Tuning Linux

Mark Komarinski

mkomarinski@wayga.org

Michael C. Jett

Tuning Linux

by Mark Komarinski

by Michael C. Jett

This work covers tuning Linux and the hardware it runs on for the best efficiency. This can make existing hardware run faster, or give network architechts information to make informed decisions about servers and their requirements.

Revision History

Revision 0.30 2001-4-17 Revised by: mfk
Outline stuff from Michael C. Jett
Revision 0.20 2001-3-27 Revised by: mfk
Created network and disk chapters. Need more info!
Revision 0.10 2001-2-28 Revised by: mfk
Second pass. Time to add some data.
Revision 0.01 2001-01-25 Revised by: mfk
First pass for the book. Define the outline.

Table of Contents

List of Tables

Foreward

The idea of a book on tuning Linux for performance is pretty much a moving target. Linux makes advances at a rapid pace, and things change often. Since its introduction in 1991, Linux has gone from a state where it could not compile itself (and barely boot) into a system that can take up to three full CD-ROMs to install much of the software. It rivals operating systems twice its age in terms of performance and stability. Its complete heritage may never be known because of the thousands of people around the world who have contributed patches, drivers, and enhancements to make Linux better.

This advancement means that some portions of Linux can literally turn on a dime. The Linux 2.0 kernel used a program called ipfwadm to administer its firewall settings. When 2.2 came out, the program was ipchains, and used completely different concepts than ipfwadm. The 2.4 kernel uses iptables, which itself is different from its predecessors. This means that Linux has little baggage holding it back when it comes to backwards compatability. Some open source applications need no changes, most may need to be recompiled, and some require programming changes that can be done by any programmer. The idea of backwards compatability in Linux is a foreign concept.

But tuning Linux is more than just tuning what you have. It is looking at your requirements, evaluating the technology that can solve that requirement, and going with the balance between cost and performance. Not everyone can afford the best cards, but the difference between good and poor performance can be a very small cost as compared to the overall system cost.

Copyright

Chapter 1. Introduction

This book is arranged into a few major sections. This will allow you to read the sections of the book that apply to the kind of tuning you want to do.

- Fundamentals

  What is the nature of performance tuning? How should you go about looking at an entire system in terms of improving the services it provides to others? What common strategies are available? We will go through the basics of performance tuning in detail for this chapter.

- Measuring performance

  This chapter will deal with measuring performance from a system on various levels, outling common tools bundled with your Linux system or available on the Internet. Once you know how fast your system is in its current state, you have a baseline to measure against.

- Disk

  This section will compare various kinds of permanent storage. We'll take a look at hard drives, how to write to them, and any tradeoffs between reliability and raw speed.

- Kernel

  This section will cover some of the popular kernels that are available and how you can use the built in tools to tune it for maximum performance. We'll cover things like the /proc directory, sysctl, and recompiling kernels.

- Network

  There are a variety of networking schemes, and one will fit your need. Even once you have chosen your networking method, there are ways of enhancing it for even better performance.

- Applications

  This section is not strictly about Linux, but poorly configured applications can quickly erode any advantages from other tuning methods. We will take a look at tuning various web servers, database applications, and cluster software.

1.1. Conventions

This book uses the following conventions:

Descriptions	Appearance
Warnings	**Caution** Warnings.
Hint	**Tip:** Hint.
Notes	**Note:** Note.

Descriptions	Appearance
Information requiring special attention	**Warning** Warning.
File Names	`file.extension`
Directory Names	`directory`
Commands to be typed	**command**
Applications Names	application
Prompt of users command under bash shell	`$`
Prompt of root users command under bash shell	`#`
Prompt of user command under tcsh shell	`tcsh$`
Environment Variables	VARIABLE
Emphasized word	*word*
Code Example	`<para>Beginning and end of paragraph</para>`

Chapter 2. System Performance Fundamentals

There are a few fundamental concepts important to understanding the performance characteristics of a system. Most of these are very obvious, they are just not often thought of when dealing with tuning a system.

2.1. Viewing a server as a system of components

The first, and most important, concept is looking at a server as a system of components. While you may get tremendous performance gains by moving from 7200 RPM to 10,000 RPM drives, you may not. To determine what needs to be done, and what benefit improving a particular component will be, you must first lay out the system and evaluate each component.

In a typical server sytem the major components are:

Processor

The processor, or processors, in a system are quite obviously the little black chip, or card edge socket, or big card/chip and heatsink combination on the motherboard. But the processor subsytem referred to here includes not only the processor, but also the L1 and L2 processor caches, the system bus, and various motherboard components.

Physical Disk

The physical disk subsytem includes the disk drive spindles themselves, as well as the interface between drive and controller/adapter card, the adapter card or cards, and the bus interface between controller/adapter card and the motherboard.

Network

The definition of the network subsystem varies depending on who you ask. All definitions include the network card and its bus interface, and the physical media it connects to, such as Ethernet. If you are looking at improving the performance of a server this is all that concerns you. However, in a larger analysis of performance this subsystem may include and number of network links between your server and clients.

Memory

The memory subsystem includes two major types of memory, real and virtual. Real memory is the chip, chips, or cards in the computer that provide physical memory. Virtual memory includes this memory as well as the virtual memory stored on disk, often referred to as a paging file or swap partition.

Operating System

The single most complex component of any system is the operating system. While there are more opportunities for tuning here, there are also more opportunities to make a simple mistake that will cost you thousands in additional hardware to handle the load.

2.2. Bottlenecks and Process Times

The next concept to understand is the total time through a system for one request. Let's say you want a hamburger from McDonald's. You walk in the door and there are four people in line in front of you (just like normal). The person taking and filling orders takes care of one person in 2 minutes. So the person being waited on now will be done in 2 minutes, when the next person in line will be waited on, which takes 2 minutes. In all it will be 8 minutes before you give the friendly counter person your order, and 10 minutes total before you get your food.

If we cut the time it takes for the counter person to wait on someone from 2 minutes to 1, everything moves faster. Each person takes 1 minute, so you have your order in 5. But since each person is in line a shorter period of time, the probability that there will be 4 people in line in front of you decreases as well. So improving processing time of a resource that has a queue waiting for it provides a potentially exponential improvement in total time through the system. This is why we address bottlenecked resources first.

The reason we say "potentially exponential" improvement is that the actual improvement depends on a number of factors, which are discussed further. Basically, the process time is most likely not constant, but depends on many factors, including availability of other system resources on which it could be blocked waiting for a response. In addition, the arrival rate of requests is probably not constant, but instead spread on some type of distribution. So if you happen to arrive during a burst of customers, like at lunch hour, when the kitchen is backed up getting hamburgers out for the drive through, you could still see a 2 minute processing time and 4 people in line in front of you. Actually determining the probabilities of these events is an exercise left to the reader, as is hitting yourself in the head with a brick. The idea here is just to introduce you to the concepts so you understand why your system performs as it does.

Another concept in high performance computing is parallel computing, used for applications like Beowulf. If your application is written correctly, this is like walking into McDonalds and seeing 4 people in line, but there are 4 lines open. So you can walk right up to a counter and get your burger in about two minutes. The down side of this is that the manager of the store has to pay for more people working in the store, or you have to buy more machines. There are also potential bottlenecks, such as only one person in the back making fries, or the person in line in front of you is still trying to decide what they want.

2.3. Interrelation between subsystems

The last concept to cover here is the interrelation between subsystems on the total systems performance. If a system is short on memory, it will page excessively. This will put additional load on the drive subsystem. While tuning the drive subsystem will definitely help, the better answer is to add more memory.

As another example, your system may be performing just fine except for the network connection, which is 10Base-T. When you upgrade the network card to 100Base-T, performance improves only marginally.

Why? It could very well be that the network was the bottleneck, but removing that bottleneck immediately revealed a bottleneck on the disk system.

This makes it very important that you accurately measure where the bottleneck is. It can not only improve the time it takes you to speed up the system, but also save money, since you only need to upgrade one subsystem instead of many.

The best way to find out the interrelations depends on the application you are using. As we go through various subsystems and applications in this book, you will have to keep this in mind. For example, the Squid web cache has heavy memory and disk requirements. Using less memory will increase the disk usage, and decrease performance. By looking at memory usage, you can see that memory is being swapped to the hard drive. Increasing the available memory so that no more swapping of squid will be your best performance bet.

2.4. How Much Machine is Required?

A frequent question when deciding to buy new hardware is "How much hardware should be purchased for the task?". A good rule of thumb is "as much as you can afford", but that may not be the whole answer. You should consider the following when defining specifications.

- If you cannot afford a fully loaded system, can it be expanded after purchase? If you application would work best with 2GB of RAM, but you can only afford 1GB, can the motherboard be expanded to add the extra GB later on? Purchasing a motherboard that can only accept 1GB means you will need to buy a new motherboard (at least) later on, increasing cost.
- How well does the design scale? If you plan on buying racks of machines, can you add new racks later and integrate them easily with the existing system? Things like switches that can accept new blades to add functionality may save space and money in the long run, while reducing issues like cabling.
- How easy is it to install, upgrade, or repair? While using desktop systems in a rack is possible and will save money, replacing or repairing the systems is harder to do. A good question to ask yourself is "What is my time worth?".

