Linux for Workgroups

Debian-GNU-Linux

Jorge L. deLyra

November, 2004

Contents

Chapter 1

Introduction

1.1 Historical Perspective

As this is written, we, the inhabitants of this little planet, are caught right in the middle of one of the few really important revolutions of our history. It finds comparison only with the other revolutions which changed the very fabric of life and society, by having a profound effect on the means of production and hence on the economic life of societies: the agricultural revolution of some 20000 years ago, which led to the dawn of civilization, and the scientific revolution of the 17^{th} century, which led to the industrial revolution of the 19^{th} century. The current process is a consequence of the second scientific revolution which rocked the science of Physics in the first quarter of the 20^{th} century, when Relativity and Quantum Mechanics were discovered. After more than half a century of development, the technical developments known as *information technology* are hitting us, in the dawn of the third millennium, with the full force of a hurricane.

No aspect of human activity is immune to the effects of this revolution. It has a role to play in every profession, in every endeavor, in every enterprise. In some, it has the potential to effect, or already is effecting, profound and permanent changes that will last for ever. The use of computers and communication networks is now essential to many activities, and will be more so in the future. Room for those who are not information-age literate will dwindle rapidly. A working knowledge of the technology will become part of the standard arsenal that every member of society should possess, just like literacy and knowledge of at least some elementary science and mathematics. Revolution means deep and permanent change and, in the pace of the modern world, fast change. Not all of it will be devoid of pain, and a capacity and willingness to learn and adapt is essential for survival in such circumstances. The information technology revolution is one of the main mechanisms driving the much-talked-about *globalization process* our world is undergoing. This is a profound and irreversible transition in the economic life of the planet which has very serious and widespread social implications.

Computers have been under development since the middle of the last century, and data communication networks appeared not long after that. They have been undergoing development under what is named an *exponential process*. Such a process is one in which the speed of development is proportional to the stage of development, because at any stage of the process the development undergone so far feeds back into de development mechanism and is instrumental in speeding it up. It might start slowly, but it soon picks up speed and eventually becomes frantically fast. It will stop or slow down only when it exhausts the resources it feeds upon. In the case of information technology any such limits for the development process are still far from visible within the current technological horizon, and one cannot predict in any reliable way when it will reach a point of saturation and cease to be an exponential process. All that can be said is that this does not seem to be going to happen anytime soon.

This kind of process is characterized by having a fairly constant characteristic time, the time it takes for the development level to double. It has been measured that all the main elements of information technology, such as processing power, memory capacity,

storage capacity and communication speed and bandwidth, undergo their development according to some such exponential process, with characteristic times of the order of something from one year to a few years, which are very short in historical terms. We can reasonably expect to see in just a few more decades the emergence of computer systems approaching the degree of complexity and connectivity of the human brain. Even before that we are likely to see dramatic increases in the capacity of data networks, that may change significantly the daily routine of our lives.

In the beginning none of this was of much interest to most people because the existing technology had low performance levels and extremely high prices associated to it. The common person could not afford any of it, and had no use for most of it. Only large organizations and government agencies had either the means or the use for such systems. However, the exponential development process has been on without break for more than half a century now and, within the last decade or so, it reached the scale of the common person. It became financially possible to have a significant computer at home, and many good reasons for having it started to appear. The use of computer communication networks followed suit not too long afterwards. Today the average person in developed societies can have at home both access to the Internet, a computer network with global reach, and to a computer many times faster than the multi-million-dollar computers that were only accessible to large universities and government agencies some tree decades ago.

This remarkable process represents an enormous potential enpowerment of the individual and is certainly a positive and desirable thing, but it also poses to everyone non-trivial challenges: the challenges of learning to use such systems; of learning to use them well and to significant ends; of learning to use them in creative and innovative ways. Many people are already doing this, and their efforts have resulted in what became the latest chapter in this profound revolution: the arise of the world of *free software*. The free software movement is a social movement based on information technology, and one with some remarkable characteristics which are unprecedented in recent history: it transcends and ignores the traditional national boundaries; participation in it is completely voluntary and open to anyone; most people involved in the process do not know each other on a personal basis and have never met, since the community is spread throughout the world; the role each one may have in it depends exclusively on technical ability and willingness to contribute their time and effort; the goods resulting from the collaborative effort are distributed openly, freely and without charge to anyone interested.

The body of free software in existence today can only be described as impressive. There are massive quantities of it, most of it of very good quality and some of it of truly exceptional quality, possibly among the best ever produced. The whole thing is spearheaded by Linux, a free operating system for many different types of processors and computers, which is probably the best operating system in existence. It comes surrounded by an extensive guard of system commands, tools, structures and applications from the GNU project of the Free Software Foundation. All this, plus innumerable

programs and applications from many different sources, comes neatly and expertly organized and packaged by Linux distribution, of which the Debian distribution is by far the best, in terms of structure, stability, technical quality and commitment to the open development model.

It is the availability of all this free software, well organized and distributed freely through the network, that allows each one of us to transform into reality the potential enpowerment of the individual that was alluded to before. Without it the whole technology would become stale, an enormous potential never to be fully realized. A free, high quality operating system such as Linux allows one to have complete control over all the potentialities of the existing hardware technology. In contrast, closed and proprietary operating systems tend to limit very strongly what one can do with the computer and the network. Rather than being in control and deciding what is to be done, the user becomes limited to whatever the software or, ultimately, the manufacturer of the software, decides can and should be done within the system.

This is a problem even for those who have no intention of contributing directly to the development of the software, because it means that crucial decisions about the development of the software, instead of being made openly, in the public eye, driven and criticized by the community of users itself, will be made secretly, behind closed doors, by just the few people that own the software. These decisions, instead of aiming at the good of the users, will tend to aim only at the profits of the software companies involved. Becoming a slave to proprietary software is tantamount to leaving in the hands of a very few people, which represent no one but their own financial interests, the future of information technology and hence the future of all, programmers and non-programmers alike. This is even more acutely the case if we are talking of such a centrally important thing as the operating system and the basic tools by means of which one interacts with the technology.

While the use of free software is the way to go in order to face the challenge of the revolution of information technology, again it poses significant challenges. Being much more powerful, free software is also much more complex than most of the proprietary software in current use today. In the world of free software one faces what can be described as an embarrassment of richness, for there is an enormous amount of freedom, a large number of possible choices, innumerous alternatives. One of these challenges is simply learning do use and promoting the training of others in the use of free software. Another one is learning how to make all this information technology, including free software, available in a stable and reliable fashion to groups or whole communities of users. This is specially problematic under the circumstances of widespread necessity coupled with strict limitation of resources which are so common today.

Even if we dream of a radiant future where everyone will have all the possible training opportunities in the world of free software, it will still be true that, while most people can become competent users of the software, not all of them will have the ability and interest to ever become managers of information technology facilities. Far fewer will ever

become programmers of free software and develop the ability to fix the most deep-seated problems and bugs that may show up in it. This is no more than a normal and necessary division of labor, in which most people will be concerned with using the technology to other and important ends, and only relatively few people will be in charge of maintaining and developing the systems. While in the world of free software there are no strict or rigid boundaries separating the definitions of user, manager and hacker or programmer, it is to be expected that only a fairly small fraction of the users will ever go deeply enough into it to become able managers of computer and network installations based on free software. It will always be necessary, therefore, for a few people to put together and manage information technology facilities for much larger groups of users.

Currently the problem of human resources for conceiving, assembling, installing, maintaining and managing such facilities is compounded by the fact that such resources are in fact extremely scarce. This is a consequence of the fact that the technology is so new in a sociological time scale, and that so few people which can have in fact already plunged into this new technology. The traditional role of the system operator or analyst is completely inappropriate for this new world. Instead of a distant and inaccessible specialist that has little contact with the users and, in all probability, no knowledge of the business the systems are involved with, what we need is a manager that is very close to the group of users, probably even part of it, and who knows very well the business of the group. While large organizations and corporations having a relatively narrow and very stable palette of uses for the systems can contract a specialized company to provide the installation and maintenance of the systems, most small to middle-size groups of users are unable to do this and are left to their own local human resources in order to accomplish their objectives within the new information technology.

It must be pointed out that the task of dealing with the new information technology, and particularly with free software, is made so much more challenging and difficult by the fact that it is a moving target, which changes very fast and continuously within the exponential processes which characterize the information technology revolution. This fast evolution may cause a significant amount of instability near the cutting edge of development, specially in communities in which the expectations for new capabilities and possibilities follow closely the development of the technology. While it is always possible to hire a hacker to assemble and install the systems, one cannot prescind of a local manager or management team which is competent enough not only to keep the systems up to date, but also to follow the development of the technology and develop the systems accordingly. If left in a static state, the systems will become obsolete even faster than what is inevitable in the middle of such a revolution, may become riddled by security problems and, in all but the most narrow-minded communities of users, will soon become unsatisfactory for most users.

In short, while the information technology and free software revolution opens up new and powerful possibilities for the individual user and for groups of users, the corresponding social and educational changes are lagging behind and will continue to do so

for some time. The problem poses itself of how to face this revolutionary time and the fast changes inherent to it. In this book we will address the problem of making the new information technology and the world of free software available to groups of users, in a robust and maintainable way, under conditions of scarcity of technically able human resources for the maintenance, management and development of the systems.

1.2 Purpose of This Book

Is this book we will describe in detail a strategy for making the new information technology and the world of free software available to a fairly large number of users with minimal expenditure of resources, in particular of human resources dedicated to taking care of the systems and the network. This scheme has been in successful operation for quite a few years now in the Institute of Physics of the University of São Paulo, serving a few small to middle-size groups of users, numbering from a few tens of users to a few hundred users. These systems are somewhat more complex than what will be described here as an example, but our example should be sufficient to illustrate the ideas and the architecture. If a new system is built based on this example, it should suffice as a solid starting point for a more complex system to develop from, in an incremental way. In principle the architecture can be scaled up almost indefinitely in order to serve really large communities of users.

One of the main ideas of the architecture of the solution presented is to provide the users with ample freedom while giving them a solid centralized support system. Centralized systems which are not properly conceived tend to become a straight jacket around the users, which is something to be carefully avoided. On the other hand, the absence of a minimum amount of centralized support just makes life too difficult for all except very experienced users. This general conflict is just another instance of the universal conflict between freedom and responsibility. Our solution tries to thread a middle path between the extremes of the straight jacket and of the bewilderment and frustration of inexperienced users left to face a powerful and complex system on their own. Starting from a well-structured centrally-managed environment, the users are kept free to move to a more personalized environment at any time and in any rhythm in which they are able to learn how to do this.

The technical scheme described may be used both for the compute nodes on a parallel processing cluster and for remote-boot terminals with X11 capability, always based on the support of one or more central servers appropriately conceived and dimensioned. It has been designed to be very robust and stable, even in environments with frequent and recurrent infra-structure problems such as power and air conditioning failures. It aims at allowing for the best possible use of the existing financial resources, while at the same time providing the necessary amount of reliability, with such things as automatic backup and a degree of redundancy. Facilitating hardware maintenance is another important issue addressed by the solution. Since the system software is so stable, one finds that after the system software has been installed and configured properly, most of the routine work on the system involves either security updates or fixing various hardware problems. Of the various hardware elements common in such systems, the most fragile seems to be the disks, followed at some distance by power supplies, memory modules and some types of input/output units, which require fairly regular cleaning operations.

An important characteristic of the solution is that there is no system disk content on the terminals or nodes, all such disk content is located on a central server. One

may have anywhere from a few to about 50 remote-boot terminals and from a few to a few hundred compute nodes running from a single server, depending on its hardware characteristics. The system images of all these terminals or nodes will fit into a single 36 GB to 72 GB disk on the server. This economy of disk space leaves room for the use of top-of-the-line SCSI (Small Computer System Interface) disks at the central server. Such disks are faster and much more reliable than the lower-cost IDE disks which are the norm on the average home computer. Of course one may still have local IDE disks on the remote-boot terminals, but they are used only for temporary scratch space. It is also possible to use a local swap partition if there is any need for it, or one may configure the terminals to do remote swap through the network on the disks of the server, although usually this is not to be recommended.

In order to describe in general terms the architecture of the solution and to illustrate the possibilities, let us go to a virtual tour around the facilities of a middle-size group using a system built on these lines. So let me invite the reader for a walk around the floor where our example system is installed. As we walk along the corridors and have a look inside the rooms occupied by individual people or small groups, you note that there are many micro-computers scattered around. They all look about the same, with 17 inch color monitors displaying a graphical windowing system and a small tower-type CPU case alongside. About two thirds of them are in fact centrally-managed remote-boot terminals, while the rest are autonomous systems managed by their users. The important point is that each user has access to an intelligent graphical terminal to the overall system or *cluster*, as we call it. Even if this intelligent graphical terminal is a remote-boot terminal, from the point of view of the user it looks just like an autonomous computer connected to the local network.

As we go from room to room you can see users doing many different things, from simply checking their mail to watching movies recorded on CD's or DVD's. You note that complete access and use of any hardware existing on the remote-boot terminals is available to the users. Many of these terminals have a CD writing unit which can be used for account backup purposes. You can see the users doing this, and also listening to music, playing games, writing and running programs, editing and formatting complex documents, and many other things. A very large amount of software seems to be installed on the terminals, where it all runs locally just as it would on an autonomous computer. You note that some of the terminals also have small printers and scanners attached to them, which also work just as they would on autonomous computers.

We now enter the public terminal room, where you can see about half a dozen terminals running. There is also a large central printer installed there. Why don't you try one of the terminals using out visitor's account? After you type the username and password in the XDM banner, you are greeted by a nice, lightweight windowing system, with an icon box and a small quantity of useful buttons on one side. A terminal window opens and in it you can try our user shell environment. As you can see, the `ls` command lists files in quite a colorful way against the black background of the window.

All the file-handling commands such as `rm`, `mv` and `cp` are configured in a safe mode, requiring confirmation before any files are deleted or overwritten. There are many other commands conveniently configured for the users through aliases or shell scripts. Other conveniences, such as a mail client interface and a web browser, are made available by means of buttons in the windowing system's button box.

By means of a command which uses the `rwho` protocol, you can list all the machines existing on the cluster. Typing the name of a machine like a command takes you to that machine. You can also use the name of a printer as a printing command for it. You can also list users which are currently active on the systems. Not only you can send mail to them, you may also talk to them in a chat-like interface, or even call them on the Internet phone, for most terminals do have sound hardware, as well as the software necessary to do this. Using the `df` command in the local terminal shows you that it is indeed a remote-boot terminal, since all the filesystems are available through NFS mounts via the network. Doing the same on the central server, after you have logged into it, shows a system with several large SCSI disks holding many filesystems, including the one containing the home directory of each user. About half of these filesystems are automatically-updated backup filesystems holding copies of other filesystems.

Next to the public terminal room, visible through a partition with glass windows, is our small server room. Access to it is restricted to the people in charge of the system but, since you are our visitor, we will take you in there to have a look, after you log out of the terminal. In the room you can see a couple of servers in large full-tower cabinets, with small 15 inch monitors which work in console mode, without any windowing systems running on them. To one side you can see an open rack holding the network equipment of the floor, in the form of several switches. To the other side there is what seems to be an open bookcase with lots of computer motherboards stored in them. This is our parallel-computing cluster, which we also like to call our PMC (Poor Man's Cluster), a fairly large set of number-crunching nodes connected to one of the servers by means of the other network switches you can see on top of the "bookcase". This cluster of compute nodes is configured in a private network, not directly connected to the Internet, but you may access it through the server which acts as its front-end, and run either multiple single jobs or parallel jobs on the nodes, by means of the PVM or MPI libraries. On another corner of the room you find a fairly large high-end no-break unit, which serves most of the machines on the floor through a dedicated power wiring infra-structure. On a wall you can see a couple of large air conditioning units, but there are no windows anywhere. All this infra-structure makes for a very stable and reliable environment in our server room.

The last stop of our tour will be the technical staff room, where you meet our system analyst and our hardware technician. In the room you can see the two remote-boot terminals used by them, an autonomous system used by them for experiments, tests and training, and a hardware workbench with an open computer cabinet on it, a monitor and several parts scattered over it. This is where our hardware gets maintained,

configured and fixed. There is air conditioning here too, but not so intense as in the server room. A couple of small no-breaks can be seen on the floor near the terminals. Our system analyst takes you on a short tour of the administration of the cluster. She tells you that all the administration of the systems can be done on the central servers and is independent of the current state of the terminals or nodes served by them. One can even install and maintain nodes or terminals which have broken down and are being fixed, or that don't exist yet. One may use all the usual administration tools of the Debian distribution, plus a few additional programs in the form of shell scripts.

As a demonstration of how some aspects of the administration operate, she shows you how we handle a request which was just submitted by one of our users, to make available in the system a certain piece of software. She first checks that the software is available as a package of the Debian distribution. The fastest way to make it available is to install it on the server, where any number of users can use it at the same time. She does so issuing a single command as root on the server, which she accesses through the network, and you can see that not only that package but others on which it depends get quickly installed on the server. Next she sends a mail message back to the user telling him the software is available on the server and will soon be available on all the terminals as well. Although all users can use the software on the server, we always install it on the terminals as well, because in this way the computing load tends to get distributed among the terminals instead of remaining concentrated only on the server. Since all the 30 or so terminals we have share what is essentially a single system image on the server's disks, this can be done with very little disk occupation impact on the server.

Installing the software on the terminals is just a bit mode complicated than the single-line operation on the server. Our analyst first goes to a certain directory on the server, edits a shell script that is there and puts in it the name of the software package to be installed. Then she runs another script in that directory and you can see a program looping from terminal to terminal and installing the software in each one. As she explains to you, the hardware of the terminals is not involved in this operation at all. All the installation work is being done by the server itself, by means of the `chroot` command. This installation procedure lasts no more that a few minutes. After the installation is done and the software is available at the terminals, our analyst runs yet another set of scripts, which scan the system directory trees of the terminals and re-link any files which may have been unlinked by the Debian package installation tools. These are filesystem maintenance tools, meant at keeping the disk occupation at the minimum possible level. It is not necessary to run them every time a package is installed, running them once in a while is sufficient. Two of these scripts, for the root and `/var` filesystems, complete their jobs quickly, but a third one, for the `/usr` filesystem, takes a while to run and she leaves it running on the background. This completes the procedure. Although there is essentially a single system image for all the terminals, this multiple installation procedure followed by the scanning and relinking is necessary in order to maintain the image structured in such a way that it looks just like an individual system

image from the point of view of each terminal.

Our analyst points out to you that the main point of all this is that the whole installation or upgrading operation is highly automated and requires very little work on the part of the human operators, even if it does require a significant amount of work on the part of the central server. Security updates and other forms of software upgrade are done in this way in a routine fashion. We just run the scripts every time that something important for us is downloaded into our local software mirrors, which are automatically updated every day. So long as the central C library is not changed, all this can be done with all the terminals up and running as usual. In the rare cases when there is an upgrade to the C library, it is usually necessary to reboot all the terminals after the upgrade. Major distribution upgrades, however, are a different story. In this case we find that it is not convenient to proceed in this way, and that it is better to take advantage of the fact that only a small number of permanent files differ among the terminals, that number being even smaller in the case of the compute nodes, which are much more homogeneous in terms of the hardware involved. In this case it is better to back up these files, delete the files of all the terminals except one, do a complete upgrade of this single one and re-clone all the other terminals, using the files backed-up as the case may require in order to recover the configurations of all the terminals.

In fact cloning, she says, is the way to go in order to install new terminals or compute nodes. As a courtesy to our autonomous managers, we also use cloning to start up autonomous systems or even home systems belonging to our users, by copying a terminal into the local disk of the machine and then making the necessary adjustments. This is a lot faster and easier than installing the distribution from scratch. In the way of another demo, she proceeds to create the system tree for a terminal that has just been bought and which is not yet installed. She runs another script, this time using the number assigned to the new terminal as an argument. A new system tree is cloned for this new terminal, from our virtual terminal numbered 0, which does not correspond to any existing hardware. The root and `/var` filesystems are cloned very quickly, but cloning the `/usr` filesystem may take a bit longer. For a terminal with about 3.5 GB of software content within the `/usr` filesystem, it will take approximately 15 minutes, while for a compute node with about 950 MB of content it will take approximately 5 minutes. The main system configuration files which characterize the new terminal are changed automatically by the cloning script. All that remains to be done, our analyst tells you, in order for the new terminal to be able to boot, is to write the network-boot program into the flash EEPROM in the network card of the new computer, and record the hardware address of that card in the DHCP configuration file of the server. Well, one must also configure the setup program of the motherboard in an appropriate way. Operations such as programming network cards and configuring motherboards are performed on the workbench.

If this was a new compute node for our parallel processing cluster or PMC, she explains, this is all that there would be to it, you could just put the new node on a shelf

of the "bookcase", switch on its power and start using it. In the case of terminals some more configuration work might be necessary, depending on the amount of hardware inhomogeneity among the terminals. One might have to configure things in order for the X11 system to work on a different video card, and in order to make all the hardware items existing in that particular terminal available to its user. In a fairly heterogeneous cluster such as ours, with terminals ranging from a 200 MHz Pentium MMX to a 2 GHz Athlon XP, with many different types of I/O units, but fairly homogeneous monitors, one can expect that there will be from 20 to 30 configuration files which are different for at least one of the terminals. In the case of the compute nodes on the PMC, this number can be as low as 4.

Other aspects of the operation of the cluster are automated. In order to save precious human resources, any kind of routine procedure which can be automated has been so. One of the most important such things is the backing up of disk content. There are two types of backup, system backup and user backup, which are done in different ways. As you can see when the system analyst shows it in her terminal, all centrally-managed backup is made on disk. In the case of system disks, this is done by mirroring onto a second disk, with the use of `rsync`, with daily updates. In this case the main objective is to prevent against a catastrophic disk failure; if it should occur, we can just plug the backup disk in place of the system disk. All the system disk content of the servers, compute nodes and terminals is backed up in this way. In the case of the user home disks, the backup is done with the use of the `tar` utility, and is incremental, updated every 4 hours. This allows us to make the backup in a double layer format, keeping not only the current version of each file, but also the previous version. In this case we are not only preventing against catastrophic disk failures, but we are also allowing users to recover, by themselves, files they might erase by accident, which is quite a common occurrence. Because of this, the home backup disk is twice the size of the home disk, and since the home backup area may in time get cluttered with old unnecessary files, there is a script that the users may use as a tool for maintaining their backup areas. Besides this, the users are encouraged to make their own backups using the CD writing units available on the terminals in the public terminal room.

Since security is always an important concern, there is also a detailed access control system, that controls the interactive access to every system in our cluster from any other system, inside or outside the cluster. This access control system is all managed centrally by means of netgroups within our NIS structure. Besides, all our systems use password shadowing, and SSL encryption is available for all external connections. The access control system uses the TCP wrapper and adopts a mostly-closed policy, meaning that access is only allowed for the machines which are registered in the access control system. With all this in place, security of network connections ceased to be an important concern, leaving only the discipline of local users to be considered. Since we happen to have no problems there either, and keep all our installed packages up-to-date in terms of security, via the use of the Debian security infra-structure, we have not had

any security-related events in a long time.

Next we talk to our hardware technician. He tells you that he takes care of a constant but not very intense string of hardware failures. The central servers are very stable, since they are built with the utmost care and are installed in a very controlled and stable environment inside the server room, with a high-end no-break and good air conditioning, forming a rock-solid foundation for our cluster. There is much more agitation with the autonomous systems and the remote-boot terminals. There is fairly regular disk trouble with the autonomous systems, power supply failures are fairly common, and sporadically we will be faced by a bad memory module. Once in a while the autonomous managers will need help with some trouble in their machines. We find that our virtual terminal is a very handy recovery tool. It can be booted from a floppy on any machine in the floor, and can be used to recover damaged systems, test disks, memory modules and I/O units, and much more. It can also be used as a platform for cloning a terminal into a newly purchased autonomous system. Our technician tells you that one of our main policies with regards to hardware maintenance is canibalization, that is, using the good parts of otherwise broken machines to fix other machines. This makes necessary our strategy of using only one hardware family, and one with a completely open hardware architecture. Hence all our machines are of the i386 architecture and its clones, with all off-the-shelf standardized parts.

Last, but not least, our local development efforts meant at following the development of the technology should be mentioned. The structure of the clusters we manage is changing slowly all the time, as new things become available and old things become obsolete. One of our tasks is to keep our users able to use the technology in spite of this continuous change, developing continuously not the software itself, but the ability to use it. It is a continuous task of finding out what is useful and adapting it for our use, always making sure it works in a stable and maintainable way. Here are some examples of technological transitions going on at this moment: DVD writers are becoming available for reasonable prices, they can be very useful for massive backups and for massive permanent data storage; kernel patches for writing to CD and DVD writing units just as if they were floppies or other magnetic removable media are available, but not yet in a very stable form, this capability would simplify a great deal the making of account backups by our users; the transport of voice over the Internet is becoming available, we already have a primitive form of Internet phone available to our users, but integration with the traditional telephonic system still eludes us; we currently use 160 MBps (million bytes or megabytes per second) SCSI cards and 10000 RPM SCSI disks, and we still have some 80 MBps SCSI disks in our servers, but the technology is going to 320 MBps interfaces and 15000 RPM disks, this last item being one which we have already missed on occasion; we are still using the kernel-based NFSroot structure for our remote-boot terminals and compute nodes, and eventually we will have to move to a more up-to-date initrd-based NFSroot system, for example to avoid trouble with the use of the APIC interrupt line re-mapping which is becoming the norm even for single-processor

machines.

So we finish our little tour, in the hope that it may have given you a general idea of the objective and the characteristics of this solution, which we are about to describe in more detail, to the problem of making modern information technology and the world of free software available to groups of users. We must now show you how to build and maintain a system such as the one described as an example. The rest of this book will be dedicated to a detailed step-by-step description of the process of putting together, configuring, maintaining and managing such a system.

1.3 Chronicle of Your Adventure

Let us suppose we have a workgroup in need of access to substantial computer power and facilities. How are we to go about providing the needed computer power and facilities with limited resources for both purchasing the equipment and assembling and maintaining it? One common practice is hiring a hacker to do it for you, but this is rarely a very satisfactory solution. Its problems are well known. The hacker comes and does everything in his own very particular and undocumented way. He has no fear of unstable and experimental software and plenty of that finds its way into your system. He has only the foggiest idea of what the needs of the group of users might be, something that the members of the group themselves are quite shaky about at this point. Many things get done because they are cool rather than useful, or things get done in ways that are cool rather than reliable and maintainable. After he leaves, his job supposedly done, the system works for a while, then small problems start to show up here and there. Of course nobody within the user group has any idea of what is going on. After a while the problems start to mount and to really bother you, and eventually you feel forced to try your hand at a software upgrade or reconfiguration in order to try and fix things. While you are at this things start to really break down and you end up with a broken, useless system. What is wrong? Is it hardware? Is it software? No one really has any idea of what went wrong, or where.

This is not your fault. Modern computer systems are extremely complex things and diagnosing problems in them is not an easy job. It sometimes feels like medical diagnosis: both are trials at establishing the causes of problems in extremely complex structures about the current state of which there is only partial, in fact scant information. Just image the medical chaos if natural biological evolution was as fast as the development of information technology. Lots of trial and error are involved in establishing even a partial diagnostic. And in trying to figure things out in the case of information technology, you are in the position of both a doctor and a psychologist, for your problems might be caused by hardware failures, software failures or complex combinations of both. If the systems are not built from the start with reliability, stability and maintainability in mind, they will eventually break and then remain broken until they are re-build from scratch. Hackers are typically very good at solving tough technical problems and at hacking at things until they work, but they are not very good at documenting what they do or at planning for a smooth future operation of the systems. They are also not very organized and disciplined, and usually make poor routine system managers.

Another common horror story is the one in which the group of users hires a professional system manager. This will be in all probability a very traditionally-minded manager, looking back with longing to the time of large mainframes in a faraway and inaccessible computer room. What happens in this case is that services and capabilities start to vanish from the system. Any real inquiries into their whereabouts are met by a barrage on quite incomprehensible technical lingo interspersed with frequent and equally incomprehensible references to system security. It quickly intimidates you and

all the users into not asking any more questions. These people are typically control freaks, subject to deep security paranoia. They must be in control of everything and they must make all the decisions unchallenged, and in order to be able to do this they hide themselves behind what is little more than an empty shell of pretense technical knowledge. These people are very good at telling people what their computer needs are, but not so good at listening to them. They are also quite good at explaining in this incomprehensible way why things you ask for cannot be done. In the end they attain their goal, which is to become system tyrants and, like all tyrants, they work only for the good of themselves. Since nobody else understands what they say and since they are the presumed experts, no one has the nerve to complain and users end up locked up into the tyrant's static vision of the world.

If you are a user and ever got involved in computer and network systems, you probably went through at least one of the traumas described above. Maybe not with the full horror-story intensity with which they were described, but to some non-trivial extent. The upshot of all this is that one must realize that in order to survive within the information technology revolution and in the world of free software, *users must take charge of their informatics lifes*. All users should be able to understand what is happening to themselves and at least some users must be able to understand what is happening to the whole group informatics-wise. This is true both with regards to putting the system together and with regards to operating it. Computer systems are too important and too closely interwoven with the activities of the group for it to be a good idea to leave their fate entirely in hands which are not part of the group. Users must be in charge, in the sense that the person or persons in charge must represent well the interests of the users. In order for the whole thing to work well, everyone involved has to learn at least enough to use the system well and to be able to determine their own changing needs with clarity. In addition, some members of the group of users must learn enough to be able to judge the merit of each alternative in every important decision that has to be made, both while building the system and in the more routine operations of managing, maintaining and upgrading it.

Of course we should not rule out the possibility of hiring people for either the construction or the operation of the system. However, capable and user-friendly system managers for free software are currently a rare and precious commodity, and either hiring them or dealing with them requires that the people in charge know and follow the subject well, if they are to remain in fact in charge. This may be related to the fact that most training centers remain focused on proprietary software, and tend to train people into a kind of static mode, able to deal only with the current version of what is usually a particular brand of operating system, designed and packaged by its manufacturer and owner. People trained in this way become obsolete as fast as everything else in the brave new world of information technology. This static vision of the operating system and of software in general is completely inappropriate for times when a revolution is going on, and more acutely so in the dynamic world of free software. Do not think of your system

as something static, once you have built it. Think of it as something very dynamic, which will change continuously and which will evolve together with the activities of your workgroup within the world of informatics. This will be true both with respect to the hardware aspects and with respect to the software aspects of the system, each one with its own pace of evolution.

