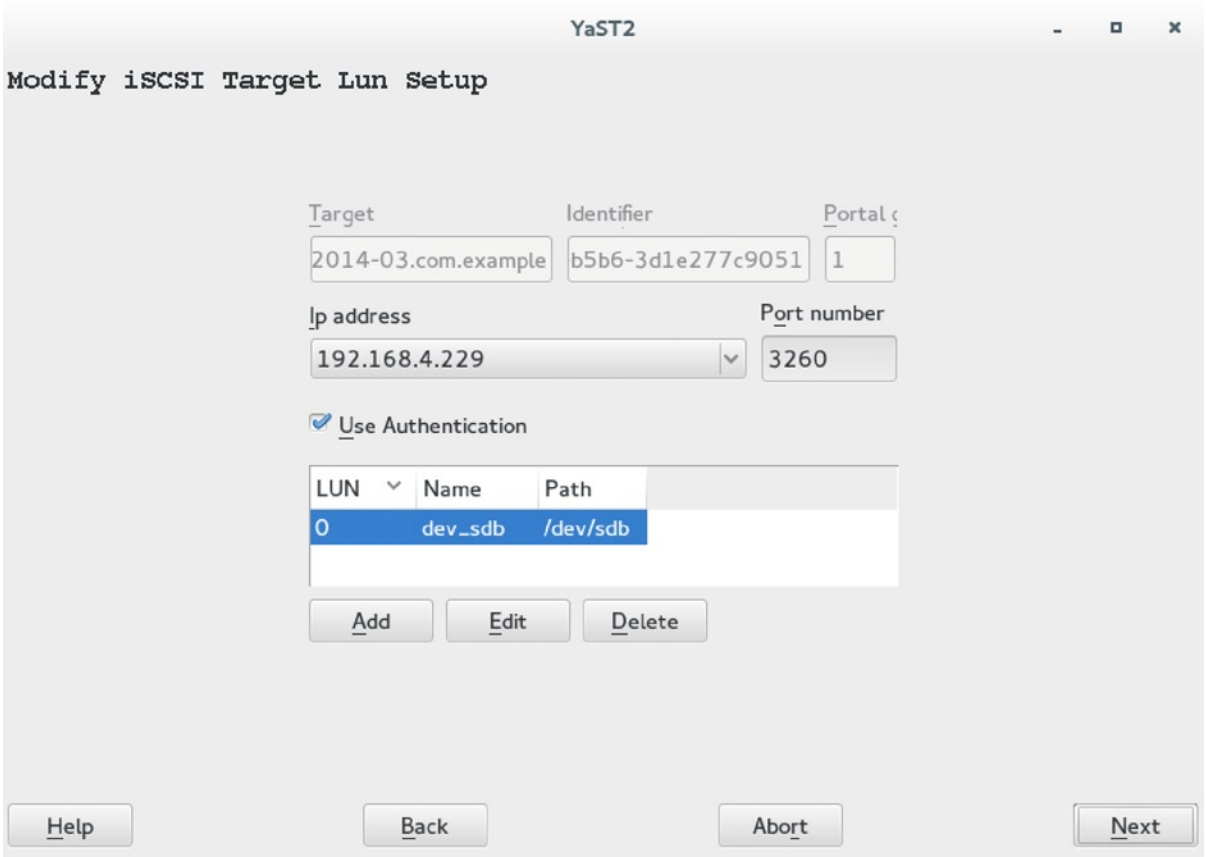

Figure 2-4. *Setting up the LIO Target from SUSE YaST*

When setting up a target, you must specify the required components. These include the following:

- *Storage device*: This is the storage device that you're going to create. If you're using Linux as the target, it makes sense to use LVM logical volumes as the underlying storage device, because they are so flexible. But you can choose other storage devices as well, such as partitions, complete hard disks, or sparse files.
- *LUN ID*: Every storage device that is shared with an iSCSI target is shared as a LUN, and every LUN needs a unique ID. A LUN ID is like a partition ID; the only requirement is that it has to be unique. There's nothing wrong selecting subsequent numeric LUN IDs for this purpose.
- *Target ID*: If you want to authorize targets to specific nodes, it makes sense to create different targets where every target has its own target ID, also known as the Internet Qualified Name (IQN). From the iSCSI client you need the target ID to connect, so make sure the target ID makes sense and makes it easy for you to recognize a specific target.
- *Identifier*: The identifier helps you to further identify specific iSCSI targets.
- *Port number*: This is the TCP port the target will be listening on. By default, port 3260 is used for this purpose.