Chapter 3. Measuring your performance

In order to see if your system is any faster, you need to measure the performance of the system before and after you try your tuning changes. There's a variety of applications for testing your hard drive, CPU, memory, and overall system performance. This allows you to also test a proposed configuration versus the existing configuration.

In all cases of testing, you should have a relatively quiet system, meaning that there is a minimum of applications running. For true testing, you should reboot the machine between each test, and run each test at least 5 times, then take the average. Rebooting clears any in-memory caches that can affect the tuning numbers, and also makes sure there is a larger amount of system RAM free.

3.1. Hard Drive

The easiest way to find raw hard drive performance is using **hdparm**, which we describe in Section 4.3.1.2. But this is raw hard drive performance, not taking into account things like overhead from the filesystem or write performance.

The other application you can use to test how the filesystem works, or for devices that do not work with **hdparm** is to use **dd**. The **dd** command is used to write or read data and perform some conversion along the way. The nice part of the command for us is that it can create a file of any size containing ASCII NULL (0), which allows us to test consistently. Since the data we want to write is being generated in the CPU and memory which is always faster than the hard drive, this gives a good look at the disk bottleneck.

This is used in conjunction with **time** which gives the amount of CPU, system, and user (real) time used to run a particular command. You can then divide out the size of the file created by the number of user seconds to run the command to get a Mbps rating.

3.2. CPU and System

Since CPUs have a variety of functions crammed into a small space, it is hard to test how fast a particular CPU is. A standard number for Linux is called "BogoMIPS", which Linus needed for some timing routines in the kernel. BogoMIPS calculate how fast a CPU can do nothing, and so will vary depending on the kind of chip used. Because of how BogoMIPS are calculated, they should not be used for any form of performance measurement.

There are a variety of CPU functions that can be measured, including Million Instructions Per Second (MIPS), Floating Point Operations Per Second (FLOPS), and memory to CPU speed (in MBps). Each of these will vary greatly depending on the choice of CPU. MIPS are almost worthless, since some chips can have one instruction that runs multiple other instructions. The MMX instruction set for Intel-based

hardware is a good example for this, as MMX is set to perform matrix operations from just a few instructions, whereas without MMX, it would take many more operations to do the same calulations. When measuring chip speed, does an MMX instruction count as one or multiple instructions? Since the speed of running one MMX instruction is slightly faster than running the multiple instructions that make it up, it will make an MMX-based chip appear slower.

Many online reviewers use games as a measurement of the CPU. The idea is that 3D games provide a good stress test of the CPU, memory interface, system bus, and video card. The result is the number of Frames Per Second (FPS). More FPS means a faster overall system. Since most servers do not have a 3D card in them, this may not be a good choice of measurement. Workstations may get a benefit from this kind of testing, however.

One other measurement form is that of other third-party applications. Applications like SETI@Home, distributed.net can give generic speed ratings that are good to compare against other machines running the same software. Other CPU-intensive applications like MP3 encoders can also be a good guide of how fast a CPU is. The down side to these are that the results have to be compared to the results of other machines running the exact same software. If the software is not available for that OS/Hardware combination, the test is worthless.

So what does this leave us with? Unfortunately, not a lot. The best bet appears to be MIPS and FLOPS, since it does test instructions per second, and as long as the chipset remains the same (Intel based, SPARC, Alpha, etc.), the measurements can be compared pretty easily and the comparisons will be close enough.

3.3. Network

Measuring the network speed is not quite easy, since there are bottlenecks outside the network interface and your machine. Cable problems, poor choice of switching gear, and congestion on the line or the remote machine you want to test against can all reduce the throughput of your network interface. In addition, protocols like SSH or HTTP have additional processing that may need to be done that occupies the CPU and reduces throughput.

If the machine you are testing is going to be a server, you can create a small group of client machines on a private network. These machines should have the same type of network card and OS revision. This will create a stable baseline of testing.

Depending on the networking application you will be using, there may be applications that already exist to automate this testing for you. Programs like webbench can coordinate the clients talking to the server, and be able to read the performance of the server in terms of pages per second.

3.4. Video card

If you want to tune a workstation, or create a killer Quake III box, you will want to pay attention to the video subsystem and see how well it performs. For 2D applications, you can test the system using **x11perf**. This program will perform a variety of tests using the X server and drivers. Since it is designed for performance testing, it is designed to run each test five times, then take the average. The machine should have no other users or activity going on, and you should disable the screen saver. You can disable the screensaver either with the command **xset s off** or by killing the process called "xscreensaver". You may need to run both commands in order to turn off the screen saver.

Testing with **x11perf** will take several hours depending on the speed of your CPU and graphics card. Once complete, you will have a log file that you can then compare against other machines that have also done testing, or against a baseline test you ran before tuning to see if the X server speed has been improved. To do this comparison, you can use **x11perfcomp** to compare two or more tests. Higher numbers are better, as the resulting numbers are in terms of objects per second.

You can test out 3D performance using applications like Quake III, that run the application through a set world and events, most of which will stress the system. Mark down the resolution, bit depth, and frames per second reported from Quake III, and you now have a baseline to work from.

A caveat to using Quake or 3D applications is that this is testing more than the video card. Other subsystems, like the CPU, video drivers, GLX (3D) drivers, and memory are also tested. If you want to use this method for comparing speed, you should also be sure the other subsystems are tuned as well.

Chapter 4. Disk Tuning

4.1. Introduction to disk tuning

Your drive system is one of the places where bottlenecks can occur. All of your database information, boot code, swap space, and user programs live on the hard drives. Hard drives are speeding up, now topping 15,000 RPM and bus rates of over 160MB per second. Even at these speeds, drives are still much slower than RAM or your CPU, with requests to drives waiting for the drive to spin to the right location, read and/or write the data that is required, and send the answer back to the CPU.

To top this off, hard drives are one of the moving parts in your machine, and moving parts are more likely to fail over time than a solid state mechanism, like your memory or CPU. One of the reasons why power supplies have such low MTBFs is the fan that keeps the power supply cool has a low MTBF. Once the fan dies, it's only a matter of time before the power supply itself overheats and dies. Some systems overcome this by installing multiple power supplies, so if one dies, the others can take over. Fortunately for hard drives, this failover is available in the form of RAID (Redundant Array of Inexpensive Disks).

Then there are questions about the cost of differing systems versus their performance versus other options. We will take a look at these options throughout this chapter, starting with an overview of existing hard drive technology for Linux.

4.2. Overview of Disk Technologies

There are, as you may know, two major hard disk technologies that work under Linux: Integrated Drive Electronics (IDE), and Small Computer System Interface (SCSI). At the lowest level, IDE and SCSI drives share enough technology that IDE and SCSI drives are exactly the same save for a PCB board and physical interface on the disk itself. That PCB and phyiscal interface can command a premium in price on drive.

4.2.1. SCSI drives

The history of SCSI drives starts not with large servers running multibillion dollar companies, but as its name implies, with small systems. SCSI had one of its first implementation on the Apple Macintosh Plus systems. SCSI was made to be a general purpose interface, supporting not only hard drives, but scanners, CD-ROMs, serial ports, and other devices off of one common bus. Cables were inexpensive, the protocol was relatively open, and was fast for its time, reaching a maximum of 4MB per second.

SCSI was picked up by server vendors, such as Sun and IBM, since it created a common way for servers to share devices. Its advantages in design made it very well suited to have multiple devices on the same

bus with little contention between devices. Unlike IDE, when a SCSI controller makes a request, the drive drops its hold on the line, fetches the information, and then picks up the line and sends the response. This allows the CPU to continue working on other tasks, so it is not I/O bound waiting for the drive. Other devices on the SCSI bus can communicate during this time. Another advantage to SCSI is that you can have multiple SCSI controllers on the same bus. This allows two or more machines to access a single SCSI bus and drives that are on that bus.

The increased performance of SCSI and low quantity of SCSI drives being made as compared to IDE drives has caused an increase in price of SCSI drives. In many cases, this can be almost double the price. But if you need the performance, the cost is worth it.

4.2.2. IDE drives

SCSI drives are actually stupid. They rely on a controller chip to tell them what to do. In many cases, a SCSI drive used by one SCSI controller cannot be moved to another controller and expect to keep all the data safe, since each controller has its own way of formatting data on the drive. IDE drives contain most of the brains on the drive itself. This makes it easier for system vendors to add support into their systems, reducing cost. These days, most systems will integrate serial ports, IDE, and USB all in one chip. SCSI still requires a (large) dedicated chip on the system to work.

The lower cost of parts and higher volume of IDE has made it very low cost for most single user systems while retaining some of the higher performance of SCSI. But many of the features of SCSI are not available in IDE: external boxes, hot swapping, and devices like scanners. In addition, when an IDE drive is asked to do something, that IDE bus is locked while the drive processes the request. IDE devices are usually limited to two devices per bus, but can have multiple busses per system. Most motherboards come with two IDE busses standard.

Burst performance of IDE drives gets close to that of SCSI, but will really only be seen on single user systems such as personal workstations.