Here is a bird's eye view of what awaits you in your technical adventure, in case you decide to do this. We will tread this path on a more leisurely pace later, but this will let you know, on general terms, what is in store for you. You start by trying to figure out what, exactly, are the informatics needs of your group of users. If you do not draw a complete blank, you will go out of that particular activity with only a very vague idea of what is needed. It is not so difficult to gather an idea of the qualitative aspects of the necessities, but as for how much of each thing, well, that is not so easy to gauge. A good balance between quality and price of hardware is another thing which is not always so easy to achieve. This is where the financial constraints of the group may come in handy. You settle for a bare minimum for now. One can always upgrade later, after the use of the system has shown more clearly what the more urgent needs are. This may represent a bit of a problem in the case of the hardware, since purchasing hardware which is later found not to be as upgradeable as it turns out to be necessary may increase the future development costs of the system. This problem is made less serious by the fact that most of the off-the-shelf hardware your system will be based on is highly modular and can frequently be re-used for functions other than those originally intended. But it can still be a problem for some sectors of the technology which are still dominated by closed boxes containing mostly proprietary hardware and system software, such as network switches.

This kind of conflict between a low initial cost and high degree of upgradeability for later use is one you will have to face quite frequently. Keep in mind that, the more closed and proprietary the hardware and system software are, the worst this problem gets. Currently, think specially carefully about your network equipment. So you make your decisions and go out to purchase, with more or less difficulty depending on where you are and what kind of institution your group is a part of, a certain amount of hardware. A little of it will be in closed boxes, presumably in a ready-to-use state. More of it will be assembled basic machines to which you will add parts yourself. Maybe a lot of it will be just parts. It is usually cheaper and more flexible that way and, in some case, such as compute nodes of a number-crunching cluster, the recommended way to go. Sometimes you will be able to find a reliable hardware supplier willing to assemble machines for you according to your detailed specifications, but unfortunately this is much more the exception than the norm. Here some of our main architecture decisions come in: first, you will use only the *Ethernet* physical transport type for your local network, since it is the de-facto universal standard; second, you will buy all your hardware within the world of the i386 architecture and its clones. There are several reasons for this, mostly relating to the openness of the hardware and to the availability of some essential elements of

open software:

- It is an entirely open hardware architecture, based on published open standards, so that there are many manufacturers for every type of part or component, of which there is a large spectrum available.
- It is a hardware architecture for which there is a well supported open-source solution for remote Linux booting, in the Ethernet physical network environment, represented by the Etherboot and Syslinux projects.
- The architecture includes some of the fastest processors in the market, made by two of the leading CPU chip manufacturers, for some of the lowest prices available, due to massive production levels.
- It is the original architecture for which the Linux operating system was written and hence it is the one in the which the system has the longest history and is the most stable, reliable and well supported.

In the future it might be possible to build a system such as the one described in this book with some of the other hardware architectures supported by the Linux kernel, but currently none of the others has all the necessary characteristics described above. For example, Sun workstations traditionally have remote-boot capabilities, but they are also traditionally characterized by proprietary hardware and tend to be much more expensive. Older Digital Alpha stations also tended to have proprietary hardware, and didn't always have remote boot possibilities; newer versions tend to be less proprietary but also do not seem to have a consistent remote boot structure. The same can be said of HP stations which do, however, support remote booting of Linux in a better way. All these other hardware architectures tend to be much more inflexible and expensive than the standard i386 PC architecture. The only other current architecture in which the solutions presented here can be applied in their entirety is the new AMD 64-bit architecture, which can be found in the AMD processors with the amd64 extension, as well as in the Intel processors with the em64t equivalent extension. This architecture is very important because it represents the most viable road for the migration of our solution to the world of 64-bit computing.

Have no fear of assembling the machines yourself, assembling hardware is the simplest task you are going to go through. It is essentially trivial compared to everything else which is coming along. You may need to read a few manuals, identify a few parts, but the whole thing is little more than plugging cards and chips in the appropriate sockets and tightening a few screws. There are some basic precautions which you should take, such as avoiding static electricity in very dry climates. Also a certain general carefulness and neatness helps a great deal. Try not to hammer anything in place, that is never necessary. But, by and large, there really isn't much to this part of the adventure. One hardware aspect which might give you a bit of trouble is assembling a solid SCSI bus

with SCSI disks and a SCSI card, since it works in way which is quite different from the more familiar IDE disks. Only those familiar with the way in which wave guides function will find this a natural thing, and those are very few people. We will go into this in detail later.

Very important: there are a few mistakes that should be avoided with great care since they will certainly result in damage to some of the hardware parts. A few important examples: mistakenly inserting a PCI card on a ISA bus slot will burn the PCI card out; the ISA bus is obsolete hardware and you should in fact avoid altogether any motherboards with ISA slots. Inverting the power connection of a floppy drive will burn out the drive; usually the connectors are keyed and can only be inserted in the correct way, but sometimes it is possible to inadvertently reverse the connection. Inserting a memory module the wrong way will burn out the memory module and probably damage the socket beyond repair; the various types of DIMM memory module sockets are keyed to allow insertion only in the right way, but some lower quality sockets will still permit some electrical contacts to close even if the module is not completely inserted and hence give room for this kind of disaster. Inverting the two side-by-side power supply connectors of an AT-format motherboard will burn the motherboard out; this is, however, an obsolete format and all your new motherboards and power supplies will be in the ATX format, which presents no such problems.

Next you try to turn one of your machines on. Never mind, they never boot on the first try. Back to checking everything out, is the power supply really set to the correct line voltage of 115 V or 220V? Reversed floppy of disk flat-cable, perhaps? Missing connections? After a few trials and mistrials it will eventually turn on and show some things on its monitor. Typically, at this point you should configure your hardware, entering the setup program of the motherboard by pressing the `[Del]` key while it powers up. This enters a menu system which you now have to deal with. Is this software, and therefore the time to start panicking? Not really, this comes later. You are entering a simple program which is hard-wired into the motherboard of your machine. You will use it only to enable or disable certain parts of your machine and, possibly, to fine-tune some operational parameters in order to optimize its performance. By and large, you can leave everything here as it comes from the factory, except in the case of remote-boot compute nodes or terminals, where some non-standard settings will be essential. Some other hardware configuration may be needed. Some SCSI and network cards may have their own configuration menus, although usually there is nothing important to be done with them. Sometimes cards may be configured by means of a separate configuration program, typically ran from a floppy disk containing a bare-bones DOS operating system. Unfortunately hardware manufacturers seem to never supply such programs for Linux.

With your hardware in working order, we come to your first installation of the system, which will be done in the central server. We must assume that you have done this before, possibly installing a home system, this is not an appropriate place to be doing

this for the very first time. If you have not, you are in dire need of help, or you must postpone this until you get familiar with the installation procedure is less unfamiliar circumstances. Here another of our central architectural choices must be mentioned. We will use exclusively the Debian distribution of the GNU-Linux operating system. There are several reasons for choosing the Debian distribution:

- It is simply the best distribution in terms of technical quality, and it has a very good system of quality control for its packages. This is very important for the more central parts of the system software.
- It has a very well-designed and well-programmed system structure, which is robust, flexible and allows the use of remote booting without any trouble. It helps to simplify and organize system administration.
- There is a very large amount of free software available in the Debian package format, both from the Debian project itself and from elsewhere, such as the Debian Backports project.
- The Debian package format is the best one, and the Debian project and other projects associated to it have the best network software and infra-structure for the installation and upgrade of packages.
- The Debian project maintains a very solid form of security support, by means of a security team that quickly issues corrected versions of packages found to contain software with security bugs.
- It is the only distribution which is completely commited to the open development model, both in regards to the software that it distributes and in regards to the software it makes, in the form of system installation and maintenance tools.

At this point we must talk a bit about the popular fallacy of "easy" installation systems. The Debian installation system is generally considered to be a "difficult" installation system, while other more graphical-window, point-and-click systems are considered to be more user-friendly or "easy". First of all, let us point out that installing an operating system from scratch is in fact a complex and difficult operation. The only kind of installation that might be considered easy is copying the operating system of a computer, in which all the problems have already been solved, into another machine with absolutely identical hardware. This is known as *cloning* and we will in fact use plenty of it. Otherwise installing an operating system is a complex and difficult task, period.

The so-called "easy" installation systems do basically two things: they try to guess at your hardware and at your software needs and automatically configure as much as possible; they also hide from you the true structure of the system and the true issues and problems inherent to the installation process and, in doing so, *prevent you from ever learning about these things*. While it is true that such an "easy" installation system,

if successful, can make easier the life of the novice, it will also put him at a complete loss if anything goes wrong in its automatic guesswork. This might be a good way to provide access to a Linux system for the absolute novice, so he can try it out, but it also puts the user in a straight-jacket that prevents him from learning and developing his skills. Basically, such "easy" installation systems cater to the image most people currently have of the computer as nothing more than a point-and-click interface. This is an extremely limited and extremely limiting vision of the user interface to information technology. In point of fact the command line is a much more powerful interface, if also much more abstract in a certain sense, which is essential for anyone aiming at really understanding the structure and the functioning of the systems.

However, this issue in the context of system installation is not its most serious instance, for the same conflict exists and the same conclusions are true in the context of the system administration structure and tools. The Debian distribution tends to use lower-lever tools that show clearly what is really going on, while other "easier" distributions try to do everything in some form of point-and-click interface that hides the true action from the user. If you are going to manage complex information technology facilities, you cannot afford not to know in detail what goes on in the system and you must make the effort it takes to learn about its structure. You must tackle the command line interface and the one thing you do not need is getting yourself into a straight-jacket. Note that there is nothing to be had against the use of good system management tools or, in principle at least, against the use high-level tools. The main thing is that the true action of the system not be hidden from the user. It is just that it is so difficult to write high-level tools that work well and that do not become straight-jackets. There are very few such tools that are worth mentioning. In terms of installation tools, including the automatic detection and configuration of a large quantity of hardware, one of the best is the Debian-based Knoppix CD. It brings newer versions of many software packages and usually works quite well, even if not without sporadic trouble. Remember that we are in the middle of a revolution, that things are always changing and may often break. If you aim at building a stable and reliable system, the best thing you can do is to install the current stable Debian distribution.

If you are going to practice system installation first in some other computer, here is some advice for you: do it on a run-of-the-mill PC, without anything fancy on it; do it on a machine with no disk content, or at least on some empty disk in the machine, so you do not run too high a risk of loosing something; repeat the installation from scratch as many times as it takes, until you understand in detail all that goes on during the process, and acquire a solid grasp of it; do it on a machine connected to a local area network (LAN), not on a standalone machine. You can do the installation from a set of CD's, which is the norm for residential machines, but we will emphasize here network-oriented installation techniques. These start with just a few floppies, most of the system software is obtained through the network from one of the many Debian mirror sites. For detailed information about the installation process you should consult

the documentation distributed by Debian, which is included in the CD's and available in the Debian sites via the network, on URL's such as this one:

`http://ftp.debian.org/debian/dists/stable/main/disks-i386/current/doc/`

We are going to use a particular strategy in installing our first system, which might be called "on-demand installation". The idea is that on a first phase we install only the most essential system software, a really bare-bones system known as the Debian base system. Later we will install each piece of software as the need for it arises. After some time, which we may name a package relaxation period, this will produce appropriate sets of packages for each type of system in your cluster. The important thing is that you will do everything very explicitly and for specific and definite reasons. In this way you will have the best possible chance to learn and absorb completely the nature of the structure of the systems you are building. It might take quite a bit longer, but it is worth it, in the long run. After you have built your first complete cluster successfully, you might use cloning to build other clusters based on it, which might be a good idea unless you want to train someone else at doing it.

Within this strategy we will choose the customized install option in the Debian installer, with no predetermined sets of packages. Besides the disk partitions meant to hold the system of the server itself, we will also leave partitions to host the system of the remote-boot compute nodes or terminals. We will stop the installation right after the basic system is in place and the machine is able to boot from its disk. This will happens when the Debian installer starts something called `dselect`; all you must know about this program is how to get out of it. Instead of using this program, we will configure the system to use a local or regional mirror through the network and we will use the `apt-get` utility to do on-demand package installation. Immediately after you have this bare system running on your server we will use it to clone the system of your first compute node or terminal. Then we will install the services on the server which are necessary to allow the remote booting of this node or terminal. On-demand package installation means that every time you feel the need for something, you install it on the spot.

At this point you will be ready to assemble, configure and boot the first remote-boot node or terminal, which should happen on the workbench, a testing station that you will keep permanently. Once you are able to boot your first node, you may install some reasonable set of packages in it and configure it carefully before you start cloning other nodes from it. This first node might be a real node, meant to run on the hardware of one of the nodes you purchased, or it might be a virtual node, which is not meant to correspond to any hardware in a permanent way. This second alternative is what we usually do in the case of remote-boot terminals, where we find that the virtual terminal has other uses besides being the template from which other terminals are cloned as needed: it also serves as a very good and reliable recovery tool for any Linux systems in your local network. If you are going to have both compute nodes and terminals, you

should do the compute nodes first. Since the system of compute nodes is much simpler than the system of terminals, after you have the compute nodes working you can copy one as you first terminal and complete the software in it before you clone more terminals out of it.

Another thing that you might do right after you install the base system on your server is to build a Network Information System (NIS) and start registering users. After the basic installation there is no definite necessary order in which to do things, you are free to proceed in way that you wish or that other constraints demand. Later on you might want to install a second server and you will be able to clone it very easily from your first server using the virtual terminal. Many services can be installed on your systems, such as services for local and remote users, network services for other machines on the local network or for the Internet at large, communications services, and so on. In later chapters we will suggest and describe some such services and explain how to put them up. Your group of users will be able to use the system optimally and you will be able to keep everything in good running order with relatively little trouble and work as compared to other possible arrangements. If you consider that the architecture of the solution can be easily scaled up from tens of users to hundreds of users, and possibly even up to thousands of users with a bit more work, you will see that the effort to learn, to do it right and to do it well, right from the beginning, will always pay off in the long run.

Chapter 2

The Network

2.1 A Short History of LAN Development

The systems we describe in this book are strongly network-oriented. This is a very different situation as compared to the more familiar stand-alone home machine. Everything will depend strongly on the infra-structure of the local area network (LAN) to which all machines will be connected. We are not talking here of the Internet as a whole, but of the immediate physical network infra-structure that you need for your cluster. This will consist of the cabling infra-structure of the building and of a set of special machines dedicated to network connectivity among your servers, compute nodes or remote-boot terminals. Having a solid and reliable LAN infra-structure with at least reasonably good performance will be essential for successful operation of your cluster. Just like everything else within the information technology revolution, LAN technology is in continuous evolution. However, its evolution present some peculiar characteristics which will affect your decision-making process in regards to investments in the network infra-structure. In order to understand the basic role and characteristics of the LAN in your cluster, we will have a quick look at the past development of this particular aspect of information technology.

The technology of LAN's is strongly dominated by the physical transport technology known as *Ethernet*. It was the first one to appear on the scene many years ago, still in a rather rudimentary form, using coaxial cables. Later other alternative technologies such as Token Ring, FDDI (Fiber-Distributed Data Interface) and ATM (Asynchronous Transfer Mode) showed up in the market, but Ethernet evolved rapidly and eventually prevailed in the field of the local area network. Soon the other technologies were relegated do use only in a few backbone and long distance connections, where ATM still sees use today, while the others are basically gone from the scene. Currently Ethernet is relatively inexpensive, can attain very good performance levels and completely dominates the market for the physical transport layer in LAN's. You will be using an Ethernet-based LAN for your cluster, and it will be the only physical transport technology to be examined to any extent here. Although the Ethernet hardware has changed greatly, the basic software interface has been always the same, so migrating to faster and better network infra-structures has always been fast and easy under Linux.

Borrowing the dates from our particular experience with LAN's, we can say that the first form of LAN to become available was the Ethernet technology for physical transport over coaxial cables, which appeared around 1989. It involved both expensive and rugged connections, using thick half-inch coaxial cables, and cheaper and more flexible connections, using thinner coaxial cables similar to those used for cable television installations, with approximately a quarter-inch diameter. The transport bandwidth was 10 Mbps, or 10 million bits (megabits) per second. The inter-building connections were already done by means of fiber-optics cables, also at 10 Mbps. Connections between different buildings must be made by means of fiber-optics and should not use any copper cables to avoid grounding problems that can cause damage to the equipment, for example during thunderstorms. The dedicated network machines of this period were mainly

simple repeaters and maybe a few bridges interconnecting LAN segments with only a very basic level of packet traffic control. The backbone of the network was typically one or more thick coaxial cables connected by repeaters or bridges. A diagram illustrating this old type of Ethernet LAN can be seen in Figure 2.1.

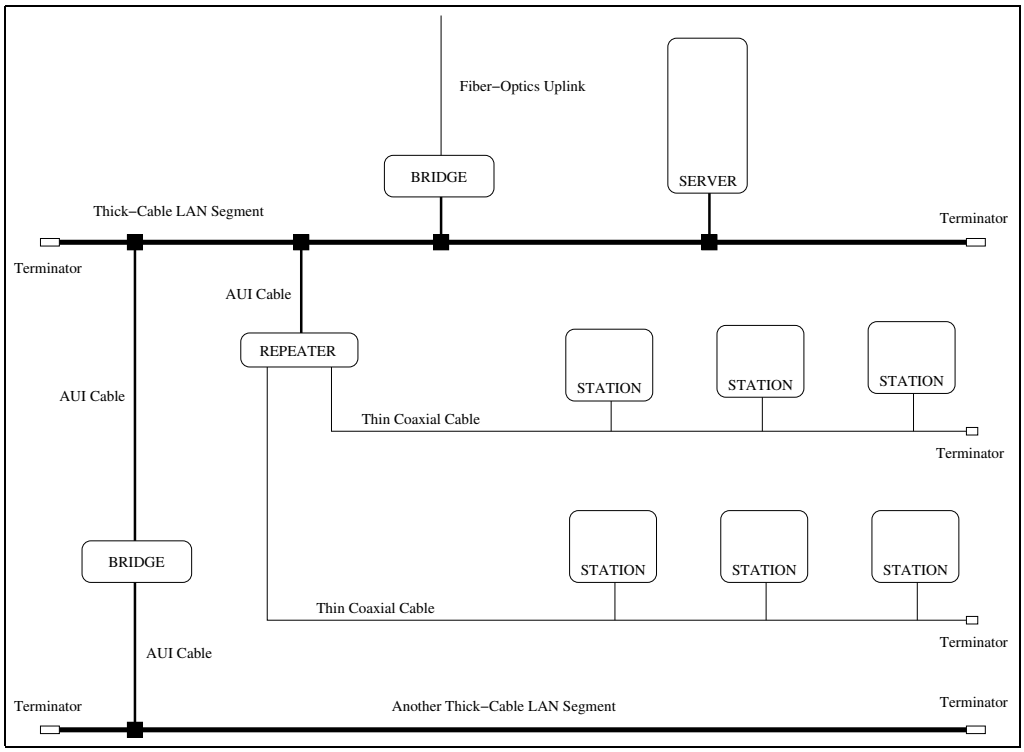

Figure 2.1: An old-style coaxial LAN: the upper bridge connects the LAN to an institutional backbone, typically a router, by means of fiber optics; the second bridge interconnects two LAN segments built around two thick coaxial cables; a central server is connected directly to the thick cable; the bridges, repeaters and servers are connected to the thick coaxial cable by means of AUI cables and taps crewed onto the cable; most stations and micro-computers are connected by means of thin coaxial cables and a repeater.

Each LAN segment, generally spanning a few rooms, a floor or a whole building, was a pure broadcast network with no switching or any other form of traffic control. The machines in a LAN segment were all connected together by means of one or more coaxial cables going from each machine to the next in a configuration known as a daisy-chain. The coaxial cable is a type of wave guide that acts like a communication bus, allowing all the machines connected to it to exchange data packets according to a physical addressing scheme, just like the SCSI connections for disks which we will see in detail later on. The addressing scheme is realized by the Ethernet addresses that are assigned uniquely to each and every network card by their manufacturers, within blocks of addresses assigned

to each manufacturer, assuring in this way that each card produced has a unique address. It consists of six bytes and is usually denoted by six hexadecimal numbers separated by colons, such as `00:00:34:4E:A6:FF`, each byte representing a number from `0` to `255` (or `FF` in hexadecimal notation).

Each computer is connected to the cable by means of some kind of probe or antenna that can interact with the electromagnetic signals traveling along the cable. In the case of the thick coaxial cable one would actually bore a hole on the cable and insert a probe into it. The thin cables used BNC connectors, a kind of connector commonly used in electronic laboratory equipment such as oscilloscopes and signal generators. In this case each computer was connected to the bus by means of a T-shaped connector. In all cases the cable had to be terminated at each end by what is called a 50 Ohm terminator (television cables are a bit thicker and use 75 Ohm terminators), so that the signals arriving at the ends of the cable are completely absorbed and do not reflect back. In this way each computer may insert in the cable a set of signals that gets transmitted in both directions, goes to the ends of the cable and does not reflect back, so that each one of the other computers receives the signals only a single time.

The physical transport technology changed from coaxial cables to unshielded twisted pair (UTP) cables, still at 10 Mbps bandwidth, around 1994. These UTP cables are similar to telephone pairs, but engineered to be able to transport signals at much higher frequencies without too much signal loss or noise. The category 4 UTP cables which were used at first were certified for frequencies up to 10 MHz. They consisted of four pair of wires marked with different colors (brown, orange, green and blue) encased in an overall plastic covering. Each pair of wires was twisted with an appropriate pace so as to minimize interference. Of the four pairs only two were actually used in the 10 Mbps standard, the others were reserved for future development of the technology. At the end of the cables one could have either RJ45 outlets or reliable yet simple and cheap RJ45 plastic connectors, similar to the RJ23 telephone connectors used in the US, mounted on the cable by crimping. It became the norm to have network outlets built within the walls, just like telephone outlets.

This change in the type of physical transport represented a very radical change in the structure of the local network. Unlike the structure of the coaxial cable backbone, which is an extended backbone structure to which machines are connected sequentially in a daisy-chain configuration, the structure of UTP cabling is star-shaped, resembling more the traditional telephone network. In this structure each machine within a LAN segment is connected directly to a central network equipment which receives and redistributes all the traffic. One characteristic of this kind of network topology is that it is much more reliable, since it avoids the easy segmentation of coaxial-cable backbones, which was so common in the case of thin coaxial cables, the most commonly used type, and which could take down the whole segment. In a star-shaped topology any cabling failure will affect only the single machine that cable serves. Besides, unlike coaxial cables, which can operate only in *half-duplex* mode, UTP cables share with fiber-optics cables the

capability of operating in *full-duplex* mode, in which packet traffic can happen in both directions at the same time. Fiber-optical connections use two fiber-optical cables, one for transport in each direction. In the case of UTP cables the two active pairs have this same role.

Another important characteristic of this kind of star-shaped network topology is what we call a *collapsed backbone*. Instead of the extended backbone represented by a long cable going from room to room and being the conductor of communications between the machines, this now happens within the local bus inside the central network machine and allows for much higher performance levels. In particular, it allows for the use of packet switching technologies which dramatically enhances the performance of each network segment. A diagram of this type of network can be seen in Figure 2.2. However, the first type of equipment to appear within this new standard here *hubs*, which were in fact simple repeaters. Just as in the case of coaxial cable repeaters, a packet received in any port is simply repeated in every other port. This means that the network card of each machine has to read the whole network traffic and filter out from it those few packages really meant for that particular machine. This can be done efficiently in terms of the processing demand by the dedicated processor existing in the card, but it still represents very poor utilization of the available bandwidth in that network segment. In a busy network segment, with many computers in it, only a fraction of the packets going through the network connection of a given machine are actually meant for it and therefore the network connectivity of the machine has an effective bandwidth which is much smaller than the actual physical bandwidth of the connection.

Eventually 10 Mbps Ethernet *switches* became available, implementing traffic isolation in a way similar to the older Ethernet bridges. In such a switch a packet received at the port corresponding to a certain machine is repeated only to a single other port, the one that corresponds to the machine the packet is addressed to. In order to to this the switch keeps in memory a dynamical map of the machines which correspond to each of its ports, based on the Ethernet address of each machine, as it shows up as the sender's address in the packets received at each port. In this way each machine connected to the switch has a clean channel with full bandwidth at its disposal. One may think that a switch like this effectively has all its active ports connected in pairs at any given time, so that there is a clean and dedicated channel of communication between the two machines involved. These connections are re-routed or switched very fast, as often as it is needed to satisfy as fast as possible all communication requests from all the computers in that network segment; hence the use of the name "switch" for such machines.

One can see from this discussion that the switch is not a simple electronic repeater, but a computer with an internal operating system which runs a program implementing packet routing algorithms. In fact, in many switches this internal system is accessible by means of a serial port or through the network itself, and the network manager can log into it and monitor packet traffic or change operating parameters. Such machines are qualified as "manageable" and are more expensive than the plainer, non-manageable

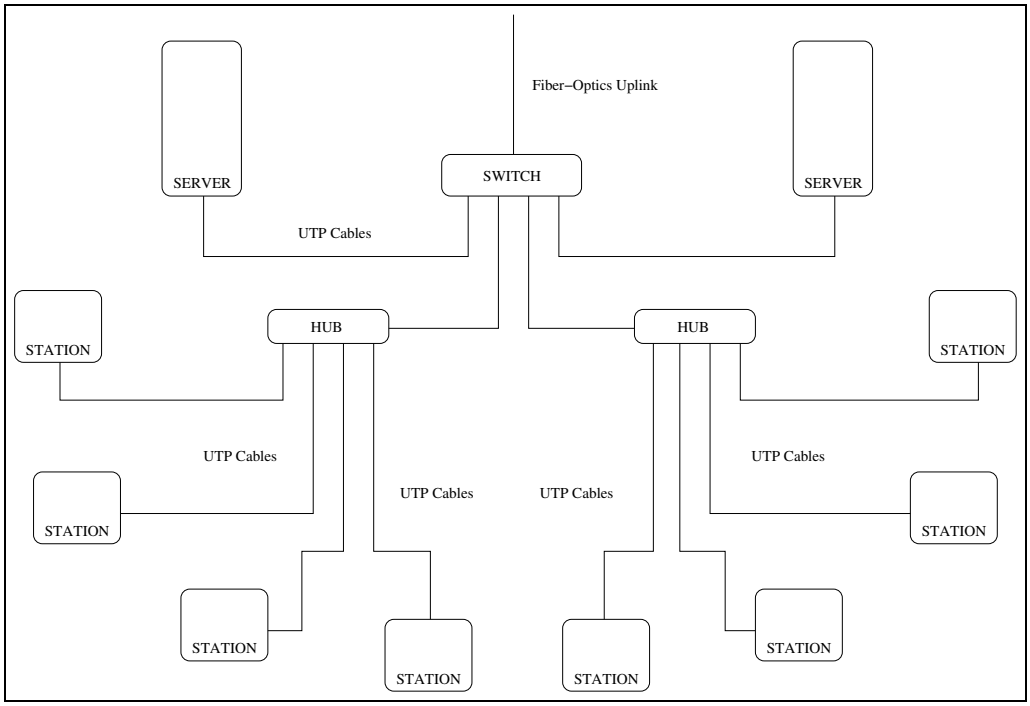

Figure 2.2: An older UTP LAN: the fiber-optics uplink is connected directly to a central switch, which also connects the servers and the other network equipment; the stations are connected through simple repeating hubs; the machines connected to a hub share the traffic on a single connection to the switch.

ones. During this period fiber-optic cables continued to be used for inter-building connections, but moving now to a larger bandwidth standard, at 100 Mbps. Around 1999 the UTP technology started a migration to this new standard, with the use of category-5 rather than category-4 cables. These new cables were certified for frequencies of up to 100 MHz, and although they too consisted of four pairs of twisted wires, still only two were used. A few 100 Mbps hubs still saw some service at this time, but they were quickly exchanged for 100 Mbps switches, which became the norm for all LAN's.

Some of the original network hubs had the property of being *stackable*, meaning that one could connect several of them together in order to produce a single larger hub with many ports. It is not so easy to do this in the case of switches and still guarantee the possibility of a clean full-bandwidth channel between any two ports, but eventually this capability was implemented in the case of switches as well. It is currently possible to assemble several fairly affordable switches to make a large switch with over 100 ports overall. We are now (2005) near the end of this 100 Mbps period, and 1000 Mbps (1 Gbps) backbone connections are already becoming quite common, using the new category-6 UTP cables, which are certified for frequencies up to 500 MHz. This time all the four twisted pairs within the cable are used, two transporting 500 Mbps packet

traffic each, in each direction. Fiber-optics cables are still the norm for inter-building connections, now mostly at 1 Gbps, and full-duplex operation has become the routine for all connections. In Figure 2.3 one can see a typical contemporary network.

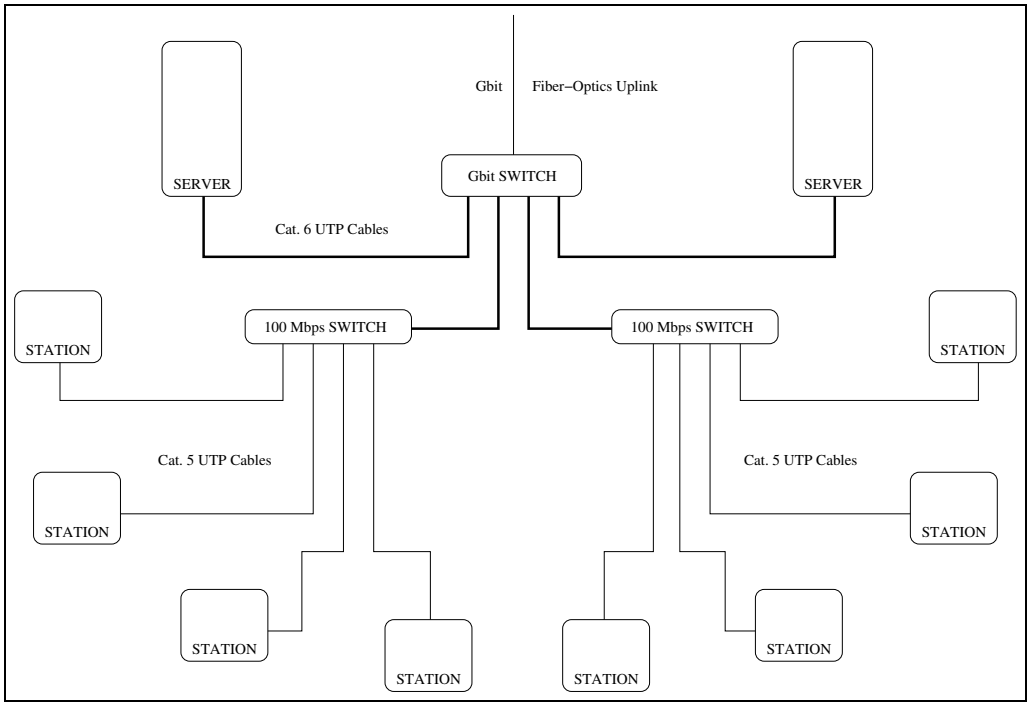

Figure 2.3: A contemporary UTP LAN: the central Gbit switch establishes a Gbit backbone to which the fiber-optics uplink, all the servers and the remaining network equipment are connected; the stations are connected through 100 Mbps switches; each station has a fairly clean 100 Mbps connection to any one of the servers, even if there is a fairly large number of them on each 100 Mbps switch.

There already are some affordable 1 Gbps switches, 100 Mbps switches now routinely have a couple of 1 Gbps ports available and most servers are already connected at 1 Gbps to solid 1 Gbps backbones. In fact, currently we are watching the transition to this new 1 Gbps or Gbit (gigabit) bandwidth standard, as it is called. There already are some 10 Gbps network cards available, using fiber-optics, but these are mostly experimental at this point. It is likely that in a few years we will find ourselves in a period of general 1 Gbps use, with a few 10 Gbps backbone connections, and possibly also a few 10 Gbps connections for central servers. At this time it is still uncertain that this new bandwidth standard will ever be supported over UTP cables, because of the very high frequencies involved. Maybe this time around we will finally see affordable fiber-optic cables arriving at our computers, first at the central servers, then at the desktop. The bandwidth possibilities of fiber-optic cables are almost limitless, depending only on the electro-optical transducing elements and electronics at either end of the cable, a

technology which seems to be currently in rapid development.

As can be seen from the description above, the development of LAN technology is somewhat different from the development of personal computer technology. While both are characterized, on the long run, by exponential processes with similar average development speeds, doubling capacity every two or three years, unlike the development of PC technology, which happens in a fairly smooth way, LAN technology tends to development in big 10-times jumps every 5 years or so, with fairly long intervals of relative stability. Within these stable intervals the new class of equipment with a 10-times bandwidth standard does appear, but is used at first only for a few network backbone connections, by beans of fiber-optics cables, then to connect only a few central servers to the backbone, sometimes using fiber-optics but mostly using UTP cables when they finally become available for the new standard. This is so because it takes a while for the new technology to become mass-produced and sufficiently cheap for general adoption.

The network interface cards, being also part of the usual PC technology, tend to become available at affordable prices quite a bit sooner, but network equipment, such as switches with a large number of ports supporting the new technology, tend to be very expensive at first and take a much longer time to become affordable for general adoption. In general the first network interface cards to become available are the more expensive cards meant for servers, to be connected to a single high-bandwidth port in the first class of network equipment to become available supporting the new bandwidth standard, generally in only one or two high-bandwidth ports of a machine with many ports of the current, lower-bandwidth standard. Only later cheaper cards supporting the new standard become available, allowing its general use in all machines when network switches with a large number of ports with the new bandwidth standard become available and affordable. While it is possible to use a pair of the newer cards to connect two machines directly card-to-card with a high-speed channel without the use of an intervening switch, this is very seldom of much use and usually one must wait for the availability of the new class of network switches in order to promote the general adoption of the new technology.

The peculiar characteristics of LAN development are a consequence of the fact that the market for it has been much more restricted and much less flexible than the market for PC technology up to now. Although a network switch is no more than a computer with a large number of LAN interfaces and some dedicated software to drive them, such a piece of hardware has only very limited and specific applications. Until a form of open-source switching technology associated with off-the-shelf interface cards containing many ports becomes available, LAN technology will continue to be dominated by proprietary and closed solutions combining the hardware and the software in a single package. There is only a small number of companies involved in this market and prices are relatively higher than what is the norm in standard PC technology. However, this may easily change in the future because, with the advent of the public Internet and the fact that it

is becoming progressively more common for people to have several computers at home, small-size LAN's are becoming more and more numerous in homes and business places.

Currently network technology is becoming a lot more common and easily accessible. There are already available quite a few low-end Ethernet switches which have very low prices and very good performance, including bandwidth auto-sensing capability and full-duplex operation. These are commonly called switching hubs and are available in the 5 to 12-port range, 8 ports being the most common configuration. These are becoming common for the 100 Mbps standard, and there are even some for the 1 Gbps standard, although these are still quite a bit more expensive. It is quite possible to have one of these 100 Mbps switches at home, for they may cost as little as some US$ 70.00 or less. Maybe in the future we will see a time when the network equipment market will be dominated by open-architecture hardware and open-source software. Linux may even come to play an important role in it, since it already supports LAN bridging, routing and other software aspects of networking. However, for the time being, specially if we need network equipment with a large number of ports supporting the latest bandwidth standard, we must rely on proprietary machines containing proprietary software.

2.2 Building a Contemporary Ethernet LAN

A local area network (LAN) consists of lots of cables of one type or another, connecting machines with each other. The machines are of two types, one being end-use computers and the other being machines dedicated to making the network operate. The latter are things like hubs, switches and routers. Routers are responsible for inter-connecting two or more different LAN's, and are therefore outside the scope of our discussion here, concerning the building of a cluster within a LAN. They will not be further discussed here, but later on we will discuss the routing issues as they apply to the end-use computers. In some cases one of your computers may do some non-trivial routing, for example in case you use masquerading on the front-end server of a PMC installed within a private network. However, you will not have to deal with dedicated routers, except for knowing the address on the default router in your LAN in order that you can configure your systems to use it as the Internet gateway to the whole wide world outside.

A LAN is a *broadcast* network, in the sense that it allows any machine to send information directly to all the other machines. The definition of a LAN could be that every computer on it can establish direct link-level (physical) communication with every other computer, without having to have packets routed through a third computer or a router. Although the first Ethernet LAN's worked entirely by broadcast, this is no longer the case, as we saw in the previous section. However, there is still the possibility of doing broadcast, by means of a special address which is not associated to any network card or network machine in particular. The Ethernet address `FF:FF:FF:FF:FF:FF` is the so-called broadcast address, and sending packets to it has the effect that every machine on the local network will receive and accept a copy of the packet sent. This means that the switches will forward any packet for this address, which is received at any port, to every other port, just like an old hub would do. Also, every network card knows that it should accept packets to two addresses: its own address and the broadcast address.

Broadcasting is an essential capability in a cluster such as the ones described in this book. Your whole cluster must be within a single LAN, because of the need for Ethernet broadcast during the procedures for booting a compute node or terminal remotely. Remote booting requires such a low-level Ethernet broadcast operation. This happens in the first step of the remote booting procedure, which involves a DHCP (Dynamical Host Configuration Protocol) call. The subsequent operations of TFTP (Trivial File Transfer Protocol), used to transfer the operating system kernel to the remote node, and NFS (Network File System) used to make the system software available at the remote node, can all be made from outside the LAN, but at least the DHCP server must be on the same LAN as the nodes it serves. There is also another level of broadcasting within a LAN, for the IP addressing super-structure also has provisions for it, as we will see later. Besides remote booting, we will also use broadcast for monitoring the status of all machines within a cluster. Anyway, we must have a broadcast LAN in place before we can start to build a cluster.

We may conclude from the discussion in the previous section that at this time (2005)

a contemporary Ethernet LAN consists of a set of 100 Mbps Ethernet switches with a few Gbit ports, a UTP cabling infra-structure and a set of Ethernet interface cards for your computers, mostly 100 Mbps cards plus a few 1 Gbps cards for the central servers. We will now examine in more detail the structure and basic functioning of this local area network, paying special attention to the hardware aspects which are relevant for its use in your cluster. The software aspects of network operation will be examined later, when we discuss the system software of the computers. Since the local network is so important for the operation of a cluster, it is important to have a good and detailed understanding of its structure and at least a basic understanding of how it functions in order to be able to make the correct decisions regarding investments on it, possibly even building it from the ground up.

There are two major aspects relating to investment in the networking hardware, one being the purchase of the necessary Ethernet switches and the other being providing the installation of cabling infra-structure in the building. The purchase of network cards will be considered later as part of the investment on the computers themselves. The first of these two aspect represents one of the most expensive single items you will tackle in your whole clustering adventure, but it becomes a rather simple matter once your purchasing decision is made. You simply buy the equipment, mount it on your network rack, power it up and connect your computers to its ports. There is no need for any configuration, any management or dealing with the internal software at all. All you may want to do is to have a look at the lights in the front panel of the switch to check the state of the network connections or the intensity of network traffic in each port. You may have to do this in case there is some cabling failure and one of your computers is unable to access the network.

If the institution you are in has a central network management team, they may want to manage your switches if they are directly connected to the general LAN of the institution. If that is the case, it is generally a good idea to let them do so. In fact, they might want you to face the extra expense of buying manageable switches rather the less expensive non-manageable ones, so that they can do this. Whether or not you will comply will depend on the finances of your group and on the internal rules of your institution. All that can be said here is that usually it is a good idea to try to comply, within reason and paying appropriate attention to the needs and financial possibilities of the group.

There is one instance when the central network management team need not and probably will not care about the management of a switch, which is the case in which you use it to assemble an independent LAN which is not directly connected to the general LAN of your institution. This is most commonly the case for the construction of PMC's dedicated to some form of number-crunching, but it may be used for remote-boot terminals as well, without loosing the possibility of accessing the Internet from these terminals, as we will see later, by the use of something called *masquerading*. We refer to such separate LAN's as *private networks* and they are useful for several other things,

such as simplifying security concerns and facing a scarcity of available valid Internet IP addresses. There may be a few limitations or complications involved in using a private network for terminals, which we will examine later, but usually these are not very important.

The second aspect relating to investment in the network has to do with the cabling infra-structure. This is typically not very expensive, but can be very messy and cause much annoyance, because it may involve construction work along the rooms and hallways of the building. With any luck, a new building will already have sufficient cabling and wall outlets and they will be of the appropriate technical type. Older buildings may have the infra-structure but it might contain obsolete cabling and outlets, in this case changing the cables and outlets will be necessary, but that probably does not represent a big problem. If the building is really old and has no network infra-structure at all, you have a big problem in your hands and must be ready for much nuisance. Installing physical network structure from scratch is no picnic and you should consider hiring a specialized company to do this for you. There are plenty of these around and it is not so expensive, it might cost you something like from US$ 10 to US$ 20 for each outlet you get in place.

So that you can judge whether your building already has enough cabling infra-structure, or so that you can plan and monitor its installation, here is an overview of everything which is involved. All the material currently in use follow either the category-5 (Cat5) or the category-6 (Cat6) standards. You need Cat5 materials for 100 Mbps connections and Cat6 materials for 1 Gbps connections. All materials listed below should be either Cat5 or Cat6 as needed. The cabling infra-structure consists typically of the following parts:

- A central network rack, where you will assemble a set of *patch panels* and, later, your network equipment. These patch panels are just large sets of RJ45 network outlets which will receive the UTP cables coming from every wall outlet in every room. Ideally it should be an open rack in a small locked room, out of bounds for users and everyone else not involved with system administration. The room should be large enough so that you can install in it, later on, at least the central servers, and possibly also a PMC. If the rack must be in a public room, then do use a closed rack with a lock.

- The UTP cables that should go along hallways and rooms within metal tubing or larger rectangular-section metallic ducts. The metallic encasing helps to shield the cables from noise and interference. These cables must be of the rigid type, in the sense that the wires in each pair within them are solid copper wires. They are less expensive this way, and better conductors of the high-frequency signals, but you should not flex them too much because they might end up breaking, so they should be used only in static assemblies. They should go from the patch panel to the wall outlet in a single piece, without break, no patching allowed along the way.

- The network outlets that should be distributed along the walls of the relevant rooms. If you can, put in more outlets per room than you currently think you need. Computer use is expanding rapidly and it is likely that your group will have more in the future. Also, you should not expect that every outlet you install will always be in use, computers have a way of moving around within the rooms and you should have all sides of each room reasonably well covered, in easy reach of some network outlet. Doubling the total number of outlets with respect to the number of computers expected to be operating is not unreasonable.

- The patch cords at each end of each connection. These are just segments of cable with RJ45 connectors crimped at each end. One uses shorter ones at the rack, to connect the patch panel outlets to the ports of the switches, and longer ones at the wall outlets, to connect the computers to them. These should all be either flexible or semi-flexible, since they may be subject to quite a bit of flexing, specially the ones used for connecting computers to outlets. In these cables the wires of each pair are in fact small cables composed of several thinner wires, being therefore more flexible.

If you are installing everything from scratch, consider using only Cat6 materials for patch panels, cables, outlets and patch cords. You can expect that in about four or five years the general use of 1 Gbps connections will be at hand and, specially in the case of the cabling infra-structure, since extensive work on it is such a hassle, it is a good idea to get it ready now if at all possible financially.

The infra-structure of metallic tubes and ducts can be avoided in the case of a PMC and also in the case of the central servers, so long as they are all installed in the same locked room as the network rack. In this case you may even connect the switch ports to the network cards directly by means of longer patch cords, hence avoiding completely the need for patch panels and wall outlets. These patch cords may be made with rigid cabling, since they will be mounted statically, even if not within tubes or ducts. In fact, it is quite simple to make them yourself in customized lengths, all you need is a crimping toll that looks like a pair of pliers, a roll of cable and plastic RJ45 connectors. In a setup like this you can simply pass the cables around the room and use self-locking nylon bracers to hold them in place along the way.

All the machines involved in this arrangement should be physically off-limits to users and it makes all sense to lock them all together in the same room. The single possible exception to installing all them in a single room might be the case of a very large parallel processing cluster of compute nodes, because such a large machine might dissipate quite a bit of heat and it might be necessary to put it in a separate room in order not to overload the air conditioning system. We will discuss these infra-structure issues in more detail later on. In any case, a PMC typically has its own private network and you can take the switch of the PMC to the other room together with the nodes, so that most connections within it still can be made directly, and only a single high-bandwidth

Contuit type	Number of cables	
	Cat-5	Cat-6
3/4-inch tube	3	3
1-inch tube	6	6
1.5-inch tube	15	12
2×2-inch duct	40	30
2×4-inch duct	80	60

Table 2.1: Maximum number of UTP cables in various types of conduits.

connection has to go from one room to the other in order to connect the switch to the front-end server.

When you plan the layout of your LAN, you must remember that each type of connection has a maximum length limit. For UTP this is about 100 m (328 ft), including the patch cords at each end. Typically one would use up to 90 m from patch-panel to outlet, reserving a maximum of 5 m for the patch cord at each end. Fiber-optic cables do not need to concern you unless your LAN is going to span more than one building, but in any case it doesn't hurt to say that they can be significantly longer, up to about 300 m (983 ft) in the case of the cheaper multi-mode fibers, and up to several kilometers for the more expensive mono-mode fibers. Usually the limit of 100 m is quite sufficient to connect even a large group of users to a single centrally-located room. However, remember that this is the maximum cable length, not the maximum distance between a computer and the network rack in a typical building, and that you must take into account all the turns that the cable must go through to get from one place to the other. If any of your computers is more than some 60 m away from the network rack in a straight line, it is likely that you will have to install a second network rack.

Another important thing to know is how many UTP cables you can fit into each size of tube or duct, so you can plan the physical layout of the conduit infra-structure. You should not cram too many cables into the tubes because one must be careful not to do any excessive physical violence to them. Curves should be smooth and one should avoid twisting the cables or bending them at sharp angles. The maximum recommended numbers of Cat5 and Cat6 cables for each type of conduit can be found in Table 2.1. The numbers are different for Cat5 and Cat6 cables because they have slightly different diameters, the Cat5 cables have a diameter of approximately 5 mm, while the Cat6 cables have a diameter of approximatelly 6 mm.

Most patch panels and wall outlets require a special connectorization tool for attaching the cables to them. After all the patch panels, wall outlets and cables are in place, and all the connectorizations are done, you should test and certify each connectorized connection for the bandwidth standard for which it is intended. There is special equipment to do this, which usually the network infra-structure companies will have and use by the end of the job. They should give you a printout certifying each connection. If

there is a central network administration team in your institution, they might have one too. The testing equipment consists of a transmitter which is connected to one of the outlets and a passive reflector which is connected to the other end. The wall outlets should be labeled with numbers indicating to which rack, patch panel and outlet they are connected on the other side. Good organization is very important here, because you will have to connect, disconnect and re-connect the switch ports corresponding to each machine as they are installed, retired or moved from one place to the other, and it is very easy to get lost in a maze of ports and outlets.

The choice of network equipment is an important task that may be quite difficult, depending on the particular stage of the development of the technology by the time you must make a decision. Unless you are very lucky and already have all the networking infra-structure that you need, you will probably have to buy one ore more Ethernet switches to be able to connect all your cluster into a local network. There are two basic preoccupations here, getting a sufficient number of ports with sufficiently good connectivity for all your machines, and providing interconnection of your cluster with the rest of the network of your institution. The larger your group is, the more complex your problem becomes. A simple and relatively compact cluster may use a single switch or a single set of switches stacked together to give the necessary number of ports. If this solution is possible for you, by all means use it. In this way you will have a completely collapsed backbone represented by the bus within the switch or stack of switches and the simplest and most performatic setup. It is currently possible to have over 100 nodes connected at 100 Mbps in such an arrangement.

Also do not fail to provide, if at all possible, a few higher-bandwidth ports for your central servers and for the uplink which will connect your switch or stack of switches to the rest of the LAN of your institution. Since a central remote-boot server typically has many nodes dependent on it and the network traffic plays such a centrally important role in the operation of the cluster, it is a very good idea to have the servers at a higher bandwidth. Currently the typical situation is having all nodes at 100 Mbps and the server or servers as 1 Gbps. It is also typical to have the uplink at the higher bandwidth, but this may depend on various other factors such as the position of your cluster in the overall LAN of the institution, the relationship of your group with the whole institution, the need or lack of need for a fast connection to the Internet, etc. In general the institution will have a current standard for connections to its overall backbone and in the absence of any special connectivity needs, you might as well follow it.

If for any reason you must use more than one switch without the possibility of stacking them properly, either all in one rack or distributed in two or more racks, then your problem becomes more complex and you should make sure that your setup does not include any serious bottlenecks for the network traffic. If you have two or more switches connected by some of their network ports, then this means that you will have an extended backbone within your cluster, represented by this connection between the

switches, which is effectively a connection between their internal backbones. All such inter-switch backbone connections should be at the higher bandwidth available, currently at 1 Gbps. However, it must be emphasized that for the typical switches available today such connections are *not* equivalent to stacking the switches and may represent a bottleneck to the network traffic. Whether or not such a bottleneck represents a serious problem will depend on the circumstances. Some examples will be discussed in the next section.

In order to better understand the issue of connecting switches together, we must start by defining in a precise way what a switch is. The definition of a switch in the sense we are using the word here is that any one of its ports be able to exchange network traffic with any other of its ports without bandwidth limitations, independently of what is going on with all the other ports. This is what is expected of any single switch that you may get today. In case the switch has high-bandwidth ports it is also to be expected that sufficiently fast switching is available in the machine, as well as appropriate buffering, so that the high-bandwidth port is able to exchange network traffic simultaneously with N low-bandwidth ports without any loss of bandwidth in any of the ports, where N is the ratio of the high bandwidth to the low bandwidth, which is typically 10.

True switch stackability was rather limited until recently, but it seems to have come of age now. Until recently you could stack only up to two switches without having to buy some form of expensive special equipment, and then at most up to four. Now it seems that it is becoming normal to stack up to six switches without a big price overhead. When a certain model of switch is stackable by means of a special stacking cable, what is expected is that any set of N switches stacked together in this way will behave exactly like a single larger switch with N times the numbers of ports. The stacking cable connecting two switches establishes a very fast connection between the bus within each switch, in order to accomplish this. Current possible values of N are from 2 to about 6. In any situation where you must have two switches in different racks, distant from each other, stacking is out of the question because the stacking cables cannot be long, being limited to at most a few feet. In this case you must use a high-bandwidth connection to interconnect the two switches and therefore will have an extended backbone.

Unlike true stackability, simply connecting 100 Mbps switches together by means of a few 1 Gbps ports has been available for quite a while and is becoming common now. Typically, one will have a 24-port 100 Mbps switch with two extra 1 Gbps ports, and will use one such port to connect to the switch above it and the other to connect to the switch below it. The first switch in this pile of switches (which is not properly a stack) will have one high-bandwidth port free for connecting a server, and the last one will have one high-bandwidth port free for connecting to the general backbone. However, one should keep in mind that this kind of interconnection has some limitations and does not truly produce a larger switch with the consolidated number of ports. Instead, it will have different connectivity levels between various pair of ports, depending on where they

are located with respect to each other and also on the general situation of the network traffic the switches are subject to at any given time.

For example, you can inter-connect without any limitations two 10-port 100 Mbps switches if they have each an extra 1 Gbps port, but for larger switches there will be a bandwidth limitation if more than 10 ports on one switch want to talk at the same time with that same number of other ports that happen to be in the other switch. For more than 10 ports in each side this would be possible only with some bandwidth bottlenecking, since the connection between the two switches through which the traffic between these two sets of ports must go is limited to 10 times the bandwidth of each port. Of course, in any case it is necessary that each switch have in fact two extra high-bandwidth ports, so you can connect at least one server and an uplink at the higher bandwidth. One of the switches will be better connected to the server, the other to the backbone. In such situations one should be careful in connecting each computer to one or the other switch in order to minimize the possibility of trouble with bottlenecks, as we will see in the next section.

Currently we are in a transitional period and it is also possible to find an affordable 8 to 12-port fully Gbit switch which you can use to connect together a bunch of 100 Mbps switches with only a single high-bandwidth port each. This is quite a bit better than simply piling up a lot of 100 Mbps switches as described above, but still does not produce a true larger switch if each one of the 100 Mbps switches connected in this way has more than 10 ports. It must be emphasized here that the situation with the technology of network equipment is changing all the time and that your decisions must be made based on a careful survey of the market at the time of the purchase. You might use the ideas and concepts presented here as guiding principles, but it is not possible to formulate any definite and detailed recommendations that will not become obsolete quite fast in this revolutionary technical world.

2.3 Examples and Recommendations

Assuming that you have made your decisions about the layout of your LAN and chosen the spot for your network rack or racks, let us now examine a few examples of clusters of various sizes with a single server and either compute nodes or remote-boot terminals, in order to establish some very general recommendations for network and switch configurations. Larger clusters can be built by scaling up the examples given here. In a later chapter we will discuss some yet untested ideas concerning extremely large clusters. Such very large cluster present special problems and are a bit outside the scope of a workgroup cluster.

Although it is possible to make a very small cluster operate even with an old 10 Mbps hub, or to have a somewhat larger cluster operate based on a 10 Mbps switch, if you do not care too much about performance, at this point in the information technology revolution 10 Mbps network equipment is definitely a thing of the past. Any kind of hub, even a 100 Mbps hub, will limit your cluster to very small sizes if it is to operate at all, both because of the bandwidth bottlenecks of a completely broadcast network and because such equipment cannot have any kind of high-bandwidth ports. If you are going to use an old hub you happen to have handy for building your first cluster of terminals, do not plan to have more than half a dozen nodes in it. For a cluster of compute nodes you might get away with a few more nodes, but computing performance will be very low if your parallel computing model has any dependence at all on network performance, so there is little point in doing this besides technical experimentation. Even if you are interested only in a very small home cluster with a server and three or four nodes, today you can get a very good and affordable 100 Mbps switch to do this and not be limited at all in terms of performance. For small clusters with a single server and up to 6 or 7 nodes, it is not important to have a higher-bandwidth port for the server, so that an 8-port 100 Mbps switch costing as little as from US$ 60.00 to US$ 80.00 will do quite nicely.

We are therefore going to talk here only about switches having a number of 100 Mbps ports and one or two 1 Gbps ports. In about five or six years from now you may have to map this discussion onto a world where all the bandwidths have scaled up by another factor of ten, and of course at that time there might be modifications and novelties which are not predictable at all at this time. Usually you can find in the market at least three general classes of switches:

- Small non-manageable tabletop switches with around 8 ports, meant for very small groups, including home use. These are generally not stackable and will have only UTP ports, typically at 100 Mbps at this time.

- Middle-sized, generally manageable but possibly non-manageable rack-mountable switches with 12, 24 or 48 ports. This class will include stackable models, as well as models with high-bandwidth UTP or optical ports, or with slots for optional high-bandwidth modules.

- Large rack-mounted fully modular and manageable switches meant for major network centers, generally having large expandability, many optical ports and some amount of high-level routing capability, if not including a full-fledge router.

We will be interested only in the first two, in fact mostly on the second class. The first class is useful only for residential systems or for very small clusters for workgroups, with no more than a server and at most half a dozen nodes. The third class is typically just too expensive to be of any use for a workgroup cluster, even for a fairly large one. So we must find our solutions within the second class, which fortunately is usually reasonably varied and flexible.

One of the major problems in purchasing switches for a cluster is to find the correct balance between immediate investment cost and upgradeability. Note that what is meant by upgradeability here is cluster upgradeability, the ability to increase the number of nodes by adding more parts without loss of performance or loss of the previous investment. What this basically means, therefore, in terms of the switches, is stackability. Given a typical type of switch, say one with 24 standard ports plus two high-bandwidth ports, which is quite a common configuration currently, you will find that the stackable model will cost about twice as much as the non-stackable model, possibly three times as much if the one is non-manageable and the other is manageable.

If you are building a small machine the tendency to settle for the cheaper switch is strong, but if later you decide to enlarge your cluster you might have to exchange it rather than add to it. In a large institution you might be able to relocate it to some other end, possibly connecting it to other switches by one of the high-bandwidth ports in some situation where this does not represent a problem. Fortunately switch prices have been falling considerably, as is the rule in the information technology revolution, so this problem is becoming less severe than it used to be, but still you will have to consider the possibility or not of a fairly short-term upgrade of your cluster in order to make your purchasing decision. But remember not to worry too much about possible upgrades in the long term, for in 5 or 6 years the technology will have evolved by another factor of 10 and by then any switches purchased now will be obsolete anyway.

The simplest example of a cluster is that of a PMC built with its own private network. A private network is a LAN using a special range of IP addresses which are reserved for this kind of use, and which is not directly connected to the general LAN of the institution it is located in, or to the Internet. In this type of arrangement the server of the cluster acts as a *front-end*, which means that it is the only machine in this private LAN which is also connected to the general LAN outside and hence also to the Internet. This front-end server cannot act as a standard router to connect the private LAN with the general LAN because the addresses of the private network are forbidden from traveling outside the private network, they are *not* valid Internet addresses. In this kind of setup, in order to access and use the nodes, a user must first make a login into the front-end from the Internet, and from there access the nodes in a variety of ways. One cannot log into a node, or access it in any way, directly from outside the private

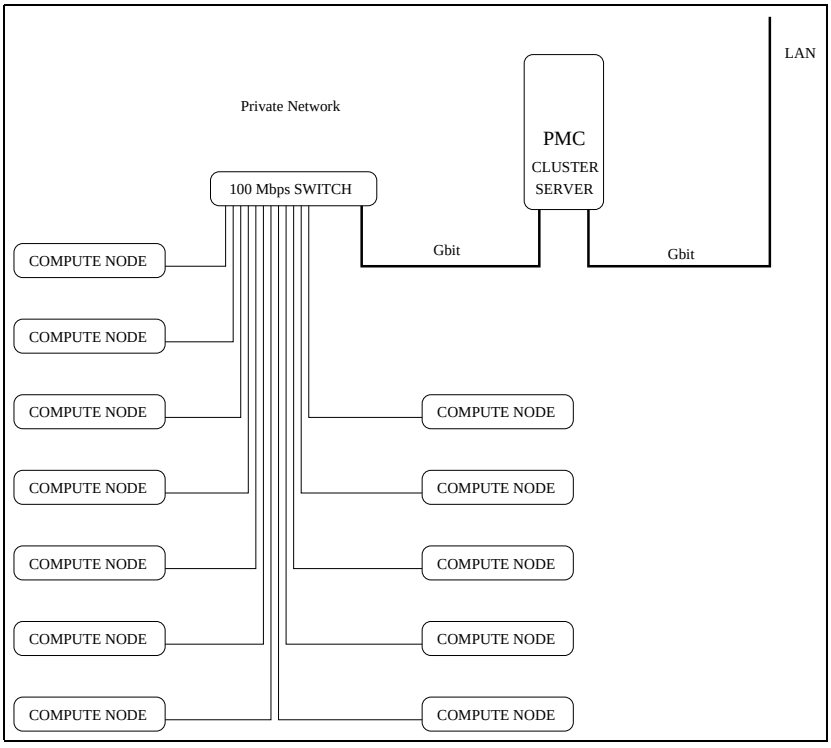

Figure 2.4: A single-switch PMC installed within a private network; all connections use UTP cables going directly from the machines to the switch.

network. The set of nodes acts as a parallel co-processor of the front-end, similar to the mathematical co-processor which every modern processor has within its circuits.

In this case a switch with a single high-bandwidth port suffices to implement the private LAN. Typically there are available switches like this with 12, 24 and 48 ports, including models which can be stacked together. In some cases there are switches which do not have a high-bandwidth port, but which can be stacked with other switches of the same make and line that do have such a port. In this way one can get a larger switch with only the single high-bandwidth port that is needed. The connection arrangement of this kind of cluster is shown in Figure 2.4. Note that the front-end is characterized by having two network cards, one for the private LAN, the other for the Internet connection. If the complete PMC is installed inside a single room which is out of bounds for users, all the connections can be done directly and there is no need for any patch panels of wall outlets, except for a single wall outlet to connect the front-end to the Internet.