In the past few years, IDE has had trouble with the existing x86 BIOS and larger capacity IDE drives. Since the BIOS has only a limited amount of space to store drive information, no way for supporting very large drives was almost impossible without modifying the BIOS itself, which would cause headaches for do-it-yourself system builders. Until 2000, Linux would not boot on partitions that were larger than 1GB. Fortunately, more recent BIOSes are smart enough to work around these size limitations, and Linux has found a way around its booting issues.

The future of IDE includes more of the existing ATA/66 and ATA/100 standards, which provide 66 Mbit and 100 Mbit burst speeds, repspectively. Also upcoming is Serial ATA, which uses a lower number of connectors and boosts speed rates to over 1Gbps.

4.2.3. What Interface Should You Use?

Based on the way that IDE and SCSI differ, you should take performance into mind when choosing a hard drive bus for your systems. A machine that is part of a cluster that relies on network storage can use the less expensive IDE drives, since the drive will not be a bottleneck. A machine that will be used as the network storage device should have the faster SCSI interface.

There are plenty of grey areas between these two extremes. But you should ask yourself a few questions while deciding:

- Will there be more than one drive in the system?
- Will the hard drive interface be a bottleneck on the system?
- Can the system the drives will be installed in support the bus I intend to use?

Considering the answers to these questions will get you a long way to answering what bus you want to use.

4.3. Tuning your drives

Now that we understand the major types of drives available for your system, let's take a look at the ways you can tune your particular setup.

4.3.1. Tuning the IDE bus

There are a few methods of tuning the IDE bus, both through the BIOS setup, and within Linux. Some of these options are not supported by all controllers and all drives, so you will want to compare specifications of each before testing. As always, be aware that some tuning commands can cause data loss, so be sure to back up your data before experimenting.

4.3.1.1. Tuning IDE from within the BIOS

Typically, most users will just leave the BIOS to automatically configure itself based off a negotiation between the hard drive and the controller in the system. There are, however, a few options within the BIOS that users can examine to get either improved performance, or debug tricky issues.

There are three methods for IDE to address a drive: Normal, CHS, and LBA. Normal mode works only with drives smaller than 500MB. Since no drives since 1996 have been made this small, we can forget this mode. CHS stands for Cylinder, Head, and Sector, while LBA stands for Logical Block Address.

CHS and LBA are fairly similar in performance, but be aware that taking a drive that is configured in CHS and moving it to a system that use LBA will destroy all the data on that drive.

Now that the drive can be accessed, there are two methods for that drive to get its data from the drive to the CPU: PIO, and DMA. PIO requires the CPU to shuttle data from the drive to memory so the CPU can use it later, but is the most compatible way for Linux to talk to a drive due to the large number of IDE controller chips. DMA mode seems to be much faster, since the CPU is not involved in moving the data between the drive and memory, freeing the CPU to do other things. Until the later Linux 2.2 kernels, DMA mode was not selected by default for access to drives due to compatability with interface chips. You can now select a specific IDE controller and select DMA access to make sure your combination is correct.

4.3.1.2. Tuning IDE from within Linux

As mentioned in Section 4.3.1.1, Linux has drivers for some specific IDE controllers, plus a generic IDE interface. Once the OS boots, Linux will ignore the state of the BIOS when it accesses hardware, choosing instead to use its own drivers and options. By default, the Linux kernel will have DMA access available, but not activated. This can be changed if you choose to compile your own kernel (see Section 6.2 for more information).

Once Linux is started, all IDE tuning occurs using **hdparm**. Using **hdparm** followed by a drive (**hdparm /dev/hda**) will return the current settings for that drive.

hdparm [-c *iovalue*] [-d] [-m *multimode*] {*device*}

```
# hdparm /dev/hda

/dev/hda:
multcount    =  0 (off)
I/O support  =  0 (default 16-bit)
unmaskirq    =  0 (off)
using_dma    =  1 (on)
keepsettings =  0 (off)
nowerr       =  0 (off)
readonly     =  0 (off)
readahead    =  8 (on)
geometry     = 1559/240/63, sectors = 23572080, start = 0
#
```

As you can see, there are a few options for tuning that are available for this drive. The biggest boost for IDE drive performance is to use DMA access, and this is already turned on. If DMA access is not turned on by default in the kernel, you can use -d1 to turn DMA access on, and -d0 to turn it off. Note that DMA access will not necessarily increase performance of getting data to and from the drive, but will free up the CPU during data transfers.

Another item to take a look at is the I/O support, which on our sample drive is set to 16-bit access instead of 32. The -c option to **hdparm** will check the I/O support for the specified device, there are a total of three options that can be given to -c to set the I/O support. The two most stable for systems are 0, used to set 16-bit access, and 3, used to set 32-bit synchronous access. Using an *`iovalue`* of 1 sets 32-bit access but may not work on all chipsets, causing a system hang if the chipset does not support regular 32-bit access.

Also test and check out multiple sector I/O with the -m option. Many newer IDE drives support transferrint multiple sectors worth of data per I/O cycle, reducing the numer of I/O cycles required to tranfer data. The -i option to hdparm will tell you what the maximum *`multimode`* is in the MaxMultSect section. By setting *`multimode`* to its highest, large transfers of data happen even quicker.

Warning

The **hdparm** manual page indicates that with some drives, using multiple sector I/O can result in massive filsystem corruption. As always, be sure to test your systems thoroughly before using them in a production environment.

Unfortunately, the correct setting for DMA, multiple sector I/O, and I/O support varies by drive and controller. You will have to do your own testing to find the correct combination for your system. The right combination should be able to:

- Have a high throughput of data from the drive to the system.
- Maintain that high throughput under heavy CPU load.
- Prevent the drive or the IDE driver from corrupting data.

Fortunately, testing throughput is easy with **hdparm**. The -t option tests the throughput of data directly from the drive, while -T tests throughput of data directly from the Linux drive buffer. To ensure the tests run properly, they should be run when the system is quiet and few other processes are running. Usually, runlevel 1, also known as single user mode, provides this environment. Tests should be run two or three times.

Once an optimum setting is found, you can have hdparm make the setting at boot time by including the relevant commands in `/etc/rc.local`

4.3.2. Tuning SCSI disks

4.3.2.1. SCSI Options

In many cases, SCSI support is an add-in card to the system, giving you many options in vendors and card types. Some server motherboards include SCSI chipset and support onboard as well. In addition, there are dedicated RAID cards that implement various RAID levels in hardware, offloading much of the logic from the CPU.

SCSI has three major protocol varieties: SCSI-1, SCSI-2, and SCSI-3. Fortunately, the protocols are backwards and forwards compatible, meaning that a SCSI-3 controller card can talk to a SCSI-1 drive, albeit at a slower speed. For best performance, make sure that the protocols of the controller card and devices match.

On the physical side, cable types range from a DB-25 pin connector on older Apple Macintoshes to 80-pin SCA connectors that not only include the SCSI pins, but power and ID information.

Here's a quick rundown on the various connector types available.

INSERT TABLE OF CONNECTORS

4.3.2.1.1. SCSI RAID

Depending on the RAID level you select, you can optimize a drive array ranging from high performance to high availability. In addition, many hardware RAID cards support standby drives, so if one drive in the RAID fails, a previously-unused drive immediately fills in and becomes part of the RAID.

RAID 0 is also known as striping, where each drive in the array is represented together as one large drive. It is very high I/O performance, since data is written sequentially across each drive. In RAID 0, no data space is lost, but if one of the drives dies, the entire array is lost. Drives in the array can be of varying sizes.

RAID 1 is known as mirroring. Each drive mirrors its contents to another drive in the array. Performance is no slower than and drive in the array, but if one drive in the array fails its mirror immediately takes over with no loss. RAID 1 has the highest overhead of drive space, requiring two drives for the storage of one. Each drive pair must be the same size.

RAID 5 is one of the most popular RAID formats. In this setup, data is spread amongst at least three drives along with parity data. In the event of any one drive failing, the remaining drives still have all the data from all drives and can continue without loss of data. If a second drive were to fail, the entire array would fail. Drive space loss comes to N-1 where N is the number of drives in the array. So in a 5 drive array, you would get the storage of 4 drives. All drives in this array must be the same size. Since parity data has to be created and written across multiple drives, the speed of RAID 5 is slow, but its reliability is quite high. For those implementing high availablity on their own, this is the best balance of speed and reliability.

There are other RAID levels available, and in some cases these are vendor-specific (such as RAID 51, which is a mirrored RAID 5 array). For best file performance, RAID 0 will be the fastest. Configuration of each of these levels usually happens outside of Linux, in the card's BIOS, and is specific to each hardware RAID card.

For performance reasons, it is not recommended that you use software RAID. Instead of offloading the RAID functionality to a hardware card and offload the functions from the CPU, software RAID uses the main CPU to create and maintain the RAID. This eats up valuable I/O cycles.

4.3.2.1.2. Tuning SCSI Drives

Since SCSI drivers are written to negotiate the best connection between SCSI controller and its drives, there is very little from the OS that is needed to improve your performance. Your best idea for performance is to have the latest kernel and drivers and to use the correct driver for your card. In some cases, there are both Linux and BSD versions of drivers for your card. Some report that the BSD drivers are faster, so if you have both drivers available, test each out to see which gives better performance.