There is the important question of how many nodes one can have, served by a single server. By and large, the following rule of thumb applies, it is a kind of square-root rule for network connectivity: if the server connection has bandwidth which is N times the bandwidth of each node then, in so far as the network is concerned, it can serve up to N^2 nodes. This gives us up to 100 nodes served by a single server. Most small to

medium-sized PMC's will fall within this range, and there is no difficulty in obtaining a single switch with up to 48 ports for nodes plus one or two high-bandwidth ports, as well as stackable switches that may have as many as 96 ports overall. In fact, currently one can go as far as 144 ports for a single stack. Since the operating system of compute nodes is rather lean and there is a limited amount of system I/O traffic due to system logging, for a PMC one may in fact go as high as this with a single server.

However, one must not forget that badly written or incorrectly used user programs have the potential of overloading the network by simply writing to disk too much and too often. Since the user-accessible home or scratch filesystems usually will be at the front end or will be accessible via the front-end by means of masqueraded NFS, such programs can seriously disturb the operation of both the nodes and the front-end itself. Of course, the larger the number of nodes, the more serious a problem of this type can get, if it happens in many nodes at the same time. This problem is not particular to clusters, but a general characteristic of number-crunching (numerically intensive) jobs: if they write out too much to disk, they become inefficient and end up being able to use only a small fraction of the available processing power. Users of a PMC should be aware of the characteristics of numerical processing and act responsibly in order not to disturb the operation of the cluster.

This same type of switch or stack of switches can be used in our next example, a cluster of X11-capable terminals rather than compute nodes. One important difference is that in this example the cluster will be completely within the general LAN of the institution and therefore every machine in it will have direct access to the Internet. In this case one may need a switch or stack of switches with two high-bandwidth ports, one for the server and another one for an uplink to the institutional backbone, if it is a Gbit backbone. The connection configuration of such a cluster is shown in Figure 2.5. In case there is no general institutional LAN, one could still build a cluster of terminals reverting to the use of a private network and connecting it to the Internet via an ADSL, ISDN or some other similar line, using the front-end server as the Internet gateway. In this case the front-end server will still have two network interfaces, the Ethernet card for the local private LAN and the interface for the Internet, which may be another Ethernet card or a special ISDN card, depending on the nature of the Internet service to be used.

Another important difference is that in this case one cannot go as high as 100 terminal nodes on a single server, not because of the network but because of disk I/O limitations at the server. This is so because intelligent terminals have a much richer and more complex operating system structure than compute nodes, and the I/O traffic due to system logging is much more intense. Experience has shown that, if you use a 10000 RPM SCSI disk at the server for the system of the terminals, you can have at most 36 terminals before severe performance degradation sets in. In fact, it would be wiser not to go above 32 terminal nodes in these circumstances. With a faster 15000 RPM disk on the server, one can go to a maximum of 48 terminal nodes.

Another way to increase disk access performance is to assemble several disk drives

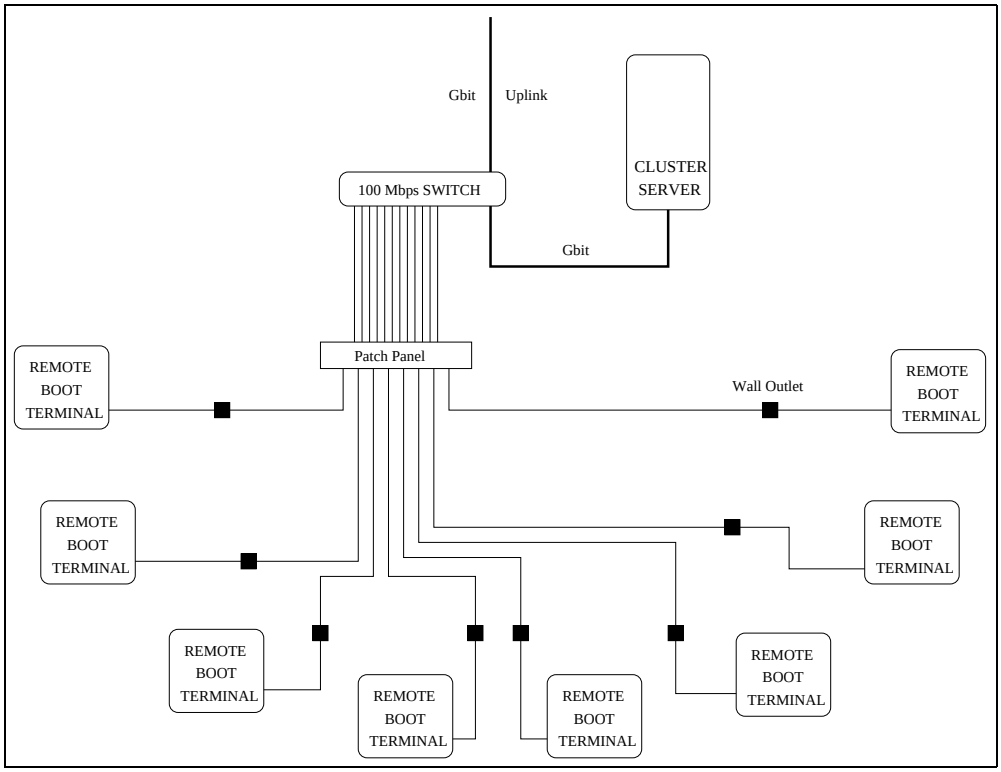

Figure 2.5: A single-switch cluster of terminals, with a Gbit uplink to the rest of the LAN; all connections are by means of UTP cables; the remote-boot terminals are connected by means of wall outlets and a patch panel.

into a RAID-0 or RAID-5 array. This can improve disk access speed by up to a factor of N, if there are N drives on the array, and hence take us from the 40 MBps typical of a single 10000 RPM drive to close to the 160 MBps or 320 MBps throughput of a contemporary SCSI bus. The number of terminals one would be able to serve from a single cluster server would then be multiplied by a similar factor. However, this may turn out to be an expensive solution, for the disks available tend to be quite large and one would end up with (and pay for) many times the disk space needed for the terminals. The use of RAID-5 would have the advantage of allowing you to dispense with a completely independent backup disk, but you would still have several times the necessary disk space. This probably makes the idea financially impractical unless some other use can be found for the excess disk space.

Note that you should always use SCSI disks, and that the limiting factor here is the maximum physical speed of sustained access to a single disk drive, not the nominal speed of the SCSI interface of either the disk or the card. This is why the criterion here is the rotational speed of the disk rather than the speed of the SCSI interfaces. One should have at least a 40 MBps SCSI card, and at least this same speed at all disk interfaces, but this is now a trivial requirement since we are currently seeing the transition from

160 MBps to 320 MBps as the standard interface speed of SCSI cards and disks. The newer class of LVD (Low-Voltage Differential) SCSI interfaces starts at 80 MBps, and for new disks and cards, one should consider having at least 160 MBps SCSI interfaces.

Our third example will be a cluster with two switches or stacks of switches distant from each other, connected by a backbone line. Note that in the case of the typical PMC there is never any need for a setup such as this one, because one does not need to scatter the nodes around the building. Quite to the opposite, in the case of a PMC one wants to keep all nodes as close as possible network-wise, in order to take full advantage of the performance of a single switch, so as to have the best possible communication performance between any two nodes. So PMC's should always have a single switch or stack of switches, so long as it is possible to find stackable switches with enough ports. In the case of terminals, however, one does have to scatter them around in order to serve well all the users. In our case here the connection diagram would be as shown in Figure 2.6.

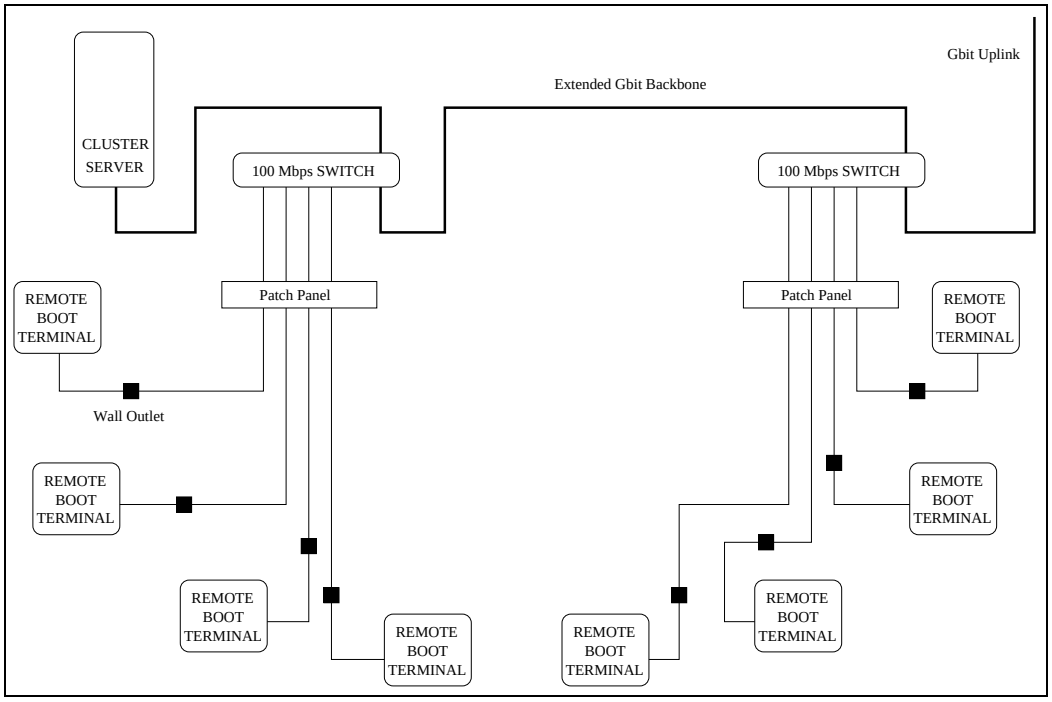

Figure 2.6: A cluster of terminals with two switches and an extended Gbit backbone consisting of a long Cat6 UTP cable; another Cat6 cable makes the Gbit uplink to the rest of the LAN; all connections are by means of UTP cables; the remote-boot terminals are connected by means of wall outlets and patch panels.

The first thing to check in a situation such as this one is whether or not one has a solid backbone for the whole cluster. Since the server is connected to the switch at 1 Gbps, if the connection between the two switches is also at 1 Gbps then there is in

fact a continuous 1 Gbps channel from the server to the internal bus of the far-side switch, and from there to the rest of the LAN through the uplink. If you look at it from the point of view of any single terminal on the far-side switch, there is a continuous 100 Mbps channel going from it to its server, just as there is for a terminal on the near-side switch. If there are more than 10 terminals on the far-side switch, of course they can overload the backbone connection, but this is not different from the situation for terminals at the near-side switch, since in both cases the connection of the server itself would be equally overloaded. None of this represents any real problem, because very seldom does a terminal keep solidly busy a 100 Mbps connection for any length of time. Even when it does happen, it is extremely unlikely that a number of terminals will be doing this at the same time. Of course, if the connection between the two switches is only at 100 Mbps, then we do have a serious bottleneck.

With a 1 Gbps backbone connection between the two switches, one should see any traffic congestion only if there is some other traffic sharing the backbone connection with the at most 24 terminals one would expect to see in the far-side switch. If it is a large switch connecting many other machines unrelated to the cluster, but which generate traffic on the backbone line between the two switches, for example because of some general-purpose server connected to the near-side switch, then one may start to have some traffic problems, depending on what those other machines do. The 1 Gbps bandwidth of the backbone connection will be effectively diminished by the competing traffic. The worst possible scenario is one in which both switches have extra high-bandwidth connections and the 1 Gbps line between the two switches turns out to be part of the backbone of the general institutional LAN, with a sizable part of a large LAN behind each one of the two uplink connections. Due to this there may be intense traffic going through the 1 Gbps line between the switches, which is completely unrelated to the traffic within the cluster. In this case the backbone of the cluster is not clean, it is tainted by other traffic and there can be serious performance degradation of the terminals on the far-site switch.

It is important to have a fairly clean backbone in a cluster which has internal backbone connections. Put as many of your terminals as possible in the same switch that the server is connected to. If the server and the terminals are all in one switch, they have their bandwidths guaranteed, independently of what is going on in the rest of the LAN. If you must have a cluster with an extended backbone, make sure it is fairly free of foreign traffic. It is OK to have a number of autonomous systems within the workgroup sharing the switches with approximately an equal number of terminals, but one should avoid having a foreign backbone connection going through the cluster. Installing remote-boot terminals on a switch which is network-wise far away from the server is probably a bad idea most of the time. For example, experience shows that on a large LAN even a single terminal on a far-side switch which is separated from the main cluster switch by two or three non-dedicated backbone connections might result in very severe performance degradation for that terminal if the intervening backbone lines are very

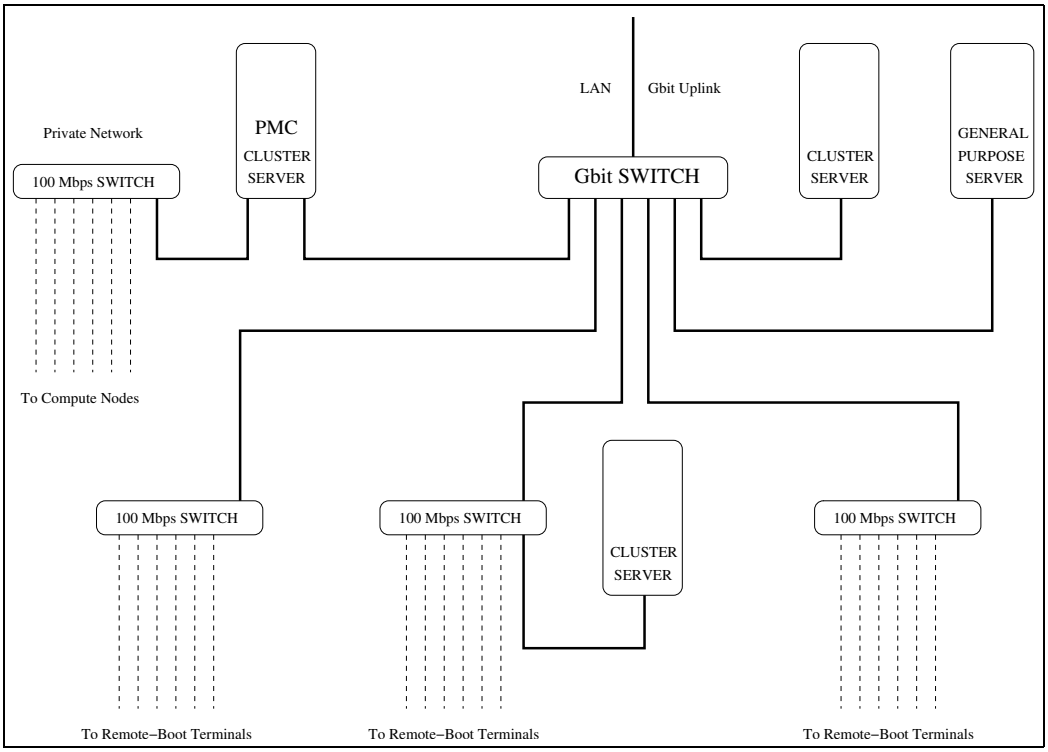

Figure 2.7: An large cluster of terminals and compute nodes, with an extended Gbit backbone built around a small Gbit switch; the remote-boot terminals are distributed on two cluster servers, and there is also a PMC cluster server and a general-purpose server; even if one has a solid Gbit backbone, it is always a good idea to have each cluster server as close as possible to its terminals, in network terms.

busy, and specially if the intervening network has lots of broadcast traffic and is prone to broadcast storms.

As our fourth and last example we consider a large cluster using a small additional full Gbit switch with 8 ports for connecting several servers and switches in an extended backbone. Not all of the servers need to be cluster servers, but on a large enough cluster one should have more than one cluster server, since one should not have more than 48 terminals served by each server even if they have the fastest disks available. The other servers could be, for example, the front end of a PMC installed on a private LAN, a general-purpose workgroup server or a machine dedicated to some special form of processing. The connection diagram for such a cluster would be as shown in Figure 2.7. Keeping a consistent and clean Gbit backbone is a preoccupation here, just as in the previous example. The fact that the single uplink connection is made at the Gbit switch to which all the servers and all the other switches or stacks of switches are connected guarantees this. Currently an 8-port rack-mountable but non-manageable Gbit switch can be obtained by approximately the same price as a manageable but non-stackable 24-

port switch with two extra Gbit ports. This is still quite expensive by current 100 Mbps standards, but is already within the realm of possibilities of a large workgroup and hence makes building a larger cluster feasible. Besides, prices of Gbit switches are going down fast, and there are already some 8-port tabletop Gbit switches available.

One can go up to fairly large clusters with an arrangement such as this one. Using an 8-port Gbit switch such as this one it is possible to have up to about 6 cluster servers and a total of up to some 200 remote-boot X11 terminals. With a 12-port Gbit switch one can scale up to about 10 cluster servers and hence up to over 300 terminals. If you have a cluster with several servers, they can be cloned from each other and managed in an homogeneous way in order to simplify the administration. It is even possible to imagine a two-layered cluster in which the first layer is a cluster of cluster-servers based on a central super-server, which in turn serves a second layer consisting of terminals. In this way one might scale the cluster up to thousands of terminals, as will be discussed briefly in later chapters. One could consider building a PMC on these lines, which could go up to several thousand nodes, but one must keep in mind that the set of 100 Mbps stacks of switches interconnected by means of their Gbit ports to a central Gbit switch is *not* equivalent to a large switch with all these 100 Mbps ports. There would be no homogeneity of connectivity among all the nodes. The nodes connected to a single stack of switches would have guaranteed 100 Mbps interconnection bandwidth, but the interconnection of two nodes connected to two different stacks of switches could be subject to bandwidth degradation depending of what is going on in the PMC's network as a whole.

As we hope to have shown by means of these examples, building small to middle-size clusters of either compute nodes or X11 terminals is a quite simple and straightforward thing to do, in so far as the design of the network is concerned. Even designing networks for a fairly large cluster, or for a cluster which must be distributed over a wider area, is still a fairly simple jigsaw-puzzle game involving combining different switches and considering their prices. Building very large clusters is a more complex problem that will require solving a more complex jigsaw puzzle. By means of these examples we hope to have shown clearly what are the main parameters and issues that one should have in mind when tackling the problem of conceiving and designing in an appropriate way the local network for a cluster.

Chapter 3

The Server

3.1 Choosing The Right Hardware

With your network infra-structure in place, your next priority will be to get your central server up and running, since everything else depends on it. The first decisions to make concern what hardware to get for it. We will discuss here how to design a typical cluster server, which may be used for either compute nodes or remote-boot terminals. One must keep in mind that PC technology changes fast and continuously, so any particular performance numbers mentioned here must be scaled up appropriately at any later time. In a typical cluster server CPU speed, although relevant, is not the most important concern. The most important concern is having a solid I/O (Input/Output) and bus communications structure, with the best possible performance. It is also important to have a fairly large amount of high-performance RAM (Random Access Memory), in order to avoid paging as much as possible and to make sure that the system will have plenty of memory for disk content caching. Having a sufficient amount of high-performance disk space is also a requirement. Last but not least, operational reliability is very important, which will imply care with the quality of all parts and a large, good and solid cabinet.

The requirements for high-performance I/O and bus communications will affect primarily the choices of motherboards, SCSI cards and network cards. The motherboard is the central component of the server, it is one of the most complex components and must be chosen with great care. It contains a system of busses which is responsible for the communications among all the main parts of the computer, both internal communications such as between CPU and RAM and external communications such as with the disks and the network. The motherboard contains a set of special chips that has the role of implementing and controlling this set of busses, and which is known as the *chipset*. The motherboard has slots and sockets to receive one or more CPU's, memory modules and various types of I/O cards. The choice of motherboard will be driven mainly by quality, the type and performance of the busses it contains, compatibility with the Linux operating system and price. Unlike the choice of CPU chips for our particular cluster architecture, which is rather limited, the choice of brands and models for motherboards is enormous. However, since we will have quite a long list of requirements for the motherboard, the choice will quickly be restricted to just a few brands and models.

Before we detail our requirements, we must review some of the technical details regarding the various types of bus which are involved. A modern motherboard will have three kinds of busses: a local or front-side bus which is used for communications between CPU and RAM; a PCI (Peripheral Component Interconnect) bus which is responsible for communications with various types of I/O cards; and an AGP (Accelerated Graphics Port) bus which is used to connect a video card to the system. Older and obsolete I/O busses such as ISA, EISA and MCA should not exist on a modern motherboard. If they do exist in some older motherboard you intend to use, the order is to solemnly ignore them. Each one of these three relevant busses will be associated to a certain number of sockets or slots by means of which you can connect components to them.

Some devices which are usually contained within the motherboard itself, called *on-board* devices, will also be connected to the PCI bus internally, by circuits within the motherboard. Some common examples of such devices are the traditional serial and parallel ports, the keyboard driver, the PS/2 mouse driver, the floppy disk driver and the IDE hard disk interfaces.

The characteristics of the local bus are determined by the type and speed of the CPU and RAM which are supported by the motherboard, so you do not have to worry about it directly. If the motherboard supports a certain CPU and a certain type of RAM, each one with a certain frequency, then it has the local bus it needs. The AGP bus need not concern us here, because we will not make any special use of the video card. A server should *not* have an X11 server running on its monitor and so we require of the video card nothing more than that it work well in the ASCII (text) console mode. Any video processor speed and amount of video memory will do, so any AGP bus will do, regardless of its characteristics. In terms of video for the server, the one thing we do *not* want is a motherboard with a video card on-board, so we must make sure that the motherboard does have an AGP slot, but we do not have to worry about its characteristics. Therefore, we are left with only the PCI bus to worry about, in regards to the choice of motherboard for the cluster server.

For a long time, and until rather recently, the standard PCI bus had a 32-bit bandwidth and ran at 33 MHz. This means that information was carried by it in 32-bit (4-byte) chunks, 33 million times per second. This gives a nominal data throughput of 133 MBps, which is fairly high but is becoming slowly obsolete. In time two types of extension of this bus appeared: 64-bit (8-byte) bandwidth busses, as well as busses able to run at 66 MHz. Any of these extensions took the data throughput to a very comfortable 266 MHz. Then busses supporting both 64-bit bandwidth and 66 MHz appeared, taking the data throughput to an excellent 533 MBps. Even more recently a new backward-compatible standard showed up in the scene, the PCI-X bus, with a 64-bit bandwidth and able to run at 133 MHz, taking the data throughput to an amazing 1 GBps. It is clear, therefore, that currently we do have a sufficient variety of I/O bus capabilities within reach. Let us now establish a list of our most basic technical requirements for the motherboard of the cluster server:

- The motherboard should accept the current class of AMD or Intel CPU chips. Currently this means the Athlon XP in the case of AMD, the Pentium IV in the case of Intel. Typical CPU frequencies at this time are in the 2 GHz to 3 GHz range. These are not necessarily the latest classes, but those that have the best current price/performance ratios within minimum performance limits.

- The motherboard should have, at the very least, 3 slots for memory modules, ideally 4, possibly more if a really large amount of RAM is required. The current memory type supported by the CPU's above is DDR memory, which comes in 184-pin DIMM format, with sizes from 256 MB to 1 GB per module. The frequencies

supported by these modules is currently in the range from 266 MHz to 400 MHz, 333 MHz being a good typical value.

- The motherboard should have as few on-board devices as possible. It is a common thing to find on-board video cards and sound cards on many motherboards. Server motherboards sometimes have on-board network cards and SCSI cards. All these should be avoided if possible, video cards in particular, specially if they share system RAM. If any such devices do exist on-board, it should be possible to disable them in the BIOS setup.

- The motherboard should be compatible with the Linux operating system. Currently this is hardly an issue, for essentially all available motherboards are compatible with Linux, with the possible exception of some on-board devices such as video and sound cards, which we are ruling out of the configuration in any case.

- Some common on-board devices are in fact required, namely a serial port and a floppy disk drive. Some other on-board devices are practically unavoidable, since almost all motherboards come with them, but they do not represent a problem and will simply be disabled. This is certainly the case for IDE interfaces, while parallel and USB interfaces typically will be disabled unless they are needed for a printer.

- The motherboard should have an AGP slot and no video card on-board. Motherboards with no AGP slot and an on-board video card are typically very cheap low-quality models that are not at all appropriate for any kind of server. Also, on-board video cards usually use part of the system RAM as video RAM and it is not possible to disable them. There may be a few motherboards with on-board video cards which do not share RAM in this way, but instead have their own memory, those are acceptable.

- The motherboard should have an appropriate number of fast PCI slots, typically 2 or 3. These slots should at least support the 64-bit bandwidth standard or the 66 MHz standard, preferably both. These faster PCI slots are currently available only in dual motherboards, that is, motherboards for two CPU's, and it is in fact a good idea to have two CPU's rather than one for several reasons. Therefore, a dual motherboard becomes part of our standard server configuration.

The reason for requiring that the motherboard have as little as possible on-board is to ensure greater flexibility, for easier upgradeability and the possibility of changing the role of the machine in the future. In general a motherboard with high-end drivers on-board, such as SCSI cards and Gbit network cards, is considerably more expensive, to account for the value of these extra devices. However, sometimes a model having some extra on-board devices such as sound cards may be actually cheaper, in this case there is

no problem in using the motherboard so long as these unwanted devices can be disabled in the BIOS setup program, which is usually the case.

A word should be said here about the issue of hardware quality. Years ago computer hardware used to be much more expensive, was produced in much smaller quantities, development was much slower and a single Unix vendor used to develop its proprietary system software in integration with the development of the hardware. As a consequence of all this, it was much easier then to ensure a high level of quality for the hardware. Today all this has changed, prices are much lower, all hardware is mass-produced, hardware development is very fast, and the hardware and the system software are developed separately by independent organizations. As a consequence of all this, now it is much harder to ensure a high level of hardware quality, and motherboards are particularly susceptible to a variety of generally small but possibly serious hardware problems, that sometimes can render it inoperant.

What those problems might be changes in time and cannot be predicted here, but keep in mind that hardware breaks as a matter of routine, and all we can hope for is that the breakage rate is not too high. For example, ventilation fans used for cabinets, processors and other chips are common items of failure and should really be considered as supplies items. Specially in lower-quality motherboards it is common to find on-board devices with hardware bugs or parts that fail after a while. Recent examples of this include: motherboards with bad capacitors that heat up and fail after some time, which one can spot by their bulging tops; motherboards with small 40 mm diameter fans on the chipset, needed because of the ever higher bus frequencies used, which slow down and cease to work after a little while; the same type of small fan is also used on the chipsets of many high-performance video cards, where they are equally troublesome; motherboards with buggy PS/2 mouse devices which can cause them to hang if the PS/2 mouse is used.

About the choice of processor, there is very little point in getting the fastest available CPU, since this will usually be quite a bit more expensive and it is not really essential to have it. Usually it is a good idea to descend a few notches within the CPU family, and get the model with the best price/performance ratio. The same is true for the type of memory, but once a CPU is chosen usually one can settle confidently for the fastest type of memory which is supported by that CPU. There is a decision to make in terms of the size of the memory modules too. Within whatever is supported by the motherboard chosen, the larger the modules the larger the total memory that the system can have, either immediately or in a future upgrade. However, the largest modules are usually considerably more expensive in terms of price per MB. The idea here is to choose the largest size of module which still has approximately the same price/MB ratio as the smaller modules. If any memory slots are left after the amount of memory intended for the machine is reached within this scheme, those remaining ones are kept for a possible future upgrade.

About the choice of CPU sub-architecture within the i386 architecture, for quite a

long time there have been only two brands that it is worth considering, AMD (Advanced Micro Devices) and Intel. Although this kind of choice may change in time, in the last several years a tendency towards preference for the AMD chips has been establishing itself rather strongly, for intensive numerical processing and other high-performance applications. The Athlon class of CPU's from AMD has consistently given the best price/performance ratios. It is true that the previous line of Thunderbird Athlons, with frequencies of up to 1.3 GHz or so, used to present cooling problems due to excessive heating, but that situation has improved greatly since the introduction of the Athlon XP (to be used for single-processor machines only) and Athlon MP (for Multi-Processor machines) lines. While it is true that the operation of the Pentium is still somewhat more stable, the price/performance ratio of the Athlon tends to tip the scale in its favor in most cases. If a very high degree of operational stability is essential to you and you are willing to pay the price for it, choosing the Pentium may be advisable. Otherwise, the choice of the Athlon is almost inevitable.

One very important issue regarding fast CPU's, which dissipate a lot of heat, is control of their temperature. Particularly, it is important to ensure the reliability of the operation of the cooling fan on the heat sink of the CPU. It is no use having a high-speed, high-performance fan if it may suddenly fail due to dust or other environmental problem. If a CPU fan does fail and this is not noticed in time so that the system can be powered off, the CPU may burn out. The very fine-grained dust commonly found suspended in the air in big polluted metropolitan areas is a very serious hazard to cooling fans, which may loose rotational speed and even stop completely. Smaller fans with higher rotational speeds are more susceptible to this than larger, slower fans. If possible, use on the CPU's of the server a larger heat sink, with a standard-sized 3-inch fan (its frame is actualy a 80×80 mm square) with a rotational-speed sensor. These larger, slower fan are not only more reliable, they are also easier to find and purchase, if you have to exchange one for maintenance purposes. In the case of smaller fans, you may have to buy a whole new heat sink and fan set if you need a new fan. If you use this larger fan, you will be able to assemble two in each heat sink, one on top of the other. This may work as a safety measure, in case one of them fails, as well as a cooling enhancement while both are operating. However, it may also increase greatly the noise level of the CPU fans. While this is not usually a problem in a dedicated server room with no human inhabitants, it may be a problem in other circumstances.