On the hardware side, make sure that you are using the best match of SCSI controller and drives. Your best performance will probably be seen with Fibre Channel controllers and drives, since they have the highest throughput. If you have a large number of drives, you can span them across multiple controllers. Linux supports up to 8 or 16 cards each with the ability to hold up to 16 devices per card, not counting the card itself.

One way of tuning the SCSI bus is to make sure it is properly terminated. Without proper termination, the SCSI bus may ratchet its speed down, or fail altogether. Termination should occur at both ends of a physical SCSI chain, but most SCSI chipsets include internal termination for their end. Purchase the correct termination for your cable and put it at the far end of the cable. This will make sure there is no signal reflections anywhere in the cable that can cause interference.

4.3.3. Tuning Filesystems

Fast hard drive access is only half of getting fast read and write access. The other half is getting the data from the hard drive, through kernel, and to the application. This is the function of the filesystem that the data lives on.

The first Linux filesystems (xiafs, minixfs, extfs) were all designed to give some level of performance to Linux while giving features close to what you would expect from a regular UNIX operating system. Over time, second extended filesystem (ext2fs) arrived and was the standard filesystem for Linux for many years. Now, the number of native filesystems for Linux include ext3, reiserfs, GFS, Coda, XFS, JFS, and Intermezzo.

Such features that are available for Linux include inodes, directories, files, and tools to modify and create the filesystem. Inodes are the basic building blocks of the filesystem, creating pointers to file data, entries in directories, and directories themselves. The directory of inodes is kept in the superblock, and these superblocks are duplicated many times through the filesystem so if one superblock gets corrupted, another can be used to recover the missing data.

In the event the OS shuts down without unmounting its filesystems, a filesystem check must be run in order to make sure the inodes point to the data it should be pointing at. A journalling filesystem (ext3, reiserfs, JFS, XFS) ensures that all writes to a filesystem are finished before reporting success to the OS.

Each type of filesystem has its advantages and disadvantages. A filesystem like ext2 or ext3 are better tuned to large files, so access for reads and writes happen very quickly. But if there are a large amount of small files in a directory, its performance starts to suffer. A filesystem like reiserfs is better tunes to smaller files, but increases overhead for larger files.

For applications that are writing or reading the hard drive, block sizes will allow Linux to write larger blocks of data to the hard drive in one operation. For example, a block size of 64k will try to write to the hard drive in 64kb chunks. Depending on the hard drive and interface, larger block results in better performance. If the block size is not set properly, it can result in poorer performance. If the optimal block size is 64k, but is set for 32k, it would take two operations to write the block to the hard drive. If it is set to 96kb, then it would take OS will wait for a timeout period or the rest of the block size to fill up before it writes the data to disk, dropping the latency of writing data to the disk.

Block sizes are usually reserved for operations where raw data is being written to or read from the hard drive. But applications like dd can use varying size block sizes when writing data to the drive, allowing you to tune various block sizes.

Chapter 5. Network Tuning

5.1. Introduction to network tuning

Mosix, Beowulf, and other clustering technologies for Linux all depend on an effective network layer. Any delays in getting data from the application to the physical layer would cause even greater delays in the time to complete the task. Much work has been done to the Linux network stack to reduce this time, known as latency, and increase the amount of data that can be sent over a period of time, known as bandwidth.

Latency is important as under high loads, the delay of getting data through the network layer will slow down the number of requests that can be handled at once. Bandwidth is important since you want services like NFS to be able to saturate its network link with data. This means that should all the services of a file server be needed, it can be provided to as many clients as request it.

This chapter will start with an explination of the available networking technologies, then get into how some of these technologies can be tuned for both latency and bandwidth improvements. We will then take a look at some of the standard TCP/IP applications and some tuning hints or gotchas.

5.2. Network Technologies Overview

As is the case in every other section of this book, the faster or better the technology, the more expensive it is. But this should not stop you from cobbling together older technology to make it a bit better than it is at face value.

The first example of this is Beowulf. The original design goal of Beowulf was to take existing, low cost technology and create a supercomputer out of it. To do this, machines were networked together using Ethernet running at 10 megabit. Since this is too slow to have good communication between the nodes, the team did something different - bonded two 10 megabit cards together to create one 20 megabit connection. This can be done in software, and doubled the performance of a system that would have cost too much to upgrade to 100 megabit connection per machine.

These days, 100 megabit connections are standard for most machines, with the cost of these cards being under $75US per card. Much like the cards of a few years ago had faster counterparts, so do today's cards. Gigabit Ethernet over copper or fiber is available for those who want to spend a few hundred dollars. If you want to get really outrageous and spend a few thousand dollars, you can get a proprietary, high speed (2GB), low latency network technology called Myrinet. Myrinet is really designed for clusters, mostly due to the high cost per server, and dropping down from this networking technology to standard Ethernet would lose many of the Myrinet advantages.

Standard Ethernet works with a card, a hub, switch or router, and a cable to connect it all together. The card acts as an interface between the system and the Ethernet bus, and there are a variety of cards available, each with varying interface chips, memory, boot roms, and prices. To explain the difference between hubs, switches, and routers, we have to take a look at Ethernet, TCP/IP, and layering.

When a packet of data leaves a web server to talk to the web browser, it is a TCP/IP packet. That TCP/IP packet is encapsulated within an Ethernet packet. An Ethernet packet is destined only for the local network, and the TCP/IP packet will be re-encapsulated if it has to go on another network. The TCP/IP packet contains the source and destination TCP/IP addresses for the packet, while the Ethernet packet contains the source and destination addresses for only the local network.

A hub is an unintelligent device in that if one port of the hub gets a packet, all other ports on the hub will receive that packet. Due to this, most hubs are very inexpensive, but have poor performance, especially as the load increases. A switch, on the other hand, knows what Ethernet devices exist on each port and can quickly route Ethernet traffic between ports. If a packet comes in on port one, and the switch knows the packet is destined to a machine on port twelve, the switch will send the packet only to port twelve. None of the other ports will see the packet. As a result, the cost for a switch is higher, but allows greater bandwidth for devices connected to the switch.

Both hubs and switches work on the Ethernet layer, so they only look at the Ethernet wrapper for the packet and work only on packets that are in the local network. When you want to connect two networks together, or connect to the Internet, you require a router. The router opens the Ethernet envelope, then takes a look at the source and destination of the TCP/IP packet and sends the packet to the appropriate interface. The nice thing about routers is that the interfaces on it does not have to be Ethernet. Many routers combine an Ethernet port and T1 CSU/DSU, or Ethernet and Fiber, and so on. The router re-addresses the TCP/IP packet to go from one phyical media to another. Routers usually have to have a good bit of brains in them to handle the packet re-writing, dynamic routing, and also requires things like SNMP management and some interface for the user to talk to it. Thus, routers will be more expensive than hubs or switches. Linux can act as a router as well, also tying in technologies like IP Masquerading and packet filtering. In some cases, a dedicated Linux box can perform routing less expensively than a standalone router, since most of the “brains” of being a router is already in Linux.

5.3. Networking with Ethernet

Ethernet is the standard networking environment for Linux. It was one of the first to be implemented in Linux, the other being AX.25 (Ham Radio networking). Ethernet has been a standard for over twenty years, and the protocol is fairly easy to implement, interface cards are inexpensive, cabling is easy to do, and the other glue (hubs and switches) that holds the network together is also inexpensive.

Linux can run a number of protocols natively over Ethernet, including SMB, TCP/IP, AppleTalk, and DECnet. By far the most popular is TCP/IP, since that is the standard protocol for UNIX machines and the Internet.

Tuning Ethernet to work with your particular hardware, as always, depends on the interface cards, cabling options, and switching gear you use. Many of the tools that Linux uses are standard across most networking equipment. Most cards available today for Linux are PCI-based, so configuration of cards through setting IRQ and base addresses is not required - the driver will probe and use the configuration set by the motherboard. If you do need to force IRQ or io settings, you can do that when loading the module or booting the system by passing the arguments irq=x and io=y.

Most of tuning Ethernet for Linux is to get the best physical connection between the interface card and the switching gear. Best performance comes from a switch, and switches have a few extra tricks up their sleeve. One of these is full and half duplexing. In a half-duplex setup, the card talks to the switch, then the switch talks to the card, and so on. Only one of the two ends can be talking at the same time. In full duplex mode, both card and switch can have data on the wire at the same time. Since there's different lines for transmit and receive in Ethernet, there's less congestion on the wire. This is a great solution for high bandwidth devices that have a lot of data coming in and out, such as a file server, since it can be sending files while receiving requests.

Normally, the Ethernet card and switching gear negotiate a connection speed and duplex option using MII, the Media Independent Interface. Forcing a change can be used for debugging issues, or for getting higher performance. The **mii-tool** utility can show or set the speed and duplexing options for your Ethernet links.

mii-tool [-v, --verbose] [-V, --version] [-R, --reset] [-r, --restart] [-w, --watch] [-l, --log] [-A, --advertise=*media*] [-F, --force=*media*] [*interface*]

Table 5-1. mii-tool Media Types

media	100Mbit	10Mbit	Full Duplex	Half Duplex
100baseTx-HD	X			X
100baseTx-FD	X		X	
10baseT-HD		X		X
10baseT-FD		X	X	

> **Note:** Also available is "100baseT4", which is an earlier form of 100Mbit that uses four pairs of wires whereas 100BaseTx uses two pairs of wires. This protocol is not available for most modern Ethernet cards.

The most common use of **mii-tool** is to change the setting of the media interface from half to full duplex. Most non-intelligent hubs and switches will try to negotiate half duplex by default, and intelligent switches can be set to negotiate full duplex through some configuration options.

To set an already-existing connection to 100 Mbit, full duplex:

```
# mii-tool -F 100baseTx-FD eth0
```

```
# mii-tool eth0
eth0: 100 Mbit, full duplex, link ok
#
```

Another way of doing this is to advertise 100 Mbit, full duplex and 100 Mbit, half duplex:

```
# mii-tool -A 100baseTx-FD,100baseTx-HD eth0
restarting autonegotiation...
#
```

Using autonegotiation will not work with many non-intelligent devices and will cause you to drop back to 100baseTx-HD (half duplex). Use force when the gear you are talking to is not managed, and use autonegotiate if the gear is managed.

> **Note:** This program only works with chipsets and drivers that support MII. The Intel eepro100 cards implement this, but others may not. If your driver does not support MII, you may need to force the setting at boot time when the driver is loaded.

5.4. Tuning TCP/IP performance

Setting the Maximum Transmission Unit (MTU) of a network interface can be used to tune performance over a TCP/IP link. The MTU is used to set the maximum size of a packet that goes out on the wire. If data is set to go out that is larger than the MTU, the packet is broken up into smaller packets. This can take up some processing time to create the Ethernet packets, and decreases bandwidth. Ethernet has a set number of bytes it adds on to a packet, no matter the size. Larger packets will have a smaller percentage of overhead used up by the Ethernet header. On the other hand, smaller packets is better for latency, since TCP/IP will wait for the MTU to be filled, or a timeout to occur before sending a packet of data. In the event of an interactive TCP/IP connection (such as telnet or ssh), the user does not want to wait long for their packet to make it from their machine to the remote machine. Smaller MTUs make sure the packet size is met earlier and the packet goes out quickly.

In addition, MTUs also have to fit into the size of the medium the packet is running over. Ethernet has a maximum packet size of ??WHATISIT??, counting the Ethernet header packets. Asynchronous Transfer Mode, or ATM, has a very small MTU, on the order of a few bytes. By default, Ethernet TCP/IP connections have a MTU of 1500 bytes. The MTU can be set using **ifconfig**:

```
# ifconfig eth0 mtu 1500
```

It is recommended to leave the MTU at the maximum number, since almost all non-interactive TCP/IP applications will transfer more than 1500 bytes per session, and a bit of latency for interactive applications is more an annoyance than an actual performance bottleneck.

When using Domain Name Servers (DNS), you may run into cases where DNS resolution is a performance bottleneck. We will get into this more in Section 7.5, but some applications recommend for best performance to log the raw TCP/IP addresses that come in and do not try to resolve it to a name. For security reasons, you may want to change this so you can quickly find out what machine is trying to break into your web server. This decision is left to you, the administrator, as part of the never-ending balance between performance and security. A potential fix for this is to run a caching name server locally to store often-used TCP/IP addresses and name, and leave the real DNS serving to another machine.

Applications like **ping** will sometimes appear to fail if DNS is not configured properly, even if you try to ping a TCP/IP address instead of using the name. The solution to this and other TCP/IP management applications, is to find the option that prevents resolution of names or TCP/IP addresses. For **ping**, this is to give the -n option.

```
# ping -n 192.168.1.50
```

5.5. Tuning Linux dialup

If your connection to the rest of the world is through a dialup link, don't worry. Section 5.4 covers many of the issues that you will see for dialup. The MTU for PPP is recommended at 1500, but slower links may want an MTU of 296 to have improved latency.

Modems in Linux should be full fledged modems, not ones labeled WinModem or Soft Modems. Each of these styles of modem pass much of the processing work off to the CPU. This makes the cards very inexpensive, but increases the load on the CPU. Modems can be internal or external, but internal modems include their own port settings that may be easier to use than with external modems, since the modem and serial port are in the same card. If you use external modems, set the data link between the serial port and the modem to be the highest the modem supports, which is usually 115,200bps or 230,400bps. This ensures that the modem can talk as fast as it needs to with the machine. You should also ensure that you are using RTS/CTS handshaking, also known as hardware, or out-of-band flow control. This allows a modem to immediately tell Linux to stop sending data to the modem, preventing loss of data.

Controlling the Linux serial port is used with the **setserial** command. You can find the available serial ports at bootup, or by looking at `/proc/tty/driver/serial`. Entries in that file that have a UART listed exist in Linux. Remember that COM1 or Serial 1 listed on your box will be listed as `/dev/ttyS0`, and COM2 is `/dev/ttyS1`. Most modern applications can comprehend speeds greater than 38,400bps, but some older ones do not. To compensate for this, Linux has made 38,400bps (or 38.4kbps) a "magic" speed, and if an application asks for 38.4kbps, Linux will translate this to another speed. Currently, this can be as high as 460kbps. To use this, the **setserial** command is used.

```
# setserial /dev/ttyS0
/dev/ttyS0, UART: 16550A, Port: 0x03f8, IRQ: 4
# setserial /dev/ttyS0 spd_vhi
# setserial /dev/ttyS0
/dev/ttyS0, UART: 16550A, Port: 0x03f8, IRQ: 4, Flags: spd_vhi
```

Speeds are listed as codes for setserial, and these codes are listed in Table 5-2. These values can be set by non-root users.

Table 5-2. Settings for setserial

Code	Speed
spd_normal	38.4kbps
spd_hi	57.6kbps
spd_vhi	115.2kbps
spd_shi	230.4kbps
spd_warp	460.8kbps

There is also a spd_cust entry that can use a custom speed instead of 38.4kbps. The resulting speed becomes the value of baud_base divided by the value of divisor.

If both sides have it available, compression using Deflate or BSD is available and can increse throughput. Even though many newer modem protocols provide compression, it isn't very strong. By using Deflate or BSD, greater compression at little loss of CPU or memory space can be attained. The down side is that both sides need to have either Deflate or BSD compression available built into the PPP software. BSD compression can be activated by using the `bsdcomp` option passed to **pppd** followed by a compression level between 9 and 15. Higher numbers indicate higher compression. Deflate compression can be used with the `deflate` option followed by a number in the range of 8 to 15. Deflate is preferred by the pppd used by Linux.

5.6. Wireless Ethernet

Wireless Ethernet, also known as IEEE 802.11b, is becoming more popular as the cost to implement decreases and availability of more products increase. The Apple AirPort and Lucent Orinoco cards have brought wireless into the home market, allowing a person to have Ethernet access anywhere in their house, and schools are deploying Wireless Ethernet across the campus. It's now available at airports, schools, many high tech companies, and soon upscale coffee shops.

Given that 802.11b is most popular for laptops, since they are portable, tuning for performance is not as great importance as tuning for power usage. Using 802.11b is often very power-consuming and can quickly drain the batteries. Some cards (such as the Lucent Orinoco card) have the ability to turn its antenna on and off at regular intervals. Instead of the antenna being on all the time, it turns on a few times a second. With the transmitter being turned off now more than half the time, the battery usage is decreased. There is an increase of latency and decrease in data rate.

To set the power management of the card, you will need to have the wireless tools, available with many distributions. This package contains three commands for managing your wireless card: iwconfig, iwspy, and iwpriv.

The **iwconfig** is an extension of **ifconfig**. Run without any options, it will check all available interfaces and checks for wireless extensions. If there are any, it will report similar to the following:

```
wvlan0    IEEE 802.11-DS  ESSID:"default"  Nickname:"HERMES I"
          Frequency:2.437GHz  Sensitivity:1/3  Mode:Managed
          Access Point: 00:90:4B:08:13:1C
          Bit Rate:2Mb/s   RTS thr:off   Fragment thr:off
          Power Management:off
          Link quality:8/92  Signal level:-88 dBm  Noise level:-96 dBm
          Rx invalid nwid:0  invalid crypt:0 invalid misc:599
```

As you can see, the output tells you a variety of statistics on the link. My current bit rate is 2Mb/s, probably because my link quality is so low. The link quality is 8 out of 92, indicating that I should either move my laptop, move my base statiopn, or throw out my 2.4Ghz phone.

You can also see that power management is currently off. Since my laptop is plugged into the wall, this is not a concern to me. If I did want to activate the power management, I would use:

```
# iwconfig wvlan0 power 1
# iwconfig
iwconfig
lo        no wireless extensions.

wvlan0    IEEE 802.11-DS  ESSID:"default"  Nickname:"HERMES I"
          Frequency:2.437GHz  Sensitivity:1/3  Mode:Managed
          Access Point: 00:90:4B:08:13:1C
          Bit Rate:2Mb/s   RTS thr:off   Fragment thr:off
          Encryption key:off
          Power Management period:1s   mode:All packets received
          Link quality:11/92  Signal level:-84 dBm  Noise level:-95 dBm
          Rx invalid nwid:0  invalid crypt:0  invalid misc:0
#
```

The power management is now set to turn the transmitter on only once per second. By default, the time is in seconds, but but appending m or u to the end of the number will make it milliseconds or microseconds.