Another thing which must be mentioned is the situation regarding 64-bit processors. All the processors mentioned above are 32-bit, but both AMD and Intel already have lines of 64-bit processors. Intel has the Itanium line (architecture ia64, as opposed to the ia32 of the Pentium) and AMD has the Opteron or Athlon-64 line (architecture x86-64). Of these the only architecture in which one can currently be fairly certain that our cluster architecture can be completely realized is the x86-64 architecture. Although it is still a bit early to do this in production mode, one can foresee that within a year or so this will be not only a practical solution, but is likely to be the dominant trend in this

area. It is quite possible that one will also be able to use the future 64-bit generation of the Pentium which is being promised by Intel, but it is unlikely that one will ever be able to fully realize the cluster architecture using the Itanium, because of both technical and financial considerations. So we see that also in this case AMD seems to be gaining the upper hand with its Athlon/Opteron family of CPU's.

We must now discuss the basic properties and characteristics of the main I/O cards you will need for your cluster server. Let us start with the network cards. A network interface card or network adapter is a card that can be inserted in a slot on a standard PC bus such as PCI. It contains a *transceiver*, or a transmitter-receiver combination that allows it to transmit and receive packages of data through some kind of physical medium, such as coaxial cables and twisted pairs. It also contains a dedicated processor which is used to perform low-level manipulation with the data read from or written to the physical media, off-loading the CPU of the machine. The old 10 Mbps cards used with coaxial cables and AUI connections used to be ISA cards, meant for use in this slow and now obsolete type of bus. The more recent 100 Mbps cards used with twisted pairs are usually for the PCI bus, the current standard I/O bus of PC technology. There are also 1 Gbps cards for PCI busses, both reasonably cheap ones for the traditional 32-bit, 33 MHz bus and more expensive ones for the newer faster types of PCI. There are even some recent PCI cards supporting 10 Gbps over fiber-optic cables, but these are still in an early release stage.

A SCSI (Small Computer System Interface) interface card or SCSI adapter is another type of card that can be inserted in a slot on any standard PCI bus. What it does is to implement an interface or bridge from the PCI bus to another kind of bus, the SCSI bus. This SCSI bus is in fact a special type of cable, which can extend either inside or outside the system cabinet, and to which one can connect SCSI devices, typically high-performance hard disks. Years ago SCSI busses used to have an 8-bit (1-byte) bandwidth and to run at 10 MHz, with a data throughput of only 10 MBps. Subsequently faster busses were released, with either a 16-bit (2-byte) bandwidth, called Wide-SCSI, or able to run at 20 MHz, called Ultra-SCSI. Either of these two had a data throughput of 20 MBps. Then busses with both a 16-bit bandwidth and running at 20 MHz appeared, with 40 MBps throughput. It is easy to recognize the development pattern, which is very similar to the one for the PCI busses. Subsequently a new standard was introduced, called LVD (Low-Voltage Differential), which was capable of transporting twice the information using the same frequency and bandwidth, and so took data throughput up to 80 MBps. We now have a 160 MBps LVD-SCSI standard, which is currently the most common, and a similar 320 MBps standard which is currently being introduced, both using higher frequencies but the same 16-bit bandwidth.

The data throughput of the PCI bus of the server should be compatible with the data throughput of the cards meant for it. Let us look at a few examples, in order to put things in perspective. Since 1 MBps corresponds to 8 Mbps, a routine but good-quality 100 Mbps network card is capable of 12 MBps, which is well below the maximum capacity

of any PCI bus; a 1 Gbps network card can handle up to approximately 125 MBps, which is already quite high but can still fit, if the network card is the only active card in the bus, into a normal PCI bus with 133 MBps; when we get to 10 Gbps network cards, the situation changes drastically, for it can rush through up to approximately 1250 MBps and only a PCI-X bus with throughput in the 1 GBps range can handle that, and barely at that. For the LVD-SCSI cards of interest to us, a 80 MBps SCSI card is the only one to still fit in the traditional 133 MBps PCI bus; a 160 MBps SCSI card already needs a 64-bit or 66 MHz PCI bus, with throughput of 266 MBps; and a 320 MBps SCSI card is only happy in a PCI bus supporting both 64-bit and 66 MHz, with 533 MHz throughput.

If one combines two fast cards, the requirements for the PCI bus may become more severe. In fact, one might think that a bus should have a data throughput compatible with the sum of the data throughput of all its cards. However, things are quite a bit more complicated than that, as the needed bus throughput may depend on many things, and the throughput actually needed usually turns out to be lower than that. A bus is much like a completely broadcast network, all devices see each stream of information as it goes along the bus in one direction or the other, choosing to accept or to ignore each data stream according to an addressing scheme. There is no switching and nothing like full-duplex operation, so a full-duplex Gbit card can cause traffic on the bus with in fact double its nominal throughput, that is, 250 MBps rather than 125 MBps. On the other hand, if a 160 MBps SCSI card has on it a single disk with a physical throughput speed of 40 MBps, then it is effectively limited to function at that lower throughput rate.

In addition to this, cards may or may not typically function simultaneously, thus having to share bus throughput. If two cards are not always being used at the same time, then there is no need for the PCI throughput to really be as large as the consolidated throughput of the cards. But if you consider that one of the main functions of a cluster server is to be a disk server for machines over the network, you see that it is inevitable that there will be a lot of simultaneous operation of the major I/O cards on the bus. On the other hand, if the data output to the network is originating from a single disk, then the physical speed of the disk may become the true limitation, specially if the data stream is long and there is little buffering or caching in memory involved in the process. This physical speed limitation for disks is usually much more severe than the limitation due to the throughput of the SCSI interfaces. Then again, if one uses the Linux software RAID driver to combine several high-performance disks together, then one can get a large effective disk with physical throughput compatible with the throughput of the interface.

One can see that it is not easy to predict exactly the bus throughput which will really be necessary for a particular machine, and that we must be content with approximate evaluations and decisions based on common sense. The conclusion of all this is that you should get the fastest PCI bus possible, within financial restrictions. Currently most brands have motherboard models with a few slots of 64-bit, 66 MHz PCI. It is a good

idea to get one of those for the server. To consider a typical case, consider a server with a Gbit network card plus a 160 MBps SCSI card; the combined throughput of the cards is about 285 MBps, which is compatible with a 266 MBps PCI bus. If most of the data traffic is from a set of fast disks on a RAID array to the network, then it will be limited to about 125 MBps for either card and we would have 250 MBps on the bus. But considering that other things may be going on in the system, it would be a lot more comfortable to have a 533 MBps PCI bus. If we have a front-end server with two Gbit cards plus a 160 MBps SCSI card, then the total consolidated throughput would be approximately 410 MBps and it would be a good idea to have the 533 MBps bus. In this case a considerable part of the network traffic could be from the nodes to the external network, passing right through the server and possibly causing in its bus a sustained throughput of up to 250 MBps, in addition to any disk traffic.

Total disk throughput on the SCSI bus should also match the throughput capacity of the bus. The physical throughput for sustained access to a SCSI disk depends on the rotational speed of the disk, as well as on its internal electronics. For a 10000 RPM SCSI disk it is in the range from approximately 35 MBps to 45 MBps. The number varies quite a bit depending on many circumstances, specially if you try to actually measure it on a busy server. The newer 15000 RPM SCSI disks are still much less common and less well known, but they are reported to have throughput in the range of 50 MBps to 70 MBps. Including the backup disks which will be discussed later, a typical cluster server will have four SCSI disks if it is also a home server. For disks in the above throughput ranges, a 160 MBps SCSI card is quite sufficient. For a front-end server with no home disks a 80 MBps card would be enough, but it is a good idea to leave room for an extra disk, so one should use the 160 MBps card here too. For the machine with 4 or more disks one would do well to use a 320 MBps SCSI card if possible. A very important detail is that all the SCSI disks should have a 68-pin SCSI connector rather than an 80-pin SCA connector, in order to fit the internal SCSI cable.

FIX THIS FOR LINKED USR

Nodes:	`/pmc/`	$S = 50 + N \times 1$
	`/pmc/var/`	$S = 100 + N \times 50$
	`/pmc/usr/`	$S = 1500 + N \times 20$
Terms:	`/trm/`	$S = 100 + N \times 2$
	`/trm/var/`	$S = 400 + N \times 100$
	`/trm/usr/`	$S = 5500 + N \times 80$

Table 3.1: Formulas for calculating the minimum disk space requirements in each filesystem, for clusters of compute nodes with up to 400 installed packages, and for clusters of remote-boot terminals with up to 2000 installed packages. Here S is the disk space in MB and N is the number of nodes or terminals.

The number and size of the disks you choose to have will depend on your needs and

on the role of the machine, of course, but for backup reasons they should always come in pairs. You should use only SCSI disks, in fact only LVD disks. All disks should be inside the system cabinet, so you should use only an internal LVD-SCSI cable, with a sufficient number of connectors (equal to the number of disks plus 2, one for the card and one for the terminator) and a LVD terminator at one end. The minimum size for system disks should be 18 GB, the current standard size for most systems should be 36 GB. In fact, hard disks are getting embarrassingly large very fast. Currently it is already difficult to find 18 GB disks and virtually impossible to find smaller ones. Fortunately they are also getting cheaper equally fast, so disk space should not be much of a problem from now on. If you want to use older disks you happen to have around, the smallest size you may still find some use for is 9 GB, below that they are not of much use at all. The necessary amount of disk space for each one of the disk partitions, in the cases of compute nodes and of remote-boot terminals, can be estimated according to the phenomenological rules shown in Table 3.1, where S is the disk space in MB and N the number of nodes or terminals. These formulas were conceived for very full systems, with up to 400 packages in the case of compute nodes and up to 2000 packages in the case of remote-boot terminals. If other system sizes are intended, then the sizes of the `/usr/` filesystems, which are the ones which are most affected by the number of installed packages, should be scaled appropriately.

FIX THIS FOR LINKED USR

48 nodes	`/pmc/var/`	2500 MB	`/pmc/usr/`	2500 MB
96 nodes	`/pmc/var/`	4900 MB	`/pmc/usr/`	3500 MB
144 nodes	`/pmc/var/`	7300 MB	`/pmc/usr/`	4400 MB
24 terms	`/trm/var/`	2800 MB	`/trm/usr/`	7500 MB
48 terms	`/trm/var/`	5200 MB	`/trm/usr/`	9400 MB

Table 3.2: Minimum disk space on the `/var/` and `/usr/` filesystems for a few typical clusters, assuming that clusters of compute nodes will have up to 400 installed packages and clusters of remote-boot terminals up to 2000 installed packages.

In the case of the root filesystems (`/pmc/` or `/trm/`) there is another criterion that has to be taken into account when choosing their sizes, which has to do with the sizes of the tables for name of files associated with the inodes of the filesystem. In practice it is usually safe to use 512 MB for both `/pmc/` and `/trm/` , as well as for the temporary filesystems `/pmc/tmp/` and `/trm/tmp/` . If one uses this size in the formulas given in Table 3.1 for the case of the root filesystems, one verifies that a 512 MB root can hold over 400 compute nodes or over 200 remote-boot terminals, numbers which are comfortably above the maximums we expect to have on a single server. In the case of the `/pmc/var/` , `/pmc/usr/` , `/trm/var/` and `/trm/usr/` filesystems one can use these formulas to verify that for most typical clusters of either compute nodes or remote-boot terminals these filesystems do fit comfortable within either 18 GB or 36 GB disk drives,

as the case may be. Some of these typical cases are shown in Table 3.2 and examples of complete partition tables of the disks will be given in later sections. One should note that the root, `/var/` and `/usr/` filesystems of the nodes must be formatted in a special way, as will be seen later.

Going back to the throughput issue, we may conclude that although a 80 MBps card could suffice for small servers, the standard for small and large servers should be 160 MBps and 320 MBps respectively. Irrespective of their physical throughput capabilities, all disks on a SCSI bus should have interface throughput speed compatible with the speed of the card, and they should all be LVD. One should not use older 16-bit 40 MBps single-ended disks on a LVD bus, because that will force the bus and all disks on it down to the 40 MBps speed. If you need to install in the server SCSI devices with either older 16-bit or 8-bit interfaces, there are SCSI cards with two independent SCSI busses, a 16-bit LVD bus for the disks and another slower one supporting either 16-bit or 8-bit devices which can be used for CD units, ZIP floppy disk units, etc.

However, it is not to be recommended that you have any such I/O devices on the server, if it is possible to avoid that. They will work perfectly well on a remote-boot terminal in a public room and it would be much better to have them there, where the users can actually get at them. Besides, in that case you can use IDE devices, which are much less expensive and quite performatic. This is so because you should never use any IDE devices on a server, since they can cause severe interruptions if intense I/O is done on them, slowing the machine down to a crawl. Due to this, on the server you should have no IDE devices at all, neither disks nor CD and ZIP units. In the near future it will be possible to deal with the I/O interruption problem in a desktop system such as a remote-boot terminal, by the use of the "preemptible kernel" option which exists in version 2.6 of the Linux kernel. However, this option should not be used on servers, since it will be detrimental to the general I/O performance of the system.

There is a new type of IDE disk which may turn out to be usable on small servers, called SATA disks. While de traditional IDE bus uses a somewhat SCSI-like protocol called ATA, the new disks use a new protocol called Serial ATA or SATA, which comes closer to the SCSI protocol. This new kind of disk drive is unique in that the communications mode is, as implied by the name, serial (one bit at a time) rather than parallel (a certain number of bytes at a time) such as the usual IDE (1-byte) and SCSI (1-byte or 2-byte) devices. Whether or not this new type of disk will perform sufficiently well to be used in servers without long bus interruptions is still rather unclear. Besides, although it is already possible to get motherboards with SATA devices on-board, the disks themselves are still rather expensive. Another disk interface still in development is called Fiber-Channel, meant to eventually replace SCSI. Maybe in the future we will see a time with lots of Fiber-Channel and SATA devices, but for the time being what we have is SCSI for servers and IDE for other uses, particularly in their faster versions SCSI-160, SCSI-320, ATA-100 and ATA-133.

While the motherboard and the Ethernet and SCSI cards and disks are the most

Motherboard:	Tyan model Tiger MPX S2466 dual-processor, with 4 DDR memory slots, with two 64-bit, 66 MHz PCI slots.
Processors:	2 Athlon MP 2400+, with ≈2.0 GHz each, with appropriate heat sinks and cooling fans.
Memory:	DDR with 1 GB total, in two 512 MB DIMM's, for a 266 MHz memory bus, registered ECC as required by the motherboard.
Network card:	3COM model 3C996BT, 1 Gbps, for a 64-bit, 66 MHz PCI slot.
SCSI card:	Adaptec model 29320-R, with 320 MBps, for a 64-bit, 66 MHz PCI slot.
SCSI cable:	LVD, internal, with 5 free 68-pin connectors and an LVD terminator at one of the ends.
System Disks:	2 Seagate model Cheeta-15K, with 36 GB each, 15000 RPM, 320 MBps SCSI interface, 68-pin SCSI connector.
Home Disks:	2 Seagate model Cheeta-15K, one with 36 GB and one with 72 GB, 15000 RPM, 320 MBps SCSI interface, 68-pin SCSI connector.
Cabinet:	Enlight 8000-series, full tower, 4 fans, floppy cable, power cable, 7 front bays with 5.25-inch width.
Power supply:	Single unit with 450 W or more, server-grade, with a second identical unit on the shelf as backup.

Table 3.3: Typical discrimination of the important parts of a standard cluster server.

important parts of the server, you will also have to get all the other necessary parts, such as: a standard US (QUERTY) PS/2 keyboard; a three-button mouse, preferably a serial one, which is less prone to serious complications, and which will be used in console mode only; a standard 1.44 MB floppy drive unit to be used sometimes for booting purposes, with a floppy cable; any small VGA monitor, a 14-inch or 15-inch one will suffice; power cables, etc. If you have several servers side by side, you can use a single monitor for all of them by means of a simple VGA switch to connect the monitor to each server in turn. This may not be a very important thing price-wise, but it is a valuable space saver inside your server room. However, it is safer to use separate keyboards for each server, for the lack of a connected keyboard can prevent a server from booting automatically after a power failure or from coming back after a remotely-issued system reboot. Each server should have its own mouse too. In this way, and also due to the lack of X11 servers, a very simple and cheap VGA switch may be used, a mechanical one without support for either keyboard or mouse.

One important item not yet examined is the cabinet for the server. This should be

Motherboard:	Soyo model SY-KT333 Dragon Lite single-processor, with 3 DDR memory slots, with a 32-bit, 33 MHz PCI bus.
Processor:	1 Athlon XP 2400+, with ≈2.0 GHz, with appropriate heat sink and cooling fan.
Memory:	DDR with 512 MB total, in two 256 MB DIMM's, for a 266 MHz memory bus.
Network card:	3COM model 3C2000, 1 Gbps, for a 32-bit, 33 MHz PCI slot.
SCSI card:	Adaptec model 29160N, with 160 MBps, for a 32-bit, 33 MHz PCI slot.
SCSI cable:	LVD, internal, with 5 free 68-pin connectors and an LVD terminator at one of the ends.
System Disks:	2 Seagate model Cheeta-10K, with 18 GB each, 10000 RPM, 160 MBps SCSI interface, 68-pin SCSI connector.
Home Disks:	2 Seagate model Cheeta-10K, one with 18 GB and one with 36 GB, 10000 RPM, 160 MBps SCSI interface, 68-pin SCSI connector.
Cabinet:	Enlight 8000-series, full tower, 4 fans, floppy cable, power cable, 7 front bays with 5.25-inch width.
Power supply:	Single unit with 300 W or more, server-grade, with a second identical unit on the shelf as backup.

Table 3.4: Typical discrimination of the important parts of a minimal cluster server.

a good-quality, well-built and large cabinet, for the physical stability of the server will depend on it. The recommendation is for a full-tower ATX cabinet, with a 450 W or larger ATX power supply, with at least 5 but preferably 7 front bays with 5.25-inch width, and independently removable side panels. There should also be a floppy drive bay and several internal 3.5-inch bays. It should also have several ventilation fans, for fast disks and fast CPU's dissipate quite a bit of heat. On the other hand a redundant power supply is very expensive and not necessary, a good-quality normal server-grade power supply will do, for redundancy you can keep another identical one on the shelf. Such a good-quality cabinet might cost four or five times as much as a very cheap one, but it is well worth it if you want a very stable and reliable server.

Finally, let us look at parts lists for a couple of basic examples of servers. In Table 3.3 you will find a possible current discrimination for a standard cluster server which acts also as a home server. It can be used to serve up to 48 remote-boot terminals or up to 96 compute nodes, possibly even more in the latter case. An important example would a be minimalistic server for a small system, with 24 or less remote-boot terminals, or with

Video card:	any good-quality low-end VGA card for AGP.
Monitor:	any standard 14-inch or 15-inch color VGA unit.
Floppy drive:	any standard 1.44 MB, 3.5-inch unit.
Disk brackets:	four pairs, for mounting 3.5-inch disks on 5.25-inch bays.
Extra fans:	standard 80×80 mm (3-inch) cabinet fans and special screws.
Keyboard:	any US-standard QUERTY, with a PS/2 connector.
Mouse:	Logitech, serial interface, with 3 buttons.

Table 3.5: The trivial parts common to both the standard and the minimal cluster servers.

48 or less compute nodes. This can be seen in Table 3.4 and establishes the minimum one should get for a cluster server. In this way one may reduce considerably the typical price of the hardware and still have a small but well-configured cluster. For example, it is currently possible to get a good Gbit network card for a 32-bit, 33 MHz PCI bus for approximately US$ 55, while the corresponding card for a 64-bit, 66 MHz PCI bus will cost approximately US$ 135. Other items will also cost less, although the price ratio might not be as large as that.

In Table 3.5 one can find the list of trivial parts which are common to the two server discriminations. Note that, although the technical requirements on the video card are very light, it should nevertheless be a *good-quality* card, for a bad or defective card can even cause your machine to hang. It is a good idea to pick a low-end model of a well-regarded brand. The pairs of disk brackets listed there are for mounting 3.5-inch disks on the 5.25-inch front bays of the cabinet. It is better to do it this way instead of using the internal 3.5-inch bays because it improves disk ventilation. The extra cabinet fans are listed in case the cabinet does not include all the fans that can be mounted on it. You must also get special self-threading screws for them, four for each fan. All fan mounting positions available on the cabinet should be used.

If one examines the numbers in the technical specifications of these server discriminations, one will find that the PCI bus of the minimal server comes out a bit short in throughput. However, experience does show that one can have a small cluster with a server like this, without any noticeable I/O problems. This is due to the fact that the most restrictive limitation is the physical throughput of a single SCSI disk. One will also note that the numbers of the standard server seem to warrant more than 48 remote-boot terminals. Again the limitation here is the physical speed of SCSI disks. One may double the number of terminal if one adds to the configuration a pair of 18 GB, 15000 RPM disks to act as system disks for the additional terminals.

Either server can be used as the front-end server of a PMC with the addition of an extra network card for the external network, either at 100 MBps or at 1 GBps. However, with a Gbit card it would be better to get a motherboard model with three rather than two fast PCI slots, in order to guarantee good communications performance between

the private network and the external network. Alternatively, if the machine will not act as a home server and hence will not have the home disks, one can use the two fast PCI slots for the two network cards and downgrade the SCSI sector to a 160 MBps card on one of the 32-bit, 33 MHz PCI slots available on the motherboard. However, it would still be necessary to keep the 15000 RPM disks in the configuration.

We end by noting that, although the servers described here were designed as cluster servers, they are not really too different from any normal workgroup server, that is, from any disk and network server meant to serve a certain collection of users and machines. Therefore, one might want to follow the ideas and guidelines described here also if one is either designing or purchasing a workgroup server, not involved with any cluster of remote-boot terminals or compute nodes for a PMC.

3.2 Putting The Pieces Together

Having solved the hard problem of deciding what to buy for your server, now you have ahead of you the possibly pleasant and fairly easy task of assembling the machine and making it operate. If you dislike doing this or are otherwise prevented from doing it, you might find a company that will either assemble it for you or deliver it to you assembled according to your detailed specifications. Usually there are many small enterprises around that do this kind of work, the only remaining problem being to choose a technically competent one that you are sure you can trust.

Assembling the machine is not difficult, but it is fairly delicate work that one should do rather carefully, with plenty of attention to detail. We will give here an assembly sequence guide in proper order, drawing attention to several important points and pointing out things one should be specially careful about, because they involve risk of serious damage to the hardware. The details in this description are based on the parts listed in the technical discriminations used as examples in the previous chapter, but the general procedure is essentially always the same.

If you decide to do it, you will need only a few very basic tools in order to perform this task. Basically, a normal number 2 phillips screwdriver, a pair of average-size pliers and a pair of cutting pliers is all that you will need. However, a pair of narrow-point pliers and a set of small screwdriver-driven hex socket drivers might also come in handy. The relevant sizes for the sockets are 3/16-inch, 7/32-inch and 1/4-inch. A screwdriver set, with interchangeable bits of all sorts, including slot bits, phillips bits and sockets, is a nice way to have all this in one handy package. The screwdriver itself should be magnetized in order to facilitate handling the screws, and do not worry, it will not do anything to your magnetic disks.

You will also need a few supplies, such as a can of contact-cleaning spray, paper towels and a number of self-locking nylon bracers of several different sizes, going from 2 mm width by 80 mm length to 5 mm width by 200 mm length. You may also need a small quantity of white thermal paste, which you can get in an electronics shop, in case the heat sinks you got for the CPU's do not come with that included. The cabinet should come with a small amount of hardware included, containing specific things such as rails for mounting devices on its bays and generic things such as screws. You might need a few extra screws of several different types in case the cabinet does not come with all that you need.

You should also have handy the hardware manuals that come with the parts you purchased, one for the motherboard, one for the SCSI card and one for each model of SCSI disk. Usually all the necessary information contained within these manuals can also be obtained from the Internet, in case you did not get a hard copy of them. Before you start, here is our first technical warning. Every time there is some risk of damage to the hardware involved in the task at hand, we will include in this section a warning formatted as follows.

> **WARNING:** in dry environments static electricity can be a serious hazard for your machine. *Electric discharges caused by static electricity can damage the delicate electronic circuitry contained in the components of your machine beyond repair!*

This can be a problem in particularly dry climates, in rooms with air conditioning and carpets, etc. In order to prevent this kind of damage, be sure to ground things, for example the cabinet you are working on. Touch frequently a grounded metallic surface in order to discharge your body. Do not touch electronic parts and cards on their golden-plated contacts, hold them by their other edges or by their metallic face plates. Be careful when you take the boards and cards that you purchased off their anti-static bags and handle them.

You start by opening the cabinet, both metal side covers should come off, as well as the plastic front cover. Lay the cabinet on its side over a table, with the side you will mount the motherboard on facing up. If it is not already in place, you can mount the power supply on the cabinet. The power supply should be a server-grade one, with many cables to power the disks and other devices, and preferably two ventilation fans, one facing outside and another one facing inside the cabinet. You will see that both are oriented so as to force air *out* of the cabinet.

> **WARNING:** check the appropriate switch on the power supply to set the line voltage to 115 V or 230 V as appropriate. *Incorrectly setting it at 115 V and plugging it into 230 V will burn things out when you try to power up the system!*

Due to this the power supplies generally come from the factory set to 230 V. Having the power supply set to 230 V and plugging it into 115 V will make LED's (light-emitting diodes) light up but the motherboard will not operate and probably give you a bit of a fright, but will not actually damage anything, except maybe your state of mind.

Mount the 80 mm fans on the cabinet, if they are not already in place. Use all available fan positions on the cabinet. All cabinet fans should be exhausting the air, sending it out of the cabinet. If some are not, reverse them. You can verify the direction in which they send the air by checking the curvature of the blades. Use the power supply fans as a reference, they always send the air out of the system cabinet. Until you assemble the motherboard you cannot turn the ATX power supply on, but if you have an older AT power supply laying around, or some older machine you can open and play around with, you can temporarily connect the fans to one of them to check the direction of the air flow. You may now also mount the floppy drive on the special bay reserved for it in the cabinet.

At this point you should check whether or not your motherboard needs any kind of hardware configuration by means of straps or dip-switches existing on it. In most cases modern motherboards will auto-detect correctly the CPU and RAM that you put on

them, but there might be one or two straps to set in some cases. This is one time when you need to read the hardware manual that comes with the motherboard. There might be pairs of contacts on the motherboard which can be closed by a strap, and there may be a block of very small switches called dip-switches.

In the case of dip-switches, there is usually a setting for automatic detection of the CPU type and frequency. It is usually the factory default and you should keep it unless your motherboard does not detect correctly your CPU. The dip-switch functions may include the selection of bus frequencies, CPU voltages and CPU frequency multipliers. Straps may be used to enable or disable on-board sound chips and other devices, to select bus or memory frequencies, to activate or deactivate an on-board beeper, and other things. You should read the motherboard manual and keep them all in their factory defaults unless there is some clear reason to change something.

One important strap that should always be there is used to clear the NVRAM (non-vanishing random access memory) of the motherboard. It is usually labeled "CLEAR CMOS" and must be used if you forget the password you define for access to the BIOS of the machine. It is also useful if it happens that the motherboard gets so badly misconfigured that it will not even complete the hardware self-test cycle (POST or power-on self test) when you turn it on, probably never getting to the point when there is video signal and something shows up in the monitor.

You are now ready to mount the motherboard on the cabinet. In order to do this you will have to first screw to or mount on the cabinet a set of supports for the motherboard. These are typically small hex-shaped metallic parts that come with the motherboard. They have a screw in one end and a threaded hole in the other, you screw them to the cabinet and the motherboard is screwed down on top of them. There may also be other types of metallic parts that you mount to the cabinet without screws. In any case, you must use the parts that come with the cabinet, or others exactly the same height, otherwise the motherboard will not be correctly positioned within the cabinet. If it is too high or to low with respect to the expansion card slots in the rear of the cabinet, the expansion cards will not fit properly.

Not all motherboards have all the mounting holes in the same places, so cabinets have many mounting holes or positions and you must choose those appropriate for your motherboard. But do use all mounting holes existing on your motherboard. These are the holes with a metallic solder ring around them, holes without this are for ventilation or other purposes and should not be used. Sometimes there are a few holes in the corners of the motherboard that do not fit any mounting holes on the cabinet. These may not have solder rings around them and may be meant for inserting plastic supports that come with the cabinet. These are meant to seat against the metal plate of the cabinet to better support the edges of the motherboard.

> **WARNING:** motherboards have many small electronic parts soldered to their upper surface, including around the screw holes. *Be very careful as you screw down the motherboard, in order not to damage any parts near the*

holes!

When screwing down the motherboard, do not use washers or insulators on the screws, they should make contact with the solder rings around the holes. You may use normal cabinet screws, but they should have smooth seats, for as their seats press against the solder rings they might damage them if they are not smooth. Do not use the screws with dented seats that one usually finds in many cabinets. Use your phillips screwdriver to tighten the screws, but do not tighten them too much, a firm but moderate tightening will do, there is no need for more than that.

Here we must start a small digression to explain all about how one assembles a LVD-SCSI bus. The bus is formed by the internal SCSI cable and a terminator at each end. One LVD terminator will actually be at one end of the cable, either crimped to it or inserted in a connector. The other one will be on-board the SCSI card which will be connected to the other end of the cable. In between, several connectors along the cable allow you to connect SCSI devices to the bus. Each device must have a unique SCSI address associated to it so that it can be identified within the bus. For a 16-bit SCSI bus such as a LVD bus the SCSI address is a 4-bit integer going from 0 to 15, so one can have at most 15 devices on the bus, besides the SCSI card. Usually address 7 is associated to the card itself, since from the point of view of data traffic it is just another device on the bus. Address 0 is usually associated to the disk the operating system will boot from. The other three disks will have addresses 1, 2 and 3, respectively.

In the older versions of the SCSI protocol you could have a terminator on-board the last disk on the bus, instead of having it at the end of the cable, but this is not the case for an LVD bus. Since most disks are backward-compatible with the older protocols, they will usually have a strap to activate the internal terminator. You should *not* activate this strap because, if you do, the disk and the whole SCSI bus will automatically go down out of LVD mode into single-ended mode and a speed limitation of 40 MBps will be enforced. There might also be a strap to directly force single-ended operation, which you should also leave alone. Other straps that you should leave alone may disable parity checking, put the disk in read-only mode, synchronize spindles and other things.