All that being said, here are a few ways in improve the link quality of your system. Any combination of these will work, so do not expect one method alone to work.

- Check the infrastructure and building materials. Thick wood or metal walls will cause a lot of interference. Line of sight to the base station is best.
- Some base stations and wireless cards support external antennas. They will greatly improve the range and quality of the link.
- Move the base station around. Line of sight is best, but not required.
- Turn off other devices that use 2.4Ghz. Some phones and other wireless gadgets use the same frequency, and if not built properly, will cause the wireless Ethernet cards to continually scan through frequencies for the correct one, dropping performance.

5.7. Monitoring Network Performance

The best way to make sure your network is not the bottleneck is to monitor how much traffic is flowing. Because of collision detection and avoidance in Ethernet, once the load gets above about 50% to 60% of its maximum, you will start to see degrading performance if using hubs. This number if higher for switches, but still exists since the silicon on the switch needs to analyze and move the data around.

To make best use of your networking equipment, you will want to monitor the amount of traffic that is flowing through the network. The easiest way to do this is to use SNMP, or Simple Network Management Protocol. SNMP was designed to manage and monitor machines via the network, be it servers, desktops, or other network devices such as switches or network storage. As you would guess, there are SNMP clients and servers avilable for Linux to monitor the statistics and usage of network interfaces.

SNMP uses an MIB or Management Information Base to keep track of the features of an SNMP device. While a Linux box can have things like monitoring the number of users logged in, a Cisco router will not need these functions. So the MIBs are used to identify devices and their particular features.

The SNMP daemon for Linux is net-snmp, formerly known as usd-snmp, and based on the cmu SNMP package. Your distribution should be mostly configured. The only thing you need to do is set the community name, which is really just the password to access the snmpd server. By default, the community name is “private”, but should be changed to something else. You will also want to change the security such that you have readonly access to snmpd.

```
#         sec.name  source          community
com2sec   paranoid  default         public
#com2sec  readonly  default         public
#com2sec  readwrite default         private
```

Change the “paranoid” above to read “readonly” and restart snmpd.

> **Note:** This setting will give readonly access to the entire world to your SNMP server. While a malicious intruder will not be able to change data on your machine, it can give them plenty of information about your box to find a weakness and exploit it. You can change the “source” entry to a machine name, a network address. Default means any machine can access snmpd.

You can test that snmpd is working properly by using **snmpwalk** to query **snmpd**.

snmpwalk {*host*} {*community*} [*start point*]

```
$ snmpwalk 192.168.1.175 public system.sysDescr.0
system.sysDescr.0 = Linux clint 2.2.18 #1 Mon Dec 18 11:23:05 EST 2000 i686
$
```

Since this example uses system.sysDescr.0 as its start point, there is only one entry that gets listed, that of the output of **uname**.

5.7.1. Network Monitoring with MRTG

The most popular application for monitoring network traffic is MRTG (http://www.mrtg.org/), the Multi Router Traffic Grapher. MRTG tracks and graphs network usage in graphs ranging from the last 24 hours to a year, all on a web page. MRTG uses SNMP to fetch information from routers. You can also track individual servers for ingoing and outgoing traffic.

> **Note:** The process of monitoring a server using SNMP will consume a small portion of network, memory, and CPU.

MRTG is available for Red Hat and Debian distributions. You can also download the source from the MRTG home page. Once installed, you will need to configure MRTG to point to the servers or routers you wish to monitor. You can do this with **cfgmaker**. The options to cfgmaker have to include the machine and community name that you want to monitor.

```
mkomarinski@clint:~$ cfgmaker public@localhost
--base: Get Device Info on public@localhost
--base: Vendor Id:
--base: Populating confcache
--base: Get Interface Info
--base: Walking ifIndex
--base: Walking ifType
--base: Walking ifSpeed
--base: Walking ifAdminStatus
--base: Walking ifOperStatus
# Created by
# /usr/bin/cfgmaker public@localhost


### Global Config Options

#  for Debian
WorkDir: /var/www/mrtg

#  or for NT
# WorkDir: c:\mrtgdata

### Global Defaults

#  to get bits instead of bytes and graphs growing to the right
# Options[_]: growright, bits


######################################################################
# System: clint
```

```
# Description: Linux clint 2.2.18 #1 Mon Dec 18 11:23:05 EST 2000 i686
# Contact: mkomarinski@valinux.com
# Location: Laptop (various locations)
######################################################################
### Interface 3 >> Descr: 'wvlan0' | Name: '' | Ip: '192.168.1.175' | Eth:
'00-02-2d-08-ae-c1' ###

Target[localhost_3]: 3:public@localhost
MaxBytes[localhost_3]: 1250000
Title[localhost_3]: Traffic Analysis for 3 -- clint
PageTop[localhost_3]: <H1>Traffic Analysis for 3 -- clint</H1>
 <TABLE>
   <TR><TD>System:</TD>     <TD>clint in Laptop (various
locations)</TD></TR>
   <TR><TD>Maintainer:</TD>
<TD>mkomarinski@valinux.com</TD></TR>
   <TR><TD>Description:</TD><TD>wvlan0
</TD></TR>
   <TR><TD>ifType:</TD>     <TD>ethernetCsmacd
(6)</TD></TR>
   <TR><TD>ifName:<TD>     <TD></TD></TR>
   <TR><TD>Max Speed:</TD>  <TD>1250.0
kBytes/s</TD></TR>
   <TR><TD>Ip:</TD>         <TD>192.168.1.175
()</TD></TR>
 </TABLE>
```

All the configuration information has been pulled from snmpd. You can redirect the output of **cfgmaker** into `/etc/mrtg.cfg`.

Most installations of mrtg will include a cron process to run **mrtg** if `/etc/mrtg.cfg` exists every five minutes. Within five minutes, you will see data on your web site.

Chapter 6. Kernel Tuning

Unlike most other UNIX or UNIX-like operating systems, the source code to the kernel is available for easy tuning and upgrading. With kernels being packaged in distributions, the necessity for most users to compile their own kernels is pretty low. There are advantages to compiling your own kernels, though:

- Compile a version of the kernel specific to your chipset.
- Add in a new driver or security patch
- Compile a kernel with modules compiled into the kernel

This chapter covers each of these situations, and also tuning of running kernels using applications like sysconf and powertweak.

6.1. Kernel versions

There are a few things to consider when deciding what kernel revision you should be using in a production environment.

Officially, Linux has two major threads of development. The stable releases have an even number as its minor number. The Linux 2.2 and 2.4 kernels are examples of stable releases. These are considered by Linus to be stable enough for production use. Few known bugs or issues exist with these kernels.

The other release is the development releases. These have experimental drivers or changes to them, and can cause crashes if used in production environments. However, these releases will be more likely to support newer hardware.

Most Linux distributions will often make a compromise between stability and performance by creating their own version of the kernel and source code. These kernels start with a base of a stable kernel, then add in some newer drivers. The resulting kernel is then tested to make sure it is stable, then released.

As a generalization, stable kernels are good for production use, while development kernels may have better performance. For your use, you should monitor the changes in the two sets of kernels and see if there are any performance increases that will make your system faster, then test that against a known stable kernel. You can then make a determination of stability of the system versus performance.

6.2. Compiling the Linux Kernel

There is a specific order to getting your kernel compiled, and there is nothing preventing you from having multiple kernels and modules on the same system. This allows you to boot test kernels while

keeping a known stable kernel in case you have trouble. Boot loaders like LILO will allow you to select a kernel at bootup.

The easiest place to pick up the latest kernels is http://www.kernel.org. This is the official site for Linux kernels. These kernels are almost always in tar-bzip2 format, instead of being in RPM or DEB formats. Check with your distribution provider for packaged formats of the kernel.

Unpack and untar the kernel. For packages compressed with bzip2, the process is:

```
# tar -jxvf linux-2.2.19.tar.bz2
```

This will create all the header files and source code for the kernel. There are three methods for configuring the kernel to compile, and each have some prerequisites. All methods will obviously require the C compiler.

6.2.1. Using make config

The siplest way (and the first) is to use **make config** from within `/usr/src/linux`. This is a very tedious way of compiling, since you must answer each question and the Linux kernel has a large number of questions now.

But an advantage to **make config** is that it does not require any other development tools to be installed in order to use it. On very lightweight systems, using this method will save drive space and memory, since we are not running very complicated or large programs.

SCREENSHOT OF make config HERE

6.2.2. Using make menuconfig

The next level of kernel configuration involves using a character based menuing system. For systems that have only a text console or ssh access, this makes an easier way to configure a kernel. It requires both ncurses and its development libraries to be installed in order for this to work. The program that runs in **make menuconfig** is part of the kernel source package and is compiled when needed.

With an on-screen menu, it is easier to organize and select the options you want to compile in. Help menus are available as well. This option is not good for those who want small systems, since it requires extra libraries.

INSERT SCREENSHOT OF make menuconfig

6.2.3. Using make xconfig

The most user friendly, but hardest to use, is the **make xconfig** option for configuring the kernel. This requires most of the X libraries to be installed, along with Tcl and Tk.