Usually SCSI disks will turn on their motors by default when powered up. If you have several disks on a machine, that can subject its power supply to quite a bit of stress when you power up the system and all disks start up at once. In order to avoid this you can configure the disks to not start their motors when powered up, but instead to wait for a start-motor command from the SCSI card or the operating system. The SCSI card may scan the bus and start up all the disks, one at a time, so as not to over-stress the power supply. If the SCSI card does not do that, the operating system will, during boot. The disks have a strap to enable this feature, labeled with something like "delay motor start", "disable motor start" or some equivalent description. Unfortunately the words used vary and sometimes are quite obscure, so you will have to read the hardware manual of each type of disk to figure things out. For example, the strap could be labeled as "enable motor start" if what is meant is enabling the starting of the motor *by the*

Address bit				SCSI	Address bit				SCSI
3	2	1	0	addr.	3	2	1	0	addr.
□	□	□	□	0	■	□	□	□	8
□	□	□	■	1	■	□	□	■	9
□	□	■	□	2	■	□	■	□	10
□	□	■	■	3	■	□	■	■	11
□	■	□	□	4	■	■	□	□	12
□	■	□	■	5	■	■	□	■	13
□	■	■	□	6	■	■	■	□	14
□	■	■	■	7	■	■	■	■	15

Table 3.6: The SCSI address bits and the corresponding SCSI addresses. The white rectangles represent open contacts and the black ones represent contacts closed by straps.

SCSI card.

> **WARNING:** disks are sensitive to physical shock, handle them with great care. This is particularly the case if they are in operation, *even a moderately severe physical shock during operation can completely destroy a disk drive!*

Do not make any jerky movements with the system cabinet while it is powered up. Do not subject it to bumps or move it around too much while it is powered up. Hard disks are sometimes damaged by shock during transportation for delivery, check them out when you receive them, they should arrive in special padded packaging for protection against shock. Handle them with care even if they are not powered up, avoid dropping them or banging them against other hard objects. In short, be gentle with them.

You should now configure each one of your SCSI disks by strapping them. Read the hardware manuals and consult the tables written on labels on the disks themselves. In each disk you should do exactly two things: set the SCSI address and disable the automatic motor start. The SCSI address is controlled by a set of four adjacent straps, each one representing a bit in a 4-bit binary number. These straps may be labeled with the numbers 0, 1, 2 and 3, or with the corresponding powers of 2, that is, with 1, 2, 4 and 8. Inserting a strap in one of these pairs of contacts means turning on the corresponding bit, from 0 to 1. In Table 3.6 you will find a list of the possibilities for the set of four bits and the corresponding SCSI addresses in decimal notation. The disks listed in our examples will be addressed as follows:

Address 0:	system (boot) disk,	18 GB or 36 GB.
Address 1:	backup system disk,	18 GB or 36 GB.
Address 2:	home disk,	18 GB or 36 GB.
Address 3:	backup home disk,	36 GB or 72 GB.

You may now mount the disks on the brackets, so that they may go into the 5.25-inch front bays. If the cabinet has rails for inserting devices into these bays, mount the rails on the brackets once these are attached to the disks. Adjust both brackets and rails so that the disks are well aligned with each other and so that they slide nicely into their bays, locking securely in place. You can insert the disks into the bays of the cabinet from the front. You should use adjacent bays, because the connectors in the SCSI cable may be mounted quite close to one another along the cable. You should use the lower bays, those closer to the motherboard, because the length of the SCSI cable usually is not very large.

The disks should not touch each other and there should be small spaces between all pairs of adjacent disks, to allow for good ventilation. The disks should always be mounted right-side-up, with their electronics cards facing *down*, towards the motherboard. The boot disk (with SCSI address 0) should be the one closer to the card, along the SCSI cable. The disks should be mounted in the cabinet in SCSI-address order, both along the cable and physically in the cabinet. The physical order will depend on how the connectors are mounted on the SCSI cable. They should all be crimped to the cable with the same orientation. These connectors are keyed so that they can only be inserted in a definite way. Look at the connectors on the cable and on the back of the disks to figure out whether the boot disk will be on the top or on the bottom of the set of disks.

It is now time to assemble the processors onto the motherboard. Be very careful while you handle the chips, hold them by the edges and do not touch the golden-plated contacts they have in one side. Do not drop the chips and do not force the contacts in any way. Should these contact get even slightly bent, the chips will no longer fit into the sockets. Since there are so many contacts, getting them straight again is a hard job. On the other side of the chips, at the center, you will see small raised and polished surfaces, with approximately 1 cm by 1 cm. This small spot is where the integrated circuit containing the CPU in fact is, the rest of the chip contains just the outside contacts. It is with this small polished surface that the heat sink will make thermal contact.

The CPU sockets have a small lever on one side, which you have to lift in order to insert them. The chips can only be inserted into the sockets in a single definite position and there is no force at all involved in doing this. Look at the pattern of pins on the chip and of holes on the socket to figure out the correct orientation. Having lifted the levers, before you insert the chips in place you should spray both the contacts on the chips and the holes in the sockets using your can of contact-cleaning spray. Then immediately insert the CPU's in the sockets and lower the levers to lock them in place. Before you lower the levers, make sure the chips are not tilted, but seated evenly and firmly over the sockets.

The contact cleaning is a precaution we will take in all electrical connections which carry high frequencies. It is worth remembering that, in the case of the CPU's, there

will be frequencies in the GHz range going through these contacts, which are very high indeed. Next you must coat the central polished surface of each CPU with the white thermal paste that comes with the heat sinks. It should be a thin and homogeneous layer distributed evenly over the raised surface. The function of the thermal paste is to ensure good thermal contact between the heat sink and the CPU chip. The pressure of the heat sink over the coated surface will squeeze the paste into a very thin layer, making the thermal contact even better. Sometimes the heat sink comes with a patch of special rubbery material on the spot where the chip touches it. This is another way to ensure good thermal contact, but since it tends to be thicker you may want to scrape it off and use the paste instead, although this is not imperative.

> **WARNING:** do *not* turn on the power of the motherboard with the CPU's in place while the CPU heat sinks and their cooling fans are not in place and properly connected. *Powering up the machine without the proper CPU cooling devices in place may burn out the CPU's!*

You may now mount the heat sinks over the CPU chips. Make sure they are in the correct position, otherwise they will not seat properly over the center of the CPU chips, may get tilted and loose good thermal contact. They are supposed to be mounted over the CPU sockets only in a certain position, but sometimes it may be possible to reverse them and still be able to clamp them in place in the wrong position. The metal clamp that they have is to be attached to the CPU socket on two opposite sides. It has a spring action and after one side is attached these is considerable force involved in attaching the other side. This other side of the clamp generally has a small slot where you can fit the tip of a common slot-type screwdriver in order to force it into place. *Be careful while you do this, lest the tip of the screwdriver fly off over the motherboard and damage something.*

When you are done with mounting the heat sinks you can connect the CPU fans, which are attached to them, to the appropriate 3-pin connectors on the motherboard. These connectors should be labeled as "CPU FAN" or "CPU FAN 1" and "CPU FAN 2", and should be keyed to allow connection only in the right way, but in any case be careful not to reverse the connections since this may damage the fans. Be sure that these fans are connected to the correct connectors, since there may be several other such 3-pin fan connectors on the motherboard, to be used for other fans such as cabinet fans and power supply fans. These fans have an internal rotational speed detector and the motherboard will read its signals. An incorrect connection may trigger a CPU fan alarm in the motherboard, possibly (but not usually) even preventing it from powering up at all.

Next you can mount the memory modules on the motherboard. The memory or DIMM sockets are easily recognizable, there should be three of four of them side-by-side on the motherboard. They should be numbered as "DIMM 1", "DIMM 2", etc. Usually you can insert your memory modules in any of the sockets, but in some cases it may

be necessary to use the first socket first, and so on. If in doubt, read the motherboard manual for the details. The modules must be inserted in the right way, observe that there are notches along the contact edge of the DIMM's, which should fit over corresponding bumps on the sockets. The positions of these notches and bumps are different for each different type of memory, so that you should not be able to insert the wrong kind of memory into a socket. There are plastic holders in either end of the sockets, which you have to open wide in order to insert the modules.

Examine everything carefully and make sure you understand clearly what you have to do before you proceed. Before you insert the modules you should spray both the sockets and the golden-plated contacts of the DIMM's with your contact cleaner, them immediately insert the DIMM's into the DIMM sockets, taking care to orient them in the right way. In order to insert the DIMM modules apply pressure vertically at the two ends of the module, in a direction perpendicular to the motherboard. Alternate the pressure at each end in a kind of rocking motion until the modules snap into place at both ends.

> **WARNING:** make sure that you insert the DIMM memory modules the right way and that they are fully inserted in their sockets. *Inserting a memory module the wrong way will burn out the memory module and probably damage the socket beyond repair!* The damage occurs instantly when you power up the system.

The plastic holders in either end of the sockets will snap in place when the modules are completely inserted. Although the various types of DIMM memory module sockets are keyed to allow insertion only in the right way, some lower quality sockets may still permit some electrical contacts to close even if a reversed module is not completely inserted, and hence give room for this kind of disaster.

Now is a good time to connect the power cable to the motherboard. This will be an ATX connector with many wires on it. The connector is keyed, can only be inserted in the right way into the corresponding connector on the motherboard, and has a clip on one side which will latch on to the connector on the motherboard. Make sure the connector is inserted fully, until the clip latches on. Sometimes the motherboard will need a second, auxiliary power connection. This is seldom the case for single-processor motherboards and often the case, in fact almost always the case, for dual motherboards.

There are three possible kinds of auxiliary power connectors, the most common kind is a small 4-pronged connector similar to the ATX connector, also keyed and also with a clip. Another type is a connector like the old type of AT motherboard power connector, but with only one connector instead of two side-by-side connectors of an AT motherboard. The third type is a common power connector like those used to power disks and other devices. All these connectors are keyed. Of course the power supply must have the correct type of connector for your motherboard in case it needs an auxiliary power connection. Usually this is not a problem since server-grade power supplies usually have all three types of connectors.

You may now also connect the cabinet front-panel control wires to the motherboard. These are the following: the power LED connector, the speaker connector, the power switch connector, the reset switch connector and the disk activity LED connector. Each of these should be so labeled on the connectors themselves, each at the end of a pair of wires in the cabinet. The corresponding contacts should be similarly labeled on the motherboard also, in case of doubt read the manual of the motherboard, where they should be described. The pairs of wires are colored and in some cases connection polarity is important. Information about this should also be in the motherboard manual. There is one exception here: since you will be using SCSI rather than IDE disks, the disk activity LED should be connected to the SCSI card rather than to the motherboard, so you will do this part in a little while, once the SCSI card is in place.

The next step will be to mount the expansion cards on the motherboard in the cabinet. In order to mount the expansion cards in the cabinet, you will have to remove a few slot covers. Before you do that, consider the order and spacing of the video, SCSI and network cards. The video card has its place predetermined, since there is only one AGP slot on the motherboard. The AGP slot is usually brown and is the one closest to the CPU sockets. The SCSI card should be closer to the video card, the network card (or cards) below it, away from the video card. It is a good idea to space the cards a bit, if possible, because some of the cards may heat up quite a bit. Hence, try to leave empty slots between all pairs of adjacent cards, in order to allow for better air circulation and cooling. This is specially the case for the video card if it has a fan on its chipset.

> **WARNING:** if your cabinet has the kind of expansion card slot cover that you have to bend and twist until they break off, *be very careful not to damage the motherboard while you do this!*

There may be many very small parts soldered to the motherboard close to these snap-off covers and you can easily rip these parts off if you are not careful with your wiggling and twisting. Be patient and twist the covers slowly and only a little bit, repeating the movement as many times as necessary for them to break off.

You may now insert the cards into their slots, in order. Make sure that each card is well seated both into the socket on the motherboard and into the slot on the back of the cabinet. In the case of the AGP card checking for full insertion may be a bit tricky, because the AGP slot if deeper than the others, with two layer of contacts, and the card tents to "click in place" twice instead of just once. Examine the card carefully and make sure it cannot be inserted any further into the socket. At this point you should verify that, due to the use of the correct mounting hardware, the motherboard is indeed in its proper position, positioned correctly with respect to the slots on the back of the cabinet. If by any chance this is not the case you will see that the cards cannot be fully inserted and screwed down in place while having their edges perfectly parallel to the motherboard.

> **WARNING:** in case your motherboard still has some ISA slots, beware that *mistakenly inserting a PCI card on a ISA bus slot will burn the PCI card out!* The damage occurs instantly when you power up the system.

You can easily recognize the ISA slots since they are larger than standard PCI slots, with fewer and larger electrical contacts. They are usually black or some dark color, while standard PCI slots are usually white. Usually the ISA slots will be near the lower edge of the motherboard, besides the standard PCI slots and away from the fast PCI slots, which are longer that the standard ones. Sometimes fast PCI slots are colored green, blue or purple.

Before you screw the cards down to the cabinet by their metallic face-plates, check to see whether the screw holes are well aligned within the openings on the face-plates. If they are not, indicating a cabinet a bit out of proper mechanical standards, a little bending outwards of the upper part of the face-plates might be enough to correct the situation. You should make sure that tightening down the screws does not produce a tendency to pull the card up from the socket.

You may now connect the power and data cables to the floppy drive. Start with the power connector, which is a small connector dangling from your power supply, probably connected in parallel to two other connectors of the larger type, which are used for hard disks and other devices. Locate the power connector at the back of the floppy drive unit, it consists of four contacts.

> **WARNING:** remember that *inverting the power connection of a floppy drive will burn out the drive!* The damage occurs instantly when you power up the system.

Usually the power connectors are keyed and can only be inserted in the correct way, but sometimes it is possible to inadvertently reverse the connection, bending a bit the contacts on the floppy drive.

For the data connection to the floppy drive you will use the floppy flatcable that probably came with the cabinet. It might have several connectors on it, but you will use only the two connectors at or near either end. Near the one that goes on the floppy drive there will be a twist in a section of the cable, reversing a few contacts. This twist marks the drive connected to that connector as the first floppy drive. The data connector of the floppy drive is very seldom properly keyed, but reversing it will not burn anything. It may, however, prevent your machine from doing its POST cycle, or from doing it correctly to the end. Check whether you can see the pin number 1 marked somewhere near the connector on the drive. The wire on the flatcable corresponding to it generally has a red stripe on it. The floppy connector on the motherboard is usually keyed with a slot, but the connector on the cable may not be. However, usually the connector on the cable, whether or not it is keyed with the appropriate plastic bump, will also have on the same side a couple of groves, while the other side is completely smooth. You can use this fact to insert it in the correct way.

You may now connect the power and data cables to the disks. The power cables should be chosen only on length considerations, so that there are cables able to reach all devices, including the cabinet fans as the case may be. The power connector might fit a bit tightly on the disks, and you should make sure that they are completely inserted. The power connector on the disks is held in place by its contacts soldered on the electronics board of the disk. It might be a bit hard to take the power cables off again, once they are inserted. If you find you must make a rocking motion in order to loosen the power connectors, *make sure you do this from side to side* (parallel to the electronics board) and *not up and down* (perpendicular to it), in order not to risk breaking their solders on the disk board. Connect also the disk activity LED to the appropriate contacts on the SCSI card. Read the manual of the SCSI card in order to locate them on the card.

Connecting the SCSI data cable in order to assemble the SCSI bus is more delicate work. The SCSI connectors are delicate, having a high density of connections, you must be careful not to bend any of the pins of the male connectors on the cable. Be gentle but do not be afraid, insert the connectors gently but firmly into their sockets. The SCSI cable is also more delicate and should be treated with care. Since there are fairly high frequencies traveling on the SCSI bus, it is a good idea to use your contact cleaning spray on both connectors before you connect each pair together. Remember that the disks should be in order, with the boot disk connected to the first connector on the cable, closer to the SCSI card.

You may now connect the power of the cabinet fans, they are the last thing remaining to connect. These fans usually are connected directly to the power supply, not to the motherboard, but there are models that you can connect to the motherboard if there are enough 3-pin fan connectors on it. If they are to be connected to the power supply, they probably are attached to a power extension, with a male and a female plugs, so you can connect them to each other or to other devices and use only a single power connector from the power supply for the whole set. Using this you will probably find that you do have enough power connectors on the power supply, while otherwise there would not be enough connectors. Check also whether your power supply has wires with a 3-pin fan connector coming out of it, it might be that its internal fans can also be connected to the motherboard. If so, connect it to a 3-pin connector on the motherboard labeled with "PRW FAN" or something like that.

Having completed the whole assembly and made all the connections, it is now a good idea to check the operation of the machine before you close the cabinet. For this test you do not need the mouse, so connect to the cabinet only a keyboard and a monitor, take a deep breath and power up the system. Besides connecting the power cable of the machine to a power outlet, you may have to press the power-up button for it to actually turn on. There is a sequence of things that you should check for:

- Whether or not lights come on; there are LED's on the cabinet, the floppy drive, the network cards, the SCSI card, and possibly on the motherboard and on the disk drives.

- Whether or not the fans start; there are fans on the CPU's, on the power supply and on the cabinet; check also the direction of the air flow they are causing.
- Whether or not there is video signal; after a little while a LED on the monitor should change color, turning from orange to green, indicating the reception of video signal.
- Video image: after the monitor warms up an image should star to show up in it; messages should become visible, showing the motherboard going through its power-up tests.
- The BIOS of the SCSI card should start to run, detecting the disks and starting their motors one at a time; you may be able to hear the motors starting; check the SCSI ID's (addresses) reported.
- Some other BIOS might run too, if it exists in some part of the machine, for example a remote booting BIOS on the network card, which you may have to disable later.
- The power-up test cycle ends and, assuming your disks and floppy drive are empty, and that there is no remote-boot server configured for this machine, the BIOS stops, reporting that there is nothing to boot.

If you got to this point all is well with your machine. Since the monitor was not yet warm enough for its image to be visible at the beginning of the process, you may have missed some of the initial messages of the power-up cycle. You may want to reboot the BIOS pressing the combination of keys `[Ctrl]-[Alt]-[Del]`, for example to see whether the video card, the CPU and the memory are being detected correctly by the BIOS. You should also verify that the power and disk activity LED's are lighting up, that the speaker or beeper does beep, and that the reset and power buttons do function as expected: the reset button should restart the power-up cycle and the power button should shut down the power supply if pressed during more than 4 or 5 seconds.

On the other hand, if the machine stops at some intermediary stage in this process, there is something wrong and you should go back and check your whole assembly sequence. Here are some common problems followed by their possible causes and solutions:

The machine simply does not turn on at all. You may have a burned-out power supply; test the voltages on one of its free power connectors, you should find there 5 VDC on the red wire and 12 VDC on the yellow wire, the black wires being ground. Some ATX power supplies have a manual ON-OFF switch on the back; check it. You may have a misconnected or bad power button; check the switch and the connections to the motherboard. You may have a bad power cable; try with another one.

The machine turns on but there are no beeps or video signal. You may have a defective power supply: test the voltages on one of its power connectors. The reset button may be stuck in the pressed position and keeps reseting the machine continuously; check the switch. The NVRAM of the BIOS may be in some inconsistent state; clear the NVRAM using the "CLEAR CMOS" strap. If the floppy LED gets stuck on, you may have a reversed floppy data cable; check the connections of the cable. You may have a defective card on the PCI bus: try again with each one of the cards taken off in turn. Maybe the line voltage is 115 V and the power supply is set to 230 V; check the switch on the back of the power supply. There may be a bad local or front-side bus frequency due to a misplaced strap; check the straps of the motherboard. You may have a bad video cable; check the pins in the connectors to see if they are not bent or shorted out. The machine is beeping but the speaker is misconnected; check its connections. In the worst-case scenario, you may have a bad motherboard; exchange it at your parts supplier.

The machine beeps but there is no video signal. The number of beeps may tell you what is wrong; consult the motherboard manual to see if you can find more information about this. In some cases a long beep followed by a few short beeps means that the video card is bad or absent; the AGP video card may not be correctly inserted in its slot, check the card. Endless long beebs generally mean that the RAM is bad or absent; try again with only one DIMM on the motherboard at a time. A siren-like or some other strange sound may be a CPU fan alarm, meaning that the CPU fan is bad or absent; check the fan. Fast high-pitch beeps may mean that the CPU is overheating; check whether the heat sink is too hot, but *be careful not to burn your fingers, for it can get very hot!*

There is video signal but the machine stops at some point. In this case whatever shows up in the screen up to that point should be sufficient to tell you what is wrong. Read the motherboard manual to find out what is going on. Unfortunately these manuals are often far from complete, you may have to read the documentation in a CD that should come with the motherboard, or find more information about it on the Internet. If the BIOS does not recognize the CPU type and frequency properly, it may be necessary to configure it manually, either by means of the dip-switches on the motherboard or using the BIOS setup program. It the BIOS of the SCSI card does not find all disk drives, you may have made a mistake on the SCSI addresses, or you may have a cabling or termination problem; go back and check it all.

After you solve all the problems and make the machine operate properly, it is time to close the cabinet, but before you do this there is a finishing operation to perform. You should use the plastic self-locking bracers to hold together and to hold in place all the many cable running loose inside your cabinet. Organize all the cables and wires in nice

bundles running on well-defined routes. Do as little violence as possible to the SCSI cable, have it running smoothly from the card to the disk drives. Make sure that the fans are all free to move, and that no cables are in risk of getting into them. Do not press any wires or cables against any sharp points or edges that there might be in the cabinet. Once you have the interior of your cabinet as neat as a well-organized drawer, you can close the cabinet and put it on the floor upright. Before you put back in place the front cover of the cabinet, make sure you take off the plastic covers of the front bays where you installed the disks, in order not to block the free air inflow which is needed to cool them.

The last type of operation to perform is to use the BIOS of each part of the machine to do a finer pass at configuring the hardware. Turn the machine on again and check once more the whole hardware test cycle. The first thing to do is to use the motherboard BIOS setup program to set a few things. In general you can enter this setup program by pressing and holding the `[Del]` key while you power the system up, or while you reboot the BIOS, but sometimes it is some other key, like `[F2]`, in doubt consult the motherboard manual. Once you are in the setup program, you can start by doing a "load factory defaults" in order to make sure the BIOS is in a safe and known state. If there is an option called something like "load failsafe defaults" or other equivalent thing, do that too. Then you have to navigate the menu system of the setup program in order to find and do the following things:

- Set the hardware clock to UTC, the universal time used by all computers on the Internet. It is the same as Greenwich Mean Time (GMT).
- Verify that the first floppy drive is configured correctly and that the on-board floppy device is enabled.
- Enable and configure the two serial ports, the first one should be at I/O address 3F8 and IRQ 4, the second one at address 2F8 and IRQ 3.
- Configure the boot devices in proper order: first the floppy drive, then the SCSI disks; it there is only "hard disk 0" to choose, do that.
- Set the state of the power supply after a power failure to ON, so that the machine may come back unattended from such an event.
- Configure the power button to turn the power supply off only after being held pressed for 4 or 5 seconds.
- Disable all other items relating to power management, suspend or sleep modes, wake-on-something, and so on.
- Disable all the unwanted on-board devices. These may include sound cards, modems, on-board SCSI devices, the two IDE (ATA) devices, the USB devices, the parallel port, on-board network cards, the PS/2 mouse, etc.

- Sometimes it may be necessary to disable something that is not working well, for example the main memory ECC check may be making the machine very slow during boot.

The items listed above are essential for the installation of the operating system and for the operation of the machine as a server. But, while you are at it, you might want to look also at the additional non-essential items:

- Set a password for access to the BIOS setup program. This is a must only if the machine is in a public room. Try not to forget the password.
- Set the keyboard auto-repeat parameters to whatever suits you; possibly that will be the fastest repeat rate possible.
- Set the AGP video card aperture size to the minimum possible value, this is not something that will have any use on a server.
- Turn off all memory allocations for video and system shadows. Linux will not use this in any case and you save a little bit of memory.
- Disable allocating IRQ's for USB, if it is not going to be used, and for the video card, since this is not usually needed under Linux.
- Enable and configure the parallel port if you intend to use it. It is a good idea to use the EPP mode, which does not waste a DMA channel.
- Disable the ECC checking of cache memory, this will make your machine run a bit faster.
- Check the speed of the fans connected to the motherboard, as well as the temperatures of the motherboard and of the CPU.

It is a good idea to check that the SCSI disks are well ventilated. With all fans driving the air outside the cabinet, the slots of the front bays are the main air intake and should cause a continuous flow of air around the disks. With the cabinet closed and the machine powered up, open the small front door of the front cover, if there is one, and hold a sheet of paper close to the front bay slots. It should be quite visibly sucked in by the air inflow, in fact it should be caught by the inflow and be held up by the pressure against the front cover. Check the temperature of the disks after they are running for a little while. Modern high-speed SCSI disks produce quite a bit of heat and may be quite warm even with a good air flow around them. Without it, they may become positively hot to the touch, which is not a good thing. Make sure your disks are very well ventilated and kept reasonably cool.

The next thing to do is to use the SCSI BIOS setup program to set a few more things. Usually you can enter this setup program by pressing and holding the `[Ctrl]-[A]` key

combination when the appropriate message appears on the screen, but it might be some other combination of keys, in doubt consult the SCSI card manual. The card may also be configured by means of a standalone configuration program, like the one discussed a little later for the network card. Once you are in the setup program, navigate its menu system in order to find and do the following things:

- Check whether the SCSI address of the SCSI card itself is set to 7, as it usually is.
- Activate all on-board terminators on the card, there may be several different ones if the card supports various SCSI versions.
- Check that the card is configured to send "start-motor" commands to all the disks.
- Check that the BIOS of the card is configured to load itself into memory, this is necessary in order to boot from the card.
- Disable any options that say they are something meant to bypass the limitation of the DOS operating system.
- Check that the SCSI boot disk is configured to be the disk with SCSI address 0, as it usually is.

If the BIOS of the network card comes in and keeps trying to boot from the network for a while during the hardware cycle, you may have to reconfigure it in order to disable network booting. The card may have a setup program for its BIOS, read its manual in order to find how to enter it. It may also be configured by means of a standalone configuration program. This program may be in a CD which comes with the card, or you may have to find it on the Internet. Having the program, you will have to make a configuration disk, which usually is just a DOS-bootable floppy disk with a copy of the program in it. In Section 4.1 we will discuss how to make such a floppy using the freely available FreeDOS clone of the DOS operating system. Before using the configuration disk, check whether the card has a strap for enabling and disabling changes to its NVRAM. You must have this enabled for the configuration program to work. You boot the machine from the floppy drive, into which you have inserted this disk, and once DOS comes up you execute the configuration program. The program should run, find the card and present you with a menu system allowing you to configure the card.

Booting from a floppy is a good way to test the operation of the floppy drive. Even if you do not have to configure anything with a standalone program, you might want to boot from some bootable floppy disk just in order to do this. It might even be a bootable floppy containing the Linux kernel, just as the one within the Debian installation floppy set. In fact you will do this next anyway, for you are done with assembling your server, and should now prepare to start installing the operating system.

3.3 Installing the Basic System

With the machine operating properly, we are finally ready to start dealing with the system software. The thing to do now is to perform the first installation of the operating system. This will be an installation of the Debian distribution on the server, which we will do from scratch. At first we will install in the server a very basic system that will have the role of a seed for the whole cluster. From it we will generate not only the final system of the server, but also the system of the compute nodes or remote-boot terminals. After this first installation is done from scratch, all other installations within the cluster will be done by a method called *cloning*, which means simply making a complete copy of a system and then adapting it for another similar system. Once the installations of the server and of the first node are finished, you can easily clone all the other nodes from the first one. You can also clone from the cluster server other servers that you may want to add to your cluster.

In fact, once you have a cluster up and running, you can use it to clone whole other clusters from it, without the need for any installation from scratch. Cloning, when it is possible, is usually a much faster and much easier way to install a system, saving a lot of repetitive installation and configuration work. However, this is only true if you clone each machine from another similar machine, such as a server from another server or a node from another node. This is why we must at first install only a very basic system on the server, so that we can clone from it a basic system for the first node. After that the installation and configuration of the two systems will proceed independently and they will diverge greatly in size and content. The important thing to remember from this discussion is that the initial installation on the server must be kept to a certain minimum in terms of the amount of software installed.

There is a list of common packages that one may install right at the beginning, in addition to the Debian base-system, because this set of packages is common to the compute nodes, servers, and remote-boot terminals of any typical cluster. The packages in this set were selected so that they do not require any kind of manual configuration, and because of this they will not ask you any mystifying questions during the installation. Later on we will see that when you install new software packages in the system you typically have to answer a few questions during the installation. In addition to this, sometimes a significant amount of manual configuration work is needed afterwards, as we shall see. You can find this list of packages in the resources directory tree, in a file named `common-package-list` in the subdirectory `root/`. The resources directory tree contains copies of configuration files, scripts, and other useful files, all located in the positions within the tree which correspond to their usual positions within the system. This resources area can be found in the resources CD which accompanies this book, or on the web at the address

`http://some.host.some.net/somedir/sys/`
`fma.if.usp.br:/latt/home/delyra/text/books/lfw/sys/`

The complete installation of the server will proceed in several stages. Not all disks existing on the server will be partitioned during the first phase of the installation, some will be dealt with later. On the disks partitioned in this first phase, not all partitions created will be used during the initial installation. On the first stage, when the basic system is installed in the server, only two of the four disks will be partitioned, the system disk (SCSI address 0) and the home disk (SCSI address 2). On these disks only six partitions will be actually used, the system partitions of the server itself and the home partition. On the second stage, when the first node is cloned from the server, the remaining four partitions on these disks will be used. On the third stage, when we install the automatic backup systems, the remaining two disks will be partitioned and most of their partitions will be used as backups of the partitions in the first two disks. On a fourth and final stage swap space will be doubled and used in interleave mode, and a level-0 RAID array will be installed for the `/tmp/` filesystem of the server. Another RAID array for a scratch filesystem may also be created at this stage. Later on a few additional things may be done to some filesystems, such as the installation of disk space quotas.