Since many server systems may not have X or its associated libraries, Tcl, or Tk installed, you may want to step back to one of the other methods of configuring the kernel. You would also need to make sure that you can export X to your desktop if you are configuring a remote system.

Figure 6-1. make xconfig

6.2.4. Options for configuring the kernel

Now that you have chosen a way to configure the kernel, how should you go about your configuration? What options should you pick for best performance, lowest memory consumption, and best security options?

Many of the answers to these questions depends on your particular situation. Selecting the right processor family for your CPI is one of the best things to do. Each processor family has its own way of alinging memory and some CPU-specific options and optimizations.

If your hardware setup is going to be stable, you can compile in drivers for various kinds of hardware for slightly better performance, and to prevent trojan modules from being loaded. See Section 6.2.4 for more on this.

Selecting DMA access for IDE devices under “Block devices” will automatically turn on DMA access, reducing CPU load. You can see Section 4.3.1.2 for more information.

If your box will be configured as a router without firewalling, you can enable “IP: optimize as router not host” under “Networking options”. This reduces the latency of packets within the kernel by skipping a few unnecessary steps. But these steps are required if you will be doing firewalling or acting as a server.

Bonding device support under “Network device support” will enable bonding multiple network devices together to create one large pipe to other bonded Linux boxes or some Cisco routers.

If you plan on having a large number of users logging in and you would need a large number of pseudo terminals (PTYs), you can enter a number in “Maximum number of Unix98 PTYs in use (0-2048)” under “Character devices”.

For systems that will not be onsite and require high availability, you can enable “watchdog timer support”. Watchdogs can be implemented either in software or hardware. Software watchdogs open and close files to make sure that everything is working. If the driver cannot open the file, it tries to trigger a reboot of the system, since this indicates that there is some error with the system. Watchdog timers are usually implemented ISA cards, but some single board computers will have the necessary chips built in. Hardware watchdogs are a bit better as it can monitor extra features such as temperature and fan speed and can trigger reboots on those events as well. Hardware watchdogs are a bit more proactive as they continually talk to the kernel driver. If the kernel driver does not respond, the watchdog triggers a reset.

6.2.5. Kernel Modules and You

While this book is not specifically covering security, it is fair to bring it up every now and then. For those who want hard core, as-uncrackable-as-possible systems, you may want to consider removing module support from your system.

The reason for this is that if a cracker were to gain access to your system, one of the more advanced methods of covering their tracks is to insert a kernel module that hides much of their activity.

Another tactic is to replace an existing module with a trojan horse. Once the module is loaded, its payload deploys in kernel space instead of userspace, giving the rogue code more access to the system.

Another reason to remove modules from your system is to create a smaller size kernel. With modules compiled into the kernel, they get compressed along with the rest of the kernel using bzip2. Normally module files are uncompressed, but bzip2 can compress sizes by more than half. This can save crucial space if it is required.

For most users, however, you will want to use modules, as it gives a very versitile way of installing new drivers by just loading the module.

6.2.6. Building The Kernel

With a configured kernel, you can now go through the build process. This is documented in a number of places, but it is included here for completeness.

The first thing you should take a look at if you want to store the kernel configuration is to get `/usr/src/linux/.config`, as it has the ouput from the configuration options selected previously. You can move the .config file to a new system, then build the kernel and skip the configuration part.

The **make dep** command sets up the dependencies needed so the kernel can be built. This is a crucial step, as the kernel will not compile without it.

Most users will want to use bzip2 for building the kernel, as it makes a much smaller kernel than gzip or uncompressed. You can build a bzip2 kernel by using **make bzImage**. The compressed image is built and put into `/usr/src/linux/arch/i386/`. You may want to just use **make bzlilo** to put the kernel into `/` and run **lilo** to let you boot the new kernel. Some users may not want their kernel in `/`, since many distributions create a small `/boot` partition that contains the kernel and enough to boot the kernel. In this event, you will want to copy the kernel by and into `/boot`.

If you will be using modules, you will also need to run **make modules** and **make modules_install**. These two commands will build and install the modules for your kernel.

6.3. Tuning a Runing Linux System

When Linux first got started, changing options to the kernel often required recompiling. Getting information out of Linux required applications be recompiled to look at the right memory location to get things like user and process lists. After the release of the 1.0 kernel, the `/proc` filesystem arrived as a way to access information in a kernel-independent way. Because of this, a command like **ps** will keep operating even if the kernel rev changes. The `/proc` filesystem also allows you to change the kernel itself wile running, allowing you to add in features like TCP/IP forwarding, manage RAID cards, and so on.

6.3.1. The `/proc` filesystem

This portion is not strictly about tuning a system, but getting information out of your urnning system. In the event of serious memory or hardware issues, you can sometimes get information out of Linux.

The `/proc` heirarchy may change depending on the kernel revision, but is generally arranged into different classes.

Processes

Contains information about each of the running processes, organized by PID. Each process has its own directory of information, including status, list of file descriptors, command line, environment variables for the application, and working directory when the application was started.

bus

Contains information about the system busses. At this point, it works with the USB, PCI, and PCMCIA busses. More, like IEEE-1394 will list devices and controllers under this directory. The information about each device is less human readable than older directoried like `/proc/pci` , but human readability is not the point of `/proc`

cpuinfo

If you want to find out information about the CPU(s) installed in your system, you can take a look at the cpuinfo file. For each processor, you will get things like CPU speed, vendor, model type

(Pentium II, Pentium III, etc.), and the ever-popular bogomips rating.

filesystems

This file has a list of the filesystems that the kernel currently knows about. We say currently, since modules can be loaded or removed to add or remove additional filesystem types. If you are having trouble mounting a filesystem, this is an easy way to make sure the filesystem type is supported by the kernel.

ide

If your system has IDE devices, this directory contains information about the IDE devices that are in your system. It is organized by IDE channels that are available, and each device has informaiton about the model name and number, serial number, capacity, and any other information specific to that IDE type. Tuning of IDE drives should be done using **hdparm**, and you can see Section 4.3.1.2 for usage.

meminfo

This file has the raw memory information that you would normally see as output of **free**. The first part of the ouput contains the memory listing in bytes, followed by most of the same information in kilobytes.

modules

You could easily mistake the contents of the modules file for the output of **lsmod**. There is a listing of the modules, size of the module, and dependencies for the module.

partitions

This file has a list of all the disk partitions that are known to Linux, including those not mounted or listed in `/etc/fstab`.

pci

This file may go away in the long term, in favor of `/proc/bus/pci`, but `/proc/pci` is more human readable, listing the device type, vendor, and device name. But this will only happen if the Linux kernel knows of the device.

scsi

SCSI-based systems can look at and tune their parameters here. Its setup if much the same as for IDE, listing chains, followed by devices on each chain.

swaps

Lists all mounted swap partitions (or files) mounted.

version

The version file contains version information of the kernel you are using. It's very helpful in that it has the user and host that the kernel was compiled on, GCC version used to compile the kernel, and date and time this kernel was compiled. This makes it much more helpful than **uname**, since you can quickly check a series of running machines and verify they all report the same kernel revision.

Not listed above is `/proc/sys`. This directory is where most of the runtime changes to the kernel take place. Access to this used to be done manually, using **cat** and **echo** to view or change settings of a running kernel. If you wanted runtime settings to be changed each time the machine booted, you had to add commands to the `rc.local` file for each setting.

An easier way to manage these settings is via the use of **sysctl**.

sysctl [-n] [*variable*] [-w *variable*=*value*] [-a] [-p *filename*]

If you use an argument of *variable*, you will get the current value of `/proc/sys/`*variable*. You can set it by adding an equal sign and *value*. You can get a list of what is available along with their current values by using -a.

As part of the Linux boot sequence on more modern distributions, **sysctl** is called with -p, which reads settings from `/etc/sysctl.conf`. This file can be edited by a text file, and contains a list of lines of the form *variable*=*value*.

```
#
# /etc/sysctl.conf - Configuration file for setting system variables
# See sysctl.conf (5) for information.
#
#kernel.domainname = example.com
#net/ipv4/icmp_echo_ignore_broadcasts=1
net.ipv4.conf.all.hidden = 1
net.ipv4.conf.lo.hidden=1
```

As you can see, *variable* can separate out directories using either a slash (/), or a dot (.). Using the dot makes it look more like SNMP variables.

Since everything under `/proc` are regular files, security for viewing or setting are handled by regular file permissions. In some cases, only root can make changes, in others, only root can view or set.

6.3.2. Using powertweak

Another interface to change options of a running kernel is through the use of **powertweak**. Powertweak also runs on startup of linux, using a daemon called powertweakd to issue changes to the kernel. Configuration files are stored in the kernel in XML format.

An advantage of using **powertweak** over **sysctl** is that powertweak has a graphical and text interface. There is a pretty extensive help system that can give you information about each setting.

For those that just want to have optimized settings for a particular use, there are pre-made settings for laptops, web servers, routers, and Oracle usage. As an example, the laptop setting changes the kernel

disk flush so the hard drive can spin down. These are just generic settings, as each different machine has different PCI devices that can also be set.