Just as the cluster is a network-oriented system, in which the LAN has a constant and essential role, the installation we will describe here is a network-oriented installation. This means that the installation will be done mostly through the network, and must be done within a LAN connected to the Internet in some way. The machine must be on the Internet, or at least have access to the Internet through some friendly firewall, for example by means of Linux IP masquerading running at the front-end of a private network, which functions as the gateway of that private network to the Internet. This is so because we are going to use one of the public mirrors of the Debian mirror system to get most of the software, and you must have Internet access in order to be able to reach one of these mirrors. You will need an *IP address* and a *hostname* assigned to the machine you are about to install, configuring it as part of the local network. You will also need some other information about the structure of the LAN, such as its *netmask*, its *gateway*, its *domain name* and the IP addresses of one or more *DNS servers*. All this will be discussed in more detail in a little while.

The cluster architecture described in this book was developed at the time of the "potato" distribution of Debian, a time when there was only 100 Mbps networking. Later it was upgraded to and has been used mostly in the "woody" distribution. It was during that interval of time that 1 Gbps networking appeared and the cluster servers and network equipment were upgraded to it. However, for an installation from scratch of a machine with Gbit networking, one must use the "sarge" distribution, the latest one from Debian, currently (2005) in an advanced state of development but not yet released, because it is the first one to support this new kind of network hardware. The sarge installer is currently still in phase beta-test, which means that the overall design and functionality are defined in fairly final form, but there may be bugs and the details may still change until the release of the distribution. The installation recipe below has

been designed to circumvent the current limitations of the installer and may have to be reviewed at later times.

Before we can start the installation recipe, there is a couple of preliminaries that we must go through: an explanation of disk partitions and an explanation of the basic logical structure of a LAN. We start with disk partitions, which are a form of dividing a large disk in several parts which can be accessed by the system as if each one was a separate disk device. Disk partitions are useful for separating parts of the system which have very distinct roles and properties and hence should be treated in very distinct and specific ways. The filesystem partitioning scheme we will be describing here is designed to improve the operational stability and to facilitate the administration of multi-user systems. It has been used with great success in many systems for many years now, and consists of the following essential parts.

- `/`: the `/` or root filesystem contains only the most basic parts of the operating system, which are essential for its operation in what is called the single-user or system administration mode; it contains essentially what we have been describing above as the most basic system; it contains the kernel and is the only filesystem needed for the system to boot; for several reasons, it should be on the first partition of the disk; it is small and very stable, not changing unless additional software is installed in the system; users have no write access to it and therefore it is in no danger of being filled up as a consequence of some mistake by a user.

- `/tmp/`: the `/tmp/` filesystem keeps essentially no permanent data, only temporary files, in general created automatically by programs; although users do have write access to it, it should not be used as scratch space for user files, since it is small and all such files will be deleted at each boot of the system; it is off course very volatile; since users can write to it, it will have a system of disk space quotas, to prevent a single user from filling it up and disturbing the operation of the system as a whole; events like this can have serious consequences, because many programs must be able to write to `/tmp/` in order to run.

- `/var/`: the `/var/` filesystem contains variable system data, as well as some specific types of user data; things like printer and mail spooling reside here, as well as the Debian system structure databases, system control files, system logs and various types of caches; it is very volatile, in fact the major source of constant system I/O; users have write access to some parts of it and therefore it will have disk quotas, to prevent users from filling it up and disturbing the operation of some important parts of the system such as mail, printing and system logs.

- `/usr/`: the `/usr/` filesystem contains most of the system files and programs used by the users; it is usually the largest filesystem containing system data in a server or autonomous system; it is very stable, users have no write access to it and it only changes when new software is installed or the root user does something in

it; its most volatile parts are the `/usr/src/` and `/usr/local/` subdirectories, in the first one things like the Linux kernel may be compiled and in the second one non-packaged software may be installed, but both things can be done only by the root user or super-user, or by a few users with special responsibilities and privileges.

- `/home/`: the `/home/` filesystem contains exclusively user data, in fact the home directory of each user account; it is usually the largest filesystem on a normal home server; it may be more or less volatile depending on the activities of the group of users; it will have quotas as a form of sharing the available disk space among the users; it will also have a special type of backup system.

There may also be other types of partitions and filesystems, such as a swap partition, which can be used as an extension of the physical memory of the machine, and filesystems with specific purposes, such as a `/sft/` filesystem for a local mirror, a separate `/usr/local/` partition for software not included in the distribution, a `/temp/` partition holding scratch space for users, and the `/trm/` or `/pmc/` partitions for the root of the filesystem trees of remote-boot terminals or compute nodes. The general rule is to separate filesystems by function and purpose, and to install disk quotas on those the users have write access to.

We will be partitioning the disks in a special way, to include in it both the system of the server and the system of the nodes. Any modern disk drive, either SCSI or IDE, has an internal table called a *partition table*, which holds all the information concerning the partitions existing in the disk, such as sizes, starting and ending positions, flags, labels, and so on. This partition table can hold up to four partitions, numbered from 1 to 4, which are called *primary* of *physical* partitions. Since we will need much more than four partitions, this is clearly not enough for us. Fortunately one of these four primary partitions can be configured as an *extended* partition, meaning that it can hold an extension of the partition table, containing many more partitions which are organized within this extended partition. These extra partitions are called *logical* partitions.

It is convenient to have the extended partition be the fourth and last primary partition on the disk. Within it one can create as many logical partitions as necessary, numbered from 5 on. In this way we will be able to have all the partitions that we need. Typically partition 1 will hold the root filesystem, partition 2 will be a swap partition, partition 3 will hold the `/tmp/` filesystem and partition 4 will be the extended partition containing all the other partitions. Therefore, the partitions which will appear as in use for filesystems will be numbered as 1, 3, 5, 6, 7, 8, and so on, while partitions 2 and 4 will appear to be absent, but will actually be in use with other functions.

A typical partitioning scheme for the 36 GB system disk of a standard cluster server is shown in Table 3.7. This is meant as an ample server for remote-boot terminals, with plenty of space for spooling and lots of space in `/usr/`. It can also function as a home server and as the central network server of a fairly large cluster. Another typical

Partition	Device	Size	Mount point
1 (P)	/dev/sda1	512	/
2 (P)	/dev/sda2	2048	none (swap)
3 (P)	/dev/sda3	512	/tmp/
5 (L)	/dev/sda5	6144	/var/
6 (L)	/dev/sda6	12128	/usr/
7 (L)	/dev/sda7	512	/trm/
8 (L)	/dev/sda8	512	/trm/tmp/
9 (L)	/dev/sda9	6144	/trm/var/
10 (L)	/dev/sda10	8192	/trm/usr/

Table 3.7: A partitioning scheme for a 36 GB system disk for a cluster of remote-boot terminals. The disk should have SCSI address 0, corresponding to the Linux device /dev/sda. Partition sizes are in MB. Primary partitions are labeled with (P), logical ones with (L).

partitioning scheme, this time for a smaller server meant to work as the front-end of a PMC of compute nodes, is shown in Table 3.8. In this case the server is not meant as a large spooler, but rather as a more run-of-the-mill workgroup server, with normal space in `/usr/`. This could be the minimal server described in the previous sections, and it might be used also as the server for a smaller number of remote-boot terminals. This table contains two alternatives for the last partitions on the disk, if the server is to be a PMC front-end there is space left for a scratch filesystem `/temp/`, but if it is going to serve terminals, the `/trm/usr/` filesystem has to be larger and there is no space for a `/temp/`. In this case all `/pmc/` mount points should be changed to `/trm/`, of course.

The other preliminary issue we must go through has to do with the configuration of your new server on the local network. There are several addresses, referring to the structure of the LAN you are in, that you must know and use during the installation process. First of all you must obtain the assignment of an IP address of the LAN for your machine, and you must choose a name for it, which must not already exist in the LAN and which you will use as its hostname. This hostname should be registered in the host database of your LAN, which will typically be a DNS (Domain Name Service) installed in some server within the LAN, its so-called *primary DNS server*. Both the address and the DNS record should be obtained via the network administration of your LAN. You should also ask them for the other necessary information about the network. Here is a list of everything you need with an explanation of each item. We use, for the sake of example, the number of a private network, but the logic is always the same, independently of the type of network.

- **The IP address of the machine:** this is a numerical address with a form such as `172.16.129.30`, which your machine will use in order to identify itself within

Partition	Device	Size	Mount point
1 (P)	`/dev/sda1`	256	`/`
2 (P)	`/dev/sda2`	1024	none (swap)
3 (P)	`/dev/sda3`	256	`/tmp/`
5 (L)	`/dev/sda5`	1024	`/var/`
6 (L)	`/dev/sda6`	4608	`/usr/`
7 (L)	`/dev/sda7`	512	`/pmc/`
8 (L)	`/dev/sda8`	512	`/pmc/tmp/`
9 (L)	`/dev/sda9`	3072	`/pmc/var/`
10 (L)	`/dev/sda10`	3072	`/pmc/usr/`
11 (L)	`/dev/sda11`	4037	`/temp/`
10 (L)	`/dev/sda10`	7109	`/trm/usr/`

Table 3.8: A partitioning scheme for a 18 GB system disk for a cluster of compute nodes. The disk should have SCSI address 0, corresponding to the Linux device `/dev/sda`. Partition sizes are in MB. Primary partitions are labeled with (P), logical ones with (L). Two alternatives for the filesystems at the end of the disk are shown, one of them for use with remote-boot terminals rather than compute nodes.

the network. It consists of four numbers separated by dots, each one from 0 to 255. Each number is an 8-bit (1-byte) unsigned integer.

- **The netmask of the LAN:** this is a numerical address with a form such as `255.255.255.0`, which your machine will use to decide whether a given address is inside or outside the LAN. It consists of four numbers separated by dots, each one from 0 to 255. The number 255 corresponds, in base 2, to a byte with all 1 bits. The initial part of the mask, with all ones, marks what is called the *network part* of the addresses, the final part, with all zeros, marks the *host part* of the addresses. Any address that has the network part identical to the corresponding part of the address of the machine itself is considered as being within the LAN, all others are considered to be outside.

- **The IP address of the LAN gateway:** this is the IP address of a machine within the LAN that is physically connected to other LAN's outside and which functions as a *router* for your LAN. If your machine has to send packages to an address which is outside the LAN according to the netmask, it will send the package to this gateway machine instead, because it is the machine which knows what to do with packages to hosts outside the LAN, using its routing rules. The IP address of the gateway must be inside your LAN according to its netmask, of course. It is customary to have the address 1 of the LAN assigned to the gateway, so usually you would have, in the case of the example above, `172.16.129.1`.

- **The IP addresses of one or more DNS servers:** DNS servers are systems that know how to map machine names into numerical addresses and vice-versa. You need to know the numerical IP address of at least one such system, possibly two or three. These addresses need not be within your LAN, but within a large institutional LAN you will usually find at least a primary DNS server and one or more secondary DNS servers which you can use here.

- **The hostname of your machine:** this is the name by which your machine will be identified within the LAN. It should be a string of up to 8 characters consisting mostly of lower-case letters, such as `fit`; the first character should always be a letter. It must be registered in the database of the DNS server in order that other systems can use it and have it mapped to the corresponding numerical address. This is necessary because all the actual packet traffic on the network uses only the numerical addresses. In order to identity itself outside the LAN (in case it is not a private LAN but instead a part of the Internet), the machine will use this name with the domain name of the LAN appended to it. It is a good idea to choose a name which characterizes your workgroup uniquely within your institution.

- **The domain name of the LAN:** this is the name that identifies your LAN among all the LAN's that make up the Internet. It is usually a set of two or more names separated by dots, each name being a string of up to 8 characters consisting mostly of lower-case letters, such as `free.univ.com`. Your machine can identity itself on the Internet using its hostname with this domain name appended to it with a dot, in our example this gives `fit.free.univ.com`. This is called the *fully qualified domain name* or "FQDN" of your machine.

- **The network address of the LAN:** this is a numerical IP address which can be used to identify your LAN among all LAN's that make up the Internet. This is not needed during installation since it can be derived from the IP address of your machine and the netmask of the LAN. It has the same network part as the IP address of your machine, with the host part consisting of all zeros. In our example above it would be `172.16.129.0`.

- **The broadcast address of the LAN:** this is a numerical IP address which can be used to address all the hosts within the LAN at the same time, in broadcast fashion. It is also not needed and it can also be derived from the other information. It has the same network part as the IP address of your machine, with the host part consisting of all ones. In our example above it would be `172.16.129.255`.

As explained above, the addresses in the last two items in the list are not really needed for the installation process, since they can be derived from the other ones, and are included here only for completeness. The example given is a typical one, consisting of a so-called Class-C network with some 250 hosts, but the logical structure described can

be easily and directly generalized to LAN's of any size, from a large Class-A network with over 16 million hosts right down to a miniature LAN consisting of only two hosts.

3.3.1 Installing the Server

Well, having grasped the basic concepts involved in disk partitioning and network configuration, we are ready to start the installation process. At this point your machine should be ready, with the hardware assembled and operating. Make sure all the necessary external connections to the system cabinet are made correctly: keyboard, mouse, monitor, power and network. In case you have two network cards in the machine, you have to connect only one of them, the one that will give you access to the Internet, and for the time being there is no need to do anything about the private PMC network. However, this will leave you in doubt as to which card to use for the Internet connection. For the moment, just connect the *upper* card to the external network. If this turns out not to be correct, you can simply change the connection later. Besides the information relating to the network, there are a few more pieces of information that you need to have handy before you start. The full set of information items needed is given in the list below:

A choice of language.
The country you are in.
The IP address assigned to your machine.
The netmask of your LAN.
The IP address of the gateway of your LAN.
The IP address of one or more DNS servers.
A hostname registered on the DNS for your machine.
The domain name of your LAN on the DNS.
The type and number of CPU's in your machine.
The timezone (or the city) you are in.
A non-trivial root password you will not forget.

For the installation we will use the Debian-sarge distribution with version 2.4 of the Linux kernel. As this is written the newer version 2.6 of the kernel is still a bit too new to be used on a regular basis in a production environment, although probably this will soon cease to be the case. The 2.6 kernel does have some new features which make its use compelling at least on servers and remote-boot terminals, but not so much on compute nodes. As yet there is, however, no imperious and immediate reason to use it during the installation of the cluster. If you have practiced this procedure before, you probably did it using an installation CD set and the very user-friendly approach of the normal installation procedure, in which many things are done in a more automatic fashion. However, we will follow here a somewhat lower-level approach, because it allows you to have better control over the whole sequence of operations during the installation

process. Besides, the more automatic procedure may still have quite a few bugs at this point, and for the time being it is better to use a more explicit step-by-step procedure. It will also make it easier for you to follow what is going on at each step and hence to learn about the structure of the system being installed.

Since your server probably does not and in fact should not have a CD drive, we will start the installation process with a small set of 3.5-inch, 1.44 MB floppy disks. These floppy disks are one of the most unreliable media around and you should be ready for quite a few media errors as you try to format and write them. In fact, you should have several extra ones and be prepared to throw away quite a few of them. You will need to make four installation floppy disks, so it is a good idea to get a brand-new box with 10 floppy disks of the best available quality. Although they are such an unreliable medium, they will be used only at the very beginning to start up the installation process, and very soon you will be free from them. Most of the installation process will involve downloading files over the network and will be independent of the floppy disks. The binary images you will use to make these floppy disks are called `boot.img`, `root.img`, `cd-drivers.img` and `net-drivers.img`, and can be found in any Debian mirror in the directory

`debian/dists/sarge/main/installer-i386/current/images/floppy/`

A complete list of all Debian mirrors can be fount at the web address

`http://www.debian.org/mirror/list`

In order to make a set of Debian floppy disks, you must first download these files from the mirror by HTTP or FTP, possibly using a web browser. In order to do this you will need access to some other computer already operating on your LAN. If you have the Linux operating system available in this computer, it is a simple matter to low-level format the floppy disks with the utility `superformat` and to write these images into the floppy disks using the low-level copying command `dd`, using a command line such as

```
dd if=boot.img of=/dev/fd0
```

for the case of the boot floppy, and similar ones for the other cases. In order to learn the details about how to write the images into the floppy disks you should read the Debian installation documentation. There is a complete installation manual in HTML format which you can find in several different languages within the tree of the Debian mirrors. This manual contains the explanations about writing the floppy disks. You can read this manual directly using a web browser, of course. The English version, for instance, is in the directory

`debian/dists/sarge/main/installer-i386/current/doc/manual/en/`

In fact, it might also be a good idea to read the Debian installer "howto" for sarge, before you start. It is not too long and can be found in any Debian mirror, in a common text file located at

```
debian/dists/sarge/main/installer-i386/current/doc/INSTALLATION-HOWTO
```

In any case, you must format a set of four floppy disks and write into them the four images that you downloaded, and do not forget to label the disks with the names of the corresponding images: boot, root, cd-drivers and net-drivers. In order to start the installation process, insert the boot floppy into the floppy drive and boot the machine from it. You will get in your monitor a boot prompt for booting the installation kernel, and you should now choose to enter into the *expert mode* of the installation system by typing `expert` at the boot prompt. The kernel and an initrd file will be loaded from the floppy, the kernel will boot and many lines of messages will appear on the screen as the kernel probes and activates various aspects of the hardware of the machine.

Soon the installation kernel will prompt you for the root floppy. This second floppy contains a small root filesystems with the most basic components of the installation system. Take off the boot floppy and insert the root floppy into the drive when it is asked for. The kernel will read and load into memory the contents of this second floppy and after a little while the main menu of the installation menu system comes up. It is a context-dependent menu which will change as you perform various operations. You will use some of the items in this main menu during the installation, but not all. Note that the order in which things are done is important. Let us now go through the installation sequence item by item. The list below contains the items you should use, in the correct order in which you should execute them.

- `Choose language.` Enter and choose a language. While the installer is in beta (not released or close to release) it may be safer to choose English, if this is acceptable to you.

- `Load drivers from a floppy.` You will use this item twice and read in both floppy disks with extra drivers: cd-drivers and net-drivers. You may need the kernel modules contained in these floppies in order to use the SCSI card and Gbit network card of your machine.

- `Choose country.` Select the country you are in. This will predefine or narrow down other choices you have later on, for example the choice of a Debian mirror close to you.

- `Select a keyboard layout.` Select the appropriate thing for your keyboard. If it is a standard US keyboard, choose defaults by simple hitting the `[Enter]` key a couple of times.

- `Detect hardware.` Note that *you should not use* the `Detect network hardware` alternative. When a menu with a list of modules comes up, de-select all the modules having to do with IDE and ISOFS, using the space bar and the cursor keys. Note that sometimes the `[Tab]` key is also needed to change from one part of a menu to another.

- `Configure the network.` If you have two network cards on the machine, this item will start with a menu which will let you choose one of them. Choose the one labeled as `eth0`. Do *not* try to detect the network hardware again when invited to do this. Since your new server is not likely to be registered in some DHCP server within your LAN, you should *not* try using the DHCP alternative. You should have all the necessary network information handy so just enter all the numbers and names by hand as they are prompted for. Verify the whole thing.

- `Choose a mirror of the Debian archive.` Choose the HTTP alternative, bypass the proxy menu if that is the case (it should be), choose either the testing (which is sarge before the release) or the stable (which will be sarge after the release) distribution. Choose a mirror well-connected to where you are, that is, close to you in network connectivity terms. If there are any Debian mirrors in your country, it should appear pre-selected here.

  If you have two network cards on the machine, it is possible that you will have a bit of trouble at this point. Once you select the mirror, the system will try to access it and pull from it some package database files. If the wrong network card is connected to the external network, this will fail. If this happens, just change the network connection to the other card and repeat the procedure.

- `Download installer components.` As you enter, a list of additional installer components comes up. You can simply bypass it using the `[Tab]` key, do not select anything in it.

- `Partition a hard disk.` We will use this item, but *not just now*, so **do not enter it now!**

The disk partitioning software of the installation system is not working perfectly well at this time and we will partition the disks in another way. Type `[Alt]-[F2]` to go from the first to the second virtual terminal, and then `[Enter]` to start a shell within the installation system. This is a root shell, from within which you can do many things in the installation system. You can also go to other virtual terminals with `[Alt]-[F3]` and `[Alt]-[F4]`, where you will find the installer messages and the system messages, respectively. The first thing we will do in the shell running on the second virtual terminal is to make sure that the kernel module for the ext3 type of filesystem is loaded in the kernel of the installation system. In order to do this execute first the command

```
depmod -a
```

This will establish and record all the dependencies among the available modules. Ignore any error messages that may appear when you do this. After this you can execute the command

```
modprobe ext3
```

in order to install the ext3 module in the kernel. We will now use the `cfdisk` command in this shell to partition some of the hard disks. Assuming that you have your system disk at SCSI address 0 and your home disk at SCSI address 2, you can start the `cfdisk` command for the system disk like this:

```
cfdisk /dev/scsi/host0/bus0/target0/lun0/disc
```

The installation system uses a special filesystem called `devfs` instead of just the usual directory `/dev/` with all the devices, so the first SCSI disk is accessed in the way above rather than `/dev/sda`. The home disk `/dev/sdc` can be partitioned using

```
cfdisk /dev/scsi/host0/bus0/target2/lun0/disc
```

The use of the `cfdisk` command if fairly self-explanatory. It brings up a text screen which you can navigate with the cursor keys, and there is a set of buttons in the bottom with which you can perform all the necessary operations. Vertical navigation changes the partition to work on, horizontal navigation changes the button to be executed. Everything that you do is kept in memory and does not really affect the disk until you execute the `[Write]` button. You should delete any partitions already existing on the disks, if any. If there is no partition table at all, the program will start by asking you whether it should create one, let it do so.

You should manipulate the partition table in each case until they look like the tables shown in Figures 3.1 and 3.2, for the 36 GB disks, or in Figures 3.3 and 3.4, for the 18 GB disks. Note that the root partitions should have the "bootable" flag on, and that the swap partitions are so marked. In order to mark a partition as a swap partition, you change its type code from 83 (Linux) to 82 (Linux swap) using the `[Type]` button. This is the default change, so you can just enter the `[Type]` button and hit the `[Enter]` key until the program returns to the main screen.

The numbers which appear in these figures as the sizes of the partitions in MB are what you actually get when you type in as the sizes the round numbers shown in Tables 3.7 and 3.8. The program will change the numbers slightly in order to better fit the geometry of the disk. Since your disks probably are not identical to those used here as examples, at least one of your partitions will probably be a bit different in size from the ones shown. You can safely let that be either the `/usr/` (`sda6` or `disc6`)

```
                Disk Drive: /dev/scsi/host0/bus0/target0/lun0/disc
                         Size: 36703934464 bytes, 36.7 GB
               Heads: 64   Sectors per Track: 32   Cylinders: 35003

   Name        Flags      Part Type  FS Type          [Label]        Size (MB)
------------------------------------------------------------------------------
   disc1       Boot        Primary   Linux                             511.71
   disc2                   Primary   Linux swap                       2047.87
   disc3                   Primary   Linux                             511.71
   disc5                   Logical   Linux                            6143.61
   disc6                   Logical   Linux                           12121.54
   disc7                   Logical   Linux                             511.71
   disc8                   Logical   Linux                             511.71
   disc9                   Logical   Linux                            6144.66
   disc10                  Logical   Linux                            8192.53



    [Bootable]  [ Delete ]  [  Help  ]  [Maximize]  [ Print  ]
    [  Quit  ]  [  Type  ]  [ Units  ]  [ Write  ]
```

Figure 3.1: The completed partition table for the 36 GB system disk as seen in the `cfdisk` screen. Note the bootable flag on the root partition and the swap partition marked as such.

```
                Disk Drive: /dev/scsi/host0/bus0/target2/lun0/disc
                         Size: 36703934464 bytes, 36.7 GB
               Heads: 64   Sectors per Track: 32   Cylinders: 35003

   Name        Flags      Part Type  FS Type          [Label]        Size (MB)
------------------------------------------------------------------------------
   disc1                   Primary   Linux                           36703.31









    [Bootable]  [ Delete ]  [  Help  ]  [Maximize]  [ Print  ]
    [  Quit  ]  [  Type  ]  [ Units  ]  [ Write  ]
```

Figure 3.2: The completed partition table for the 36 GB home disk as seen in the `cfdisk` screen. This disk is to have a single partition containing all the available space.

```
                Disk Drive: /dev/scsi/host0/bus0/target0/lun0/disc
                         Size: 18373206016 bytes, 18.3 GB
               Heads: 64   Sectors per Track: 32   Cylinders: 17522

   Name        Flags      Part Type  FS Type          [Label]        Size (MB)
------------------------------------------------------------------------------
   disc1       Boot        Primary   Linux                             255.86
   disc2                   Primary   Linux swap                       1024.46
   disc3                   Primary   Linux                             255.86
   disc5                   Logical   Linux                            1024.46
   disc6                   Logical   Linux                            4608.50
   disc7                   Logical   Linux                             511.71
   disc8                   Logical   Linux                             511.71
   disc9                   Logical   Linux                            3072.33
   disc10                  Logical   Linux                            3072.33
   disc11                  Logical   Linux                            4013.95


    [Bootable]  [ Delete ]  [  Help  ]  [Maximize]  [ Print  ]
    [  Quit  ]  [  Type  ]  [ Units  ]  [ Write  ]
```

Figure 3.3: The completed partition table for the 18 GB system disk as seen in the `cfdisk` screen. Note the bootable flag on the root partition and the swap partition marked as such.

```
                Disk Drive: /dev/scsi/host0/bus0/target2/lun0/disc
                         Size: 18373206016 bytes, 18.3 GB
               Heads: 64   Sectors per Track: 32   Cylinders: 17522

   Name        Flags      Part Type  FS Type          [Label]        Size (MB)
------------------------------------------------------------------------------
   disc1                   Primary   Linux                           18373.15









    [Bootable]  [ Delete ]  [  Help  ]  [Maximize]  [ Print  ]
    [  Quit  ]  [  Type  ]  [ Units  ]  [ Write  ]
```

Figure 3.4: The completed partition table for the 18 GB home disk as seen in the `cfdisk` screen. This disk is to have a single partition containing all the available space.

partition for the 36 GB disk or the `/temp/` (`sda11` or `disc11`) partition for the 18 GB disk, since in these the amount of available space is not very critical. The home disks will always have a single partition containing all the available space, of course.

When you are finished setting up the partition table for a disk, you should write the table to the disk executing the `[Write]` button. The program will ask for confirmation, which you will give typing `yes` at the appropriate prompt. You can then exit the program using the `Quit` button or just pressing the `[Q]` key. Except in the case of the `[Write]` button, the capitalized letter in each button indicates the key which will also execute the corresponding command. When you are done with all the disks, it is a good idea to enter the `cfdisk` program again with each disk, to check whether the partition tables were really written correctly to them. You can recall the previous commands you issued by using the arrow-up cursor key at the shell prompt.

Having partitioned the disks using the shell in the second virtual terminal, which you accessed by means of `[Alt]-[F2]`, you can now press `[Alt]-[F1]` to go back to the first virtual terminal, where the installer is running. It should be waiting there at exactly the point where you left it. You will now use the partitioning routine of the installer in order to finish the work on the disks, which consists of making the physical filesystems in them and mounting them in their respective roles in order to install the system software.

- `Partition a hard disk.` You can enter this item now. A table should come up showing the partition tables you have just created on the two disks. Do *not* enter the lines corresponding to whole disks. You will now enter some of the lines corresponding to partitions in order to mark them for formatting, to choose the type of filesystem to be used in them, and to choose the mount points where they should be mounted. You will do this to the following partitions:

System disk	partition 1	mount point `/`
System disk	partition 3	mount point `/tmp/`
System disk	partition 5	mount point `/var/`
System disk	partition 6	mount point `/usr/`
Home disk	partition 1	mount point `/home/`

  For each one of these partitions you should use the following menu items in order to perform the necessary tasks on them:

  - `Usage method:` choose the option `Format the partition`, which means that the installer will create a filesystem structure in the partition.
  - `File system:` choose the option `ext3 file system`, which is the standard journaling filesystem in Linux.

  - `Mount point:` choose the appropriate option according to the contents of Tables 3.7 and 3.8.
  - `Done setting up the partition`. Use this when you are done.

  In the case of the second partition of the system disk, which is already marked as a swap partition, you should enter the corresponding line in order to mark it for formating as swap space. Use the following entries:

  - `Usage method:` if it is not already selected, choose the option `Use the partition as a swap area`.
  - `Format the swap area:` just enter in order to mark the partition for formating as swap space.
  - `Done setting up the partition`. Use this when you are done.

  After you are done with all the relevant partitions and back at the screen with the partition table, you can let the installer finish the work using the following item, which is at the top of the screen:

  - `Finish partitioning and write changes to disk.` This will format all the partitions and mount them in their corresponding roles for installation purposes.

- `Install the base system.` This might take e little while, because many packages have to be pulled from the mirror over the network and installed in the disks. Just wait until it finishes. Near the end of this step you will have to choose a Linux kernel for installation in your system. In order to choose automatically the latest available version of the 2.4 kernel, use one of these alternatives, depending on the type of machine:

Athlon single:	`kernel-image-2.4-k7`
Athlon dual:	`kernel-image-2.4-k7-smp`
Pentium single:	`kernel-image-2.4-686`
Pentium dual:	`kernel-image-2.4-686-smp`

- `Install the LILO boot loader on a hard disk.` The boot loader will be installed on the first disk. You should install it on the MBR (Master Boot Record) of the disk.

- `Finish the installation.` Enter here and confirm, so that the installer will continue, reboot the system and the newly-installed system will be booted from the disk.

The machine will be rebooted. You will be presented with a prompt of the LILO boot loader. Either wait until it boots or hit `[Enter]` to boot at once from the disk. Again you will see the kernel messages as it probes the hardware. You now get to the menu system of a new installation step called "Configuring the base system". The items of this menu which you will use are as follows:

- `Configure timezone`. Say `yes` to the question about the hardware clock being set to GMT, since you did set it so before.
- `Set up users and passwords`. Say `yes` for using shadow passwords. Then set a root password and open an auxiliary user account called help, with the same password. Real user accounts will come later on.
- `Set the hostname`. Use the same hostname as before, which is the default action.
- `Configure apt`. Choose the HTTP protocol, choose the testing (sarge before release) or stable (sarge after release) distribution, say `yes` to the question about non-free software, choose the same mirror that you chose before for the installation, and say `yes` to the question about security updates.
- `Configure the Mail Transport Agent`. Say `no` to the question about splitting the configuration file. What you will do here depends on what the role of the machine will be regarding electronic mail. Have a look at the available alternatives, later you will have use for the one relating to a "smarthost". For the time being you may choose the alternative of not configuring anything at this time if you are not sure about what you should do.
- `Finish configuring the base system`. The installation system will end and you will be left with a freshly installed system running in multi-user mode.