You can get the latest version of powertweak from http://www.powertweak.org/

Figure 6-2. Powertweak

Chapter 7. Application Tuning

While one can tune hardware and the OS and get great measurements from it, most applications for Linux have their own rules for improving performance. Tuning a hard drive for an application that uses a lot of memory will not improve the speed of that application, whereas investing in more memory will create a vast improvement in speed. Let us examine some of the applications for Linux and how you can tune a box specifically for it.

7.1. Squid Proxy Server

Squid proxy server has two common uses. The first is as a proxy cache for internal clients access the external web. The second is as a reverse proxy, caching content from internal web servers.

7.1.1. Squid as a proxy cache

Squid as a proxy cache stores large numbers of web pages, enabling them to be sent to clients upon request instead of requiring another internet fetch. Squid uses large amounts of disk space for the cache, but by nature doesn't access any one that much more frequently than any other.

The first thing to avoid is a disk bottleneck. In a high-usage environment, you will definitely want to distribute the cache file across as many spindles as is practical. Since the cache data is relatively worthless, in that it is flushed out as a matter of course, the additional overhead of a RAID 5 or other relatively fail-safe multi-spindle system is not necessary. RAID 0 striping will provide the fastest access with no fail-safe requirements.

The next thing to avoid is a memory bottleneck. There should ABSOLUTELY be no OS paging with Squid, Squid will page enough on its own. This is a case where Squid has in effect its own virtual memory system, with thousands of pages on disk and tens or hundreds in memory. If the OS is paging to provide the tens or hundreds in memory, your performance will be horrible.

Network performance will probably not be a big deal. If your byte hit rate is 25%, and you are maxing out a T1, you are still only putting around 2Mbps out your Ethernet connection.

7.1.2. Squid as a reverse proxy

As a reverse proxy, the total set of web pages that Squid is proxying is reduced greatly. It must store only the limited number of pages behind it on your web servers, as opposed to the entire internet. Since disk storage requirements are down by an order of magnitude, so are memory requirements.

However, one approach to this situation would be to build up real memory until the entire working set of pages could be stored, with no paging required.

In this case we're trading large store/infrequent access for a smaller store/frequent access. We want to optimize the byte hit rate in Squid with real memory to reduce disk operations. Again, there should be NO OS PAGING. If any DNS lookups are required, it might be a good idea to run a DNS cache locally. Otherwise, this shouldn't be too difficult to tune to get good performance.

7.2. SMTP and Mail Delivery

While this section addresses sendmail directly, most of the concepts should apply to any mail delivery system.

7.2.1. Sendmail

A sendmail system will probably be the most likely to be network bottlenecked, but that bottleneck will also most likely be outside the system itself. There are no special tricks since a typical sendmail system will be fairly balanced against typical hardware. Depending on the size of the mail store, some type of RAID system may be implemented, but the mail store data is very valuable so use a RAID 5 or other fault-tolerant option. If the store does get very large, you might also investigate the benefits of multiple caching layers down to the card.

Sendmail itself is very dependent on DNS lookups for its operation, and may be doing ident lookups as well. As such, some tricks on the network can really help. Linux 2.4 kernels offer a lot of QoS features, these can be used to prioritize DNS and ident lookups over bulk traffic, preventing blocking of POP, IMAP, and SMTP waiting on a DNS lookup. I would also strongly suggest implementing a DNS cache locally, maybe even on the sendmail server itself.

7.2.2. Mail Delivery (POP and IMAP)

Memory and processor requirements will depend on the number of usersand whether you are providing POP or IMAP services, or only SMTP. POP and IMAP require logins, and each login will spawn another POP or IMAP process. If concurrent usage of these services is high, watch for OS paging.

Using POP for delivery has a very low memory and hard drive footprint. In POP, e-mail is typically downloaded from the server to the client. The POP server merely handles authentication and moving the data to the client. When a mail message is sent to the client, it is typically removed from the server.

Using IMAP puts much more stress on the system. IMAP will copy an entire mailbox into memory, and perform sorting and other functions on the mailbox. When a client requests a message, the message is sent to the client, but is not removed from the server. So all mail messages remain on the server until the client explicitly deletes it. The advantage to this is a user can have IMAP clients on multiple machines and still be able to access all the messages. Since most of the processing is offloaded to the server, this makes IMAP a great mechanism for light clients or wireless devices that do not have much memory or processing power.

Since the data on a POP or IMAP server are critical, you should be using RAID-5 or RAID-1 to protect the data and keep the server running. POP has low memory requirements, while IMAP has very high memory and disk requirements, and both will likely increase as more users are introduced to the system and they start working with mailboxes in the thousands of messages.

In either case, using SSL on top of POP or IMAP will also increase the CPU load of the protocols, but will have minimal affect on disk or memory usage.

It is not advisable to put a shell account on the same machine as one running IMAP or POP, due to the CPU and memory used by these protocols. Nor is it advisable to use NFS to export a mailbox to a shell machine, as NFS locks can cause corruption of a mailbox if sendmail tries to deliver a message and the user is deleting another message from the same mailbox.

7.3. MySQL, PostgresSQL, other databases

7.4. Routers and firewalls

7.5. Apache web server

Apache works best when it is delivering static content. But if you're only delivering static content, you should consider using Tux, the Linux kernel HTTP server.

There are a number of small quick-tips you can use to increase the throughput and responsiveness of Apache.

One such tip is to turn off DNS resolving in Apache. When Apache receives a request to send data to a remote server, Apache will request a reverse DNS lookup on that IP address. While the requests winds its way through the DNS heirarchy, Apache is stuck waiting for the answer before it can send the page back

to the requesting client. By turning this feature off, Apache will merely log the IP address without name in its log file, and then deliver the page. A problem with this is if the machine is cracked through the web server, you will have to do your own reverse DNS lookup to find out what country and domain the machine is from. A way to have the best of both worlds is to run a caching-only name server run on the local network or on the web server box itself. This allows Apache to quickly get the DNS information, and store a hostname and domain instead of just an IP address.

Cutting the size of graphic images is also a way to improve performance. Using algorithms such as JPG or PNM will cut the size of the image file without reducing the quality of the image by much. This allows more image files to go out per second.

Instead of reinventing the wheel, see if Apache has a module for the feature you need. Modules for authentication against SMB, NIS, and LDAP servers all exist for Apache, and you will not have to add the functionality to your server side scripts.

Make sure your database is tuned properly. This has been covered in Section 7.3, but a poorly tuned database can slow everything down. If you expect heavy usage, do not put the database and web server on the same machine. One machine running the database, and another running the web server connected by high speed (gigabit Ethernet) will allow both machines to operate quickly.

Since so many web sites are using server-based applications and CGI, it makes sense to take a look at the server interpreters being used. If you use a language like Java that is known to be pretty slow, you can expect lower performance as compared to using applications compiled in C.

But Java has its own advantages over C. The biggest is that Java is designed to be much more portable than C is. Since Java is an object oriented language, its code reuse is much better, allowing delopers to create applications quicky.

There are three other major server languages used by Apache. These are Perl, Python, and PHP. All three are included in most Linux distributions, and if they’re not on your system, each is easy to install and get working.

7.5.1. Perl

Perl is pretty much the granddaddy of CGI and server programs. It has excellent string manipulation abilities, plenty of libraries to let you create web applications quickly, and has drivers for just about anything you want to do. It is known as the programmer’s swiss army knife, and for good reason.

The downside to perl is that it is an interpreted language, meaning each time a perl application is run, the perl interpreter has to be loaded, the application has to be loaded into memory, and the interpreter has to compile the application. This can create a heavy load on a system.

The best way around this is to use the mod_perl library with apache. This library first loads the perl interpreter so it is always in memory and ready to run. It also caches frequently-used perl CGI scripts in memory already in a precompiled state. When a perl application starts up, many of the bottlenecks (loading and interpreting) have already been done. The cost to the sysadmin and programmer is that mod_perl takes up a lot of memory, and its memory use will increase as more perl scripts are added to a web server.

> **Note:** Adding more memory and CPU speed to a machine running mod_perl will improve its performance.

7.5.2. Python

Python was one of the first of the interpreted languages to implement object oriented programming, allowing programmers to reuse and share code. It also learned a bit from perl and it's interpreted nature by comiling python code into a pre-compiled format. This format is not portable like Java, but does much of the grunt work of compiling a python application. The python interpreter then spends less time compiling the application each time it's run.

Programmers may cringe a bit when they start writing in python. Syntax is much more strict than C or Perl, using tabs and newlines to delineate parts of code. And, like perl, python requires the interpreter to load into memory and run.

There is also a mod_python for apache that operates simiar to mod_perl. More memory and CPU speed will be a great improvement.

7.5.3. PHP

PHP is a relative newcomer to the languages listed here. But PHP was designed for server side applications. The interpreter is a module in apache, and PHP code can exist within a HTML page, allowing authors to combine the web page and application into one file.

PHP has two downsides to it from a programer's perspective. First, PHP is still relatively new. It is not quite as seasoned as Perl or Python, each of which have over ten years worth of development behind them. Second, PHP does not have modules that can be easily loaded and used like Perl and Python have. If your implementation of PHP does not have the graphics libraries installes, you will have to recompile PHP and add them in. Neither of these should prevent you from giving a serious look at using PHP for your server side application.

7.6. Samba File Sharing

7.7. Network File System