Note that we have stopped at the installation of the base system, and have not installed any extra packages. In particular, at this stage of the overall installation process you should *not* use the items

`Select packages to install`

and

`Install selected packages`

on this menu to install packages, that can be done later, as the need arises. You should now log into the system as root, and have a good look around to check everything: `df` will show you the mounted partitions, `lsmod` the set of modules loaded in the

kernel, `ifconfig` will show the state of the network interfaces, `cat /proc/cpuinfo` will show CPU information, `cat /proc/swaps` will show the activated swap partition, `cat /proc/meminfo` will show memory information, `cat /proc/pci` will show the devices on the PCI bus, `dpkg -l | more` will show all the software packages installed in the system, and `cat /etc/apt/sources.list` will show the configuration of the `apt` utility to get software from the Debian mirror you chose during the installation process.

One useful thing that you can do is to label the disk partitions, using the `e2label` command. It is a good idea to label the partitions with the mount point they are intended for, in order to record this information for possible later use. These labels may also be used by some system programs when they report on actions they took on the partitions. Here is the command for labeling the home partition, as an example.

```
e2label /dev/sdc1 /home
```

The commands for the five partitions used so far on the system disk are analogous. After this you can enter the partition table of the disks again to see the labels displayed, this time you should use `cfdisk /dev/sda` and `cfdisk /dev/sdb`. But be careful *not* to use the `[Write]` button, so as not to do anything else to the partition table. Damaging it can destroy all your installation work so far.

You may now install the short list of common packages you can find in the resources area, in the file `root/common-package-list`. In order to copy this file into your system you can use a floppy disk, FTP or HTTP. In order to use FTP, for example, you must first install the `ftp` command, which you can accomplish executing the command

```
apt-get install ftp
```

This will install the `ftp` command into your system quickly and easily. Use it to download a copy of the file to the `/root/` directory on your machine and then, while in this directory, execute the command

```
apt-get install ‘cat common-package-list‘
```

Note the *reversed* (that is, left) single quotes around the last part of this line. After a little while and many messages on the screen, this will result in a total of a little over 300 packages installed in the system. This list of packages includes several shells, compilers, utilities and a few useful text editors, including an emacs-compatible editor called `jed` and a vi-compatible one called `vim`. You will now be able to use one of these text editors to fix configuration files in the system. Modifying text configuration files with a text editor is the norm for configuring software on Linux systems, you will be doing a lot of it.

In the resources directory you will find configuration files for the shell `tcsh`, which help to set up a nice user environment for ease and safety when using the system as the root user. If you want to use these configurations, which may be a very advisable thing if you are not a very experienced user, you must first change the shell of the root user from its default `bash` to this other shell, which is a successor of the traditional Unix `csh`. You can do this with the following command:

```
chsh -s /bin/tcsh
```

In the `root/` subdirectory of the resources area you will find the files `.cshrc` and `.login` which are used to configure this shell. The file `.cshrc` is executed by every shell which is started, the file `.login` is executed after `.cshrc`, but only by a login shell. The configurations are separated in several other files which are called by these two. The `.cshrc` file calls the files `.paths`, `.settings` and `.aliases`, which define the path for the executables in the system, set various shell variables and define various useful aliases. The `.login` file calls files `.terminal` and `.environment`, which define the characteristics of the terminal and the general operating environment of the login session. There is also a file called `setdisplay` which is used by a command of the same name defined in the configuration. Later on a similar set of configuration files will be installed for non-root users. These will be located within subdirectories of the `/usr/local/` directory of the server. In order to avoid some spurious errors when implementing the root configuration, it is advisable to create these directories now, with the commands

```
mkdir /usr/local/cbin
```

and

```
mkdir /usr/local/cetc
```

You may now download all these files to the `/root/` directory of your system, taking care to first save the original versions of the `.cshrc` and `.login` files that may already exist there, for example changing their names to `.cshrc.ORIG` and `.login.ORIG`, so that they are not overwritten and lost. Then you may use one of the text editors to look at them and to modify them as needed. If you do not have previous experience with vi-type editors, we strongly suggest that you use the `jed` editor. The files are commented to document the function of each command in them. In particular, note that they implement some important safety measures relating to the accidental deletion or overwriting of files. If you are using the `jed` editor, when you are finished modifying the files and have written new versions to disk, you may exit the editor and delete the backups of the older versions, which are created by the editor, using the command `rm -i .*~`. You can start a new shell using these new configurations simply typing

`tcsh` followed by `source .login`. This new shell and its configurations will be your default for all future logins.

One very important thing to do at this point is to check carefully the configuration file of the `apt-get` utility, which we will use constantly to get and install packages of software in the system. The file is in the directory `/etc/apt/` of the server, it has the name `sources.list`, and it should already contain configuration lines reflecting the choices you made during the installation of the basic system. Besides comment lines, it should contain the three following configuration lines, respectively for the sites of the main part, the non-US part and the security part of the Debian distribution:

```
deb http://http.us.debian.org/debian sarge main contrib non-free
deb http://non-us.debian.org/debian-non-US sarge/non-US main contrib non-free
deb http://security.debian.org sarge/updates main contrib non-free
```

The server addresses shown are just examples, you should have in each case, instead of them, the address of the mirror which is closer to you network-wise. The directory shown together with the address of the server may also change from mirror to mirror. If, except for these addresses and the corresponding directories, these lines are not exactly as shown above, edit the `sources.list` file and fix them. These are the basic configuration lines needed in order for you to be able to get and install automatically into the system binary software packages. Note the use of the keyword `sarge` in the lines, rather than `testing` or `stable`. It is better this way, because the keywords `testing` and `stable` have a variable meaning, which may change in time. While it is not released, the "sarge" distribution is pointed to by the "testing" keyword, and the "stable" keyword points to the previous version of the distribution, code-named "woody". After the release of sarge, it will cease to be the testing distribution and will become the stable distribution. Since the change from one major version to the next major version of the distribution is not a trivial undertaking, it is better if the change caused by a release does not get you by surprise.

The `sources.list` file may contain other lines, meant at enabling you to get from the servers the sources of the Debian packages, through the use of the Debian tools. These lines are not necessary if you do not intend to download sources from the servers, but they may already be there, and they should look something like this:

```
deb-src http://http.us.debian.org/debian sarge main contrib non-free
deb-src http://non-us.debian.org/debian-non-US sarge/non-US main contrib non-free
deb-src http://security.debian.org sarge/updates main contrib non-free
```

You may also include in this file other configuration lines, for example for sites distributing Debian packages for softwares which are outside the Debian distribution. A complete example of a `sources.list` file, with explanatory comments, can be found in the resources area, in the subdirectory `etc/apt/`. After you are finished fixing this file in your server, you should run the command

```
apt-get update
```

It will access all the servers listed in the configuration file and download from them files containing a database of all the packages available in each server. The program does this intelligently, so if you run the command again it will only check that the databases are up to date and not really download anything. If there are any errors because either the server of a site or the files expected within the site are not found, try other mirrors or comment out the corresponding line in the configuration file. You need, of course, to have at least one working site, for the main part of the distribution. These are the files which will be used by the APT tools to determine which packages are available and to resolve dependencies among them, every time you use them to do something. While the sarge distribution is not yet released and stable these files will change often, typically every day, so it is a good idea to run this command quite often, for example before you do something else with the `apt-get` command, every time you use it.

Another thing you might want to do is to provide your system with a nice login banner at the consoles. In the subdirectory `etc/` of the resources area you will find files called `issue` and `issue.net` with a suggestion of how to do this. The file `issue` contains some control characters which will control colors and manipulate the screen of the console, the file `issue.net` does not contain any such control characters and is used for logins over the network, for example with the command `telnet`. These files are meant for 80-column terminals, but you can easily adapt them to other circumstances. You may download these files to the `/etc/` directory of your system after you move the original files to `issue.ORIG` and `issue.net.ORIG`, then use one of the text editors to customize them for your server. This time you can delete the backups versions of the files modified with the `jed` editor using the command `rm *~`.

It is also a good idea to install in the server a better configuration for the LILO boot loader. However, before you do this you should make a backup boot floppy for the server, just in case something goes wrong. You can do that inserting a blank floppy into the floppy drive and running the command `mkboot`. After the floppy is written without error, make sure to write-protect it. After that you can use a file like this as the configuration file `lilo.conf` for the LILO boot loader:

```
# JLdL 25May04.
#
# Install the boot block in the MBR of the first SCSI disk.
boot=/dev/sda
#
# Use a nice menu-type boot loader interface.
install=menu
#
# Where the kernel loading map goes.
map=/boot/map
#
# Make a compact map, to speed up the loading of the kernel. With a
```

```
# _ compact map, one should use the old geometric mode, but this is
# _ not a problem since the root is at the beginning of the disk.
compact
geometric
#
# Stop with a prompt for 30 seconds so the operator can react.
prompt
timeout=300
#
# Require a password if any boot options are submitted manually.
restricted
password=XXXXXXXX
#
# Where the root will be mounted from.
root=/dev/sda1
#
# The VGA mode (font size) for the console.
vga=normal
#
# The first image is the default one.
image=/vmlinuz
        label=linux
        read-only
        initrd=/initrd.img
#
# The second image is the trusty old kernel.
image=/boot/vmlinuz.old
        label=oldlinux
        read-only
        optional
        initrd=/initrd.img.old
#
# The third image is a memory testing program.
image=/boot/memtest86
        label=memtest
        read-only
        optional
```

There is a copy of it in the resources area, but remember to move the original file to `lilo.conf.ORIG` before you download the new one. This file contains a password and should be protected against reading by everybody except root, so after you download it into the directory `/etc/` of your server, execute the command `hide lilo.conf`, or `chmod go-rwx lilo.conf` if you are not using our standard tcsh configurations. You may want to edit it to change a few things, for example the password, or to try a different console font that your video card may accept. Try using `vga=ask` in order to explore the possibilities. After you finish changing the file, you should run the command `lilo` to reinstall the boot loader on the disk, and then reboot your machine to see the new boot menu. If you get in trouble doing this, reboot your system from the boot

floppy you made and comment out the lines `compact` and `geometric` in the file.

The `memtest86` memory testing program which appears in this file is a handy thing to have in all your machines. This is a very useful little program, which tests the RAM of the machine very thoroughly. There is a version of it which you can run within the system, but it has limitations of access to the complete memory of the machine due to the fact that it is running under the operating system, which reserves some of the memory for its own use. This bootable version works better because it can test the existing memory more completely. This file is available in the `memtest86` package of the Debian distribution, where it is located in the directory `/boot/` and has the name `memtest86.bin`. You will also find is a copy of it in the resources area, in the subdirectory `boot/`, with the name `memtest86`. You can install this package with the command

```
apt-get install memtest86
```

As more typical examples of installation of software packages, during which you have to answer a few questions, we will finish by installing the packages lprng, libpaperg and ssh. The first is the printing queue daemon, the second will define the default paper size and the third is the secure remote shell, which encrypts all the network traffic. All you have to do is to execute as usual the `apt-get` command, first for the lprng package,

```
apt-get install lprng
```

This time a text menu will appear on the screen, asking whether or not the daemon should be started on boot. You can say yes, although the printing system in not yet really configured. Later on we will configure any existing printers in each system, by manually editing the file `/etc/printcap`. Now you may install the libpaperg package,

```
apt-get install libpaperg
```

This time the menu will ask for a choice for the default paper size. You may choose whatever is convenient for your site, in general it will be either letter or a4. Finally, you can install the ssh package,

```
apt-get install ssh
```

This time a few text menus will appear on the screen, asking a few questions that you must answer. You should say `no` to the question about using only version 2 of the SSH protocol, otherwise other systems in your LAN still using version 1 will not be able to connect to your new server. Answer with `yes` to the question about installing a certain component in "suid" mode and about running the `sshd` daemon, this last one is needed so that other systems can access your server.

If you are using the `tcsh` shell for root, you should execute after the installation of the packages the shell command

```
rehash
```

so that the shell can find all the executables which were made available within the system by the installation. This `rehash` command is an intrinsic command of the shell `tcsh` and what it does it to rescan the path of directories configured in the shell as directories containing executable files, and adding the executables it finds to a *hash table* within the shell, so it can find the files very quickly when you try to execute the commands. You should *always* run the `rehash` command after the installation of a package.

From now on, the idea is to use the `apt-get` command to install the software as the need for it arises. The command will not only install the particular package you ask for, but all other packages on which that particular package depends. In this way each one of your systems or types of system will grow with its own pace and in its own way, as the usage of the systems by the users dictates. Let us do one last thing with the `apt-get` utility before we go on. Since we are using the HTTP protocol to access the Debian mirror and to download the packages, the package utility will leave copies of the packages you installed in the directory `/var/cache/apt/archives/`. Since these copies may take up a significant amount of disk space and are no longer necessary, we may now clean up the package cache by issuing the command

```
apt-get clean
```

Later on, if you decide to build a local Debian mirror in your LAN, you will be able to export the contents of this mirror to all local systems via NFS, and in this case these copies will not be made by the Debian package management system, so it will no longer be necessary to do such cleanup operations. However, while you are using HTTP or FTP to download packages, it is important to do this once in a while, in order to keep disk usage under control in the `/var/` filesystem.

3.3.2 Cloning the First Node

We are now ready to clone the first node from the basic system which we just installed on the server. In the way of an example, let us imagine that we are cloning a compute node. In fact, the case of remote-boot terminals is not too different at this initial stage, what will be different is the subsequent development of each system. If you are going to do both, you should start with the compute nodes, because the system of a terminal is a superset of the system of a compute node so, once you have a compute node ready, you can always clone a terminal from it and then complete its system. Proceeding in this order will save you some of the adaptation work to transform the basic system of the server into the basic system of a remotely-booted machine. Although a few differences between the cases of compute nodes and remote-boot terminals will be pointed out below as they arise, if you are not yet an experienced user it may be a good idea to read through to Chapters 4 and 5, which deal respectively with compute nodes and remote-boot terminals, before you actually start cloning your first node.

What we will be doing here is a manual repetition of some of the operations which the Debian installer did for us during the installation of the server. In order to clone the first node, we must first format four partitions not yet in use on the system disk `/dev/sda`, which will be used for the root, `tmp/`, `var/` and `usr/` filesystems of the nodes. We recall from our previous tables that they are the following ones:

Partition `/dev/sda7`	mount point `/pmc/`
Partition `/dev/sda8`	mount point `/pmc/tmp/`
Partition `/dev/sda9`	mount point `/pmc/var/`
Partition `/dev/sda10`	mount point `/pmc/usr/`

Of these, three must be formatted in special ways, namely the root, `var/` and `usr/` filesystems need adjustments in their structure. We will use the command `mke2fs` in order to format these partitions, with the option `-j` so that it will in fact create `ext3` filesystems. You can format the root filesystem with the command

```
mke2fs -j -i 1024 /dev/sda7
```

The option `-i 1024` will effectively increase the number of inodes in the filesystem and consequently increase the size of the tables meant to hold names of files. This is needed because in this filesystem a single actual file, which corresponds to a single inode, usually will have many different names associated to it. The `tmp/` filesystem can be formatted in the usual way, with

```
mke2fs -j /dev/sda8
```

The `var/` filesystem also needs adjustment in the number of inodes, but a somewhat smaller one, so you can format it with the command

```
mke2fs -j -i 2048 /dev/sda9
```

The `usr/` filesystem is similar to the `var/` filesystem in terms of the need for more inodes, so you can format it with

```
mke2fs -j -i 2048 /dev/sda10
```

With this, all the necessary partitions now have appropriately formatted filesystems within them. You may now label these new filesystems, using the `e2label` command just as you did before,

```
e2label /dev/sda7 /pmc
e2label /dev/sda8 /pmc/tmp
```

and so on, for all four partitions. If you wish you can now enter once more the `cfdisk` command to look at the partition table with all the labels recorded within it.

The next step is to mount the filesystems on the corresponding mount points and transfer a copy of the basic system to them. Since the other filesystems are mounted under the `/pmc/` filesystem, you must start by mounting this one and by preparing it before you mount the others. The first thing to do it to create the mount point in the system tree of the server, since it does not yet exist. This is just the simple creation of a directory, which you can do with

```
mkdir /pmc
```

Once you have the mount point, you can mount the filesystem within the tree of the global filesystem of the server with

```
mount /dev/sda7 /pmc
```

You can now use the command `df` to check that the filesystem is in fact mounted. Note that it has only a small amount of content, which is due to the hidden journal file and to the recovery directory `lost+found/` that it already contains. Go to the pmc filesystem root with `cd /pmc` and check that indeed there is no other content there, using the `ls -a` command. You may now create the mount points for the other three filesystems with

```
mkdir tmp var usr
```

Check it out with the `ls` command and mount the remaining filesystems with

```
mount /dev/sda8 /pmc/tmp
```

and similar mount commands for the other two. Use the `df` command to check out all these mounts. You should now have the complete filesystem tree of the nodes mounted on the server.

We are almost ready to start making a complete copy of the basic system of the server into this new filesystem tree. However, in the filesystems of the nodes the data will not be copied into the root of each filesystem as it usually would, but rather into subdirectories in them. We will be creating an initial node numbered as `0000` and called `n0000`, so you should start by using the `mkdir` command to create the directories `/pmc/0000/`, `/pmc/tmp/0000/`, `/pmc/var/0000/` and `/pmc/usr/0000/`, which will function as the root, `tmp/`, `var/` and `usr/` filesystem roots of the node `n0000`. The names of these node subdirectories should have exactly four digits because that is expected by the administration scripts we will see later on.

In order to make the copy of the filesystems of the server into the new tree, it is better to first take the system down to single-user mode, in order that all filesystems

be completely quiescent. In this system administration mode all the daemons and all the automatic action on the system, such as the system logging action caused by the syslogd and klogd daemons, is stopped, and there is no I/O to any of the filesystems other than what the administrator causes manually. If you are not already in single-user mode, you can go to it from multi-user mode using the command

```
shutdown now
```

This will not halt or reboot the system, but the shell you are using will die out and you will be prompted again for the root password. All other shells that you might be running on the other virtual terminals will also exit, and all remote logins and other connections through the network will be terminated. In single-user mode you will have only a single terminal to use. Once you are back in the system in single-user mode, use the `df` command to verify that all the filesystems you mounted are still there.

We will now make exact copies of the basic system of the server into the root directories of the filesystems of node `n0000` using the command `tar`. This is quite a simple operation, but you must know something about the `tar` utility and its options in order to understand how this works. If do now know all that you need about it, you might want to have a look at the manual page of the command with `man tar`. Here is the command you should use to clone the root filesystem:

```
tar -C / -clf - .  | tar -C /pmc/0000 -xpf - &
```

Here is a short explanation of each element contained in the pipeline of commands above. First, this is in fact a pipeline of commands, in which two instances of `tar` exchange data. The option `-C` makes each `tar` change to the appropriate directory before starting its action. The options `-c` and `-x` mean that the first `tar` will be creating a stream of data out of the disk content it reads, and that the second `tar` will be reading a stream of data and writing it to disk. The option `-l` on the first `tar` means that its scanning of the disk will remain within the root filesystem of the server. The option `-p` on the second `tar` will cause it to preserve all file protection and ownership information. The `-f -` option will make the first `tar` write its stream to stdout (the standard output channel) and the second `tar` to read its stream from stdin (the standard input channel). The pipeline `|` connects the output of the first `tar` to the input of the second. The dot in the argument of the first tar will make it read recursively its current working directory, which is `/`. The character `&` at the end of the line means that the whole thing will run in the background of the shell.

Since you will be running this in single-user mode, in which you have only one terminal to access the system, and since this operation can take a while to complete, it is important to run it in the background. In this way your single available terminal will be free for you to use for other things, like monitoring the progress of the operation, which you can do using the `df` command and watching the occupation of the `/pmc/`

filesystem. It is also important to make the pipeline run quietly, *not* using the "verbose" option `-v` on the `tar` commands, otherwise your terminal will be swamped with irrelevant output that will render it useless. After the command completes without error, you can go to the `/pmc/0000/` directory to check its contents. Since this directory is not the root of a physical filesystem, you can delete the `lost+found/` directory that the `tar` command created in it, with

```
rmdir lost+found
```

In the case of the `/pmc/tmp/` filesystem there is no content to copy, of course, so all that is needed is to fix the permissions of the directory corresponding to the node `n0000` with

```
chmod 1777 /pmc/tmp/0000
```

This will make the directory readable and writable by anyone, and will activate the "sticky bit" which will prevent one user from destroying files belonging to other users. You can check the result of this command executing `ls -l /pmc/tmp/`. For the `var/` filesystem you can now run

```
tar -C /var -clf - .  | tar -C /pmc/var/0000 -xpf - &
```

When this pipeline completes, you can delete the directory `lost+found/` within the directory `/pmc/var/0000/`, just as you did before in the case of the root directory of the node. We are now left only with the `usr/` filesystem to do so we run the pipeline

```
tar -C /usr -clf - .  | tar -C /pmc/usr/0000 -xpf - &
```

When this one completes, you can delete the directory `lost+found/` within the directory `/pmc/usr/0000/`, as you did in the other cases.

The cloning of the basic system for the node n0000 is now complete. While some of the details of its structure are not yet appropriate for use on a remotely booted system, the basic structure is in place. There is now need for some work within this freshly cloned system to adapt it for its intended use. Some of this work can be done immediately, but other parts will have to wait until some necessary utilities and network services are installed, configured and running on the server. You may now get out of single-user mode and back into multi-user mode. Just press the `[Ctrl]-[D]` key combination to terminate the shell you are in now, this will make the system change back to multi-user mode. You will have again six virtual terminals available on the console, and may log back in as root in one of them.

You may now go to the directory `/pmc/0000/etc/` and use your favorite editor to change a few files there. For example, the file `hostname` should have its contents changed from `fit` to `n0000`. The file `hosts` should be changed to look like this:

```
# JLdL 22May04.
#
127.0.0.1       localhost
#
192.168.0.1     pmcs00.free.univ.com    pmcs00
#
192.168.0.10    n0000.free.univ.com     n0000
192.168.0.11    n0001.free.univ.com     n0001
192.168.0.12    n0002.free.univ.com     n0002
192.168.0.13    n0003.free.univ.com     n0003
#...            ...                     ...
+
```

The network `192.168.0.0` with netmask `255.255.255.0` will be used for the private network of the PMC. The host `pmcs00` which appears in this file in the name of the server within the private network. When a machine has two or more network interfaces, it will always have several names, one corresponding to each interface. If our server here is the front-end of a PMC, it will have the name "fit" in the external network, and `pmcs00` (PMC server `00`) on the private network. You will find copies of these files in the resources area; in the case of the hosts file it is a version ready for a cluster of 24 compute nodes. You should also create entries for `pmcs00` and for the PMC nodes you have in the hosts file of the server itself. You can simply copy the lines from one hosts file to the other. Another file of the node that you must adjust is the `interfaces` file in the `network` subdirectory. You should change it to look like this:

```
# JLdL 22May04.
#
# /etc/network/interfaces -- configuration file for ifup(8), ifdown(8)
#
# The loopback interface
auto lo
iface lo inet loopback
#
# The first network card (network, broadcast and gateway are optional)
auto eth0
iface eth0 inet static
        address 192.168.0.10
        netmask 255.255.255.0
        network 192.168.0.0
        broadcast 192.168.0.255
        gateway 192.168.0.1
```

Note that we started numbering the nodes with 10 on the last part of the IP, within a private network of 256 addresses, leaving the first ten addresses for the server or servers and other possible uses, such as the Ethernet switches in the private network. Another important file to fix is the `fstab` file of the node, which should look like this:

```
# JLdL 25Sep04.
#
# <file system>         <mount point>   <type>  <options>         <dump>  <pass>
pmcs00:/pmc/0000        /               nfs     rw,nolock         0       0
#
proc                    /proc           proc    defaults          0       0
#
pmcs00:/pmc/tmp/0000    /tmp            nfs     rw,nolock         0       0
pmcs00:/pmc/var/0000    /var            nfs     rw,nolock         0       0
pmcs00:/pmc/usr/0000    /usr            nfs     rw,nolock         0       0
#
tmpfs                   /tmpfs          tmpfs   defaults,size=1M  0       0
```

Note that all the system mounts are NFS mounts in this file, from the server `pmcs00` in the private network. The `tmpfs` mount relates to a type of RAM-based filesystem which is supported by the kernel of the node and which is made necessary due to some difficulties with the NFS filesystem which are present in some MPI libraries, requiring that some small per-node configuration files reside in a non-NFS filesystem. You should also create the mount point `tmpfs/` in the root of the node. Another file that you must adjust is the `/etc/inittab` file of the node. This file defines the run levels in which the system can operate, as well as some services and properties which should be active in each one of them. The reason that we adjust this file is that the nodes have no monitors or video cards, and so there cannot be any active virtual terminals. The `inittab` file determines by default that there should be `getty` daemons running on the first six virtual terminals, which are used to prompt the user and acquire keyboard input so as to permit user logins in these terminals. The six lines which configure this in this file are the following:

```
1:2345:respawn:/sbin/getty 38400 tty1
2:23:respawn:/sbin/getty 38400 tty2
3:23:respawn:/sbin/getty 38400 tty3
4:23:respawn:/sbin/getty 38400 tty4
5:23:respawn:/sbin/getty 38400 tty5
6:23:respawn:/sbin/getty 38400 tty6
```

You should just comment them out, inserting a `#` symbol as the first character of each one of them. Another change you should do is to add a couple of lines in order to have `getty` processes running on the two serial ports of the node. This will allow you to use the serial ports in order to access the nodes, by means of the serial console which will be described later. The two lines that should be added to the file in order to do this are these:

```
T0:23:respawn:/sbin/getty -L ttyS0 -f /etc/issue 38400 vt100
T1:23:respawn:/sbin/getty -L ttyS1 -f /etc/issue 38400 vt100
```

A complete copy of this file, with the adjustments made, can be found in the resources area, inside the `pmc/0000/etc/` subdirectory. Note, however, that these modifications in `inittab` should *not* be done in the case of remote-boot terminals, which do have video cards and monitors. Also, you should keep the original file with a name such as `inittab.ORIG`, because you may need to use it temporarily even in the case of compute nodes, if and when you decide to perform network booting tests on the workbench, as will be described in Section 3.4. Another thing you should do is to empty the `mtab` file that the node inherited from the server. This is a file that records the mounts which are effective in the system at any given time and, therefore, its current contents are not appropriate for the node. You can do this with the command

```
cat /dev/null >!  mtab
```

executed within the `/pmc/0000/etc/` directory. It is also a good idea to prevent the automatic generation of the `/etc/motd` (message of the day) file at each boot, by editing the file `etc/default/rcS` of the node. The entry `EDITMOTD=yes` that you will find there should be changed to `EDITMOTD=no`. You may then empty the `motd` file with a command such as the one used above for the `mtab` file. Other files you may want to fix are the `issue` and `issue.net` files of the node. All the files shown or mentioned here can be found in their usual location in the resources area. Other things you can do is to delete the `pmc/` directory within the root of the node with the command `rmdir /pmc/0000/pmc/`, since it clearly has no role to play there. Finally, it is a good idea to go to the directory `/pmc/var/0000/run/` and eliminate from there all files recording PID's (Process Identification Numbers) which might be left there, using the command

```
find . -name \*.pid -exec rm -f \{\} \;
```

Make sure you only run this after going into the directory. This command will find within the given directory and its subdirectories, and subsequently delete, any files with names ending with `.pid`. These are files containing the PID's of system processes such as daemons, and may still contain data pertinent only to the server.

These are all the adjustments that we can do on the first node for the time being. This work is not yet finished, but in order to continue it we must first install some essential network services on the server. Some things like installing new packages or taking packages off the node can only be done once these services are up and running. It is very important to have the first node perfectly well adjusted before you clone others from it, in order not to increase the work later. Therefore, we will adjust and test the first node thoroughly before we go on to build the complete PMC cluster. Some of the work still needed can be done using only the server, but eventually the tests will include remotely booting an actual node assembled and configured in the private network,

Let us end by noting that, although the route we took here is the one that minimizes the overall installation work, it is also possible to clone the systems from each other in almost any given circumstances. For example, one can even clone a compute node from a completed server, but there will be many packages to take off and many more adjustments to make, so there will be lots of work involved. As a more feasible example, we may point out that a terminal can be cloned from a server with much less work, since their sets of packages are not so different. In fact, although a few packages would have to come off, in this case you would in fact have to install a fair number of new packages, because the terminals may have and usually will have a larger number of packages installed than their servers.

3.4 Setting up Network Booting

Our next objective is to set up in the server the essential services that will allow us to test the software structure of the single node which we just created by the process of cloning. At first we will do this using only the server itself, without having to actually assemble and operate the hardware of the node. This can be done by means of a command called `chroot`, which starts a shell within which the root of the system is changed to a given directory. We may use this command changing the root to the actual root of the node within the filesystem structure of the server. This allows us to operate and modify the software structure of the node in a way which is completely independent of the current state of its hardware, which in fact might not even exist yet. So long as one is careful about a few things, it is also possible to do this while the node is up and running in full multi-user operation. For the time being we will use this to test the system of the first node. The actual operation of the hardware of the node will be delayed for a little while and will be discussed in the next chapter.

In this phase we will also install on the server the software needed for the future operation of the nodes. Since there are two network boot programs which we may use, we will describe the software needed for each one of them. The two programs we will discuss are the PXELinux and the Etherboot network boot loaders. They work according to very different operating strategies. While the Etherboot program is supposed to be written directly into the EEPROM chip of the network card, from where it is loaded into the node by the BIOS of the motherboard, the PXELinux program is loaded into the node by TFTP, by a standard PXE program residing in the EEPROM of the card. In this case it is this standard PXE program which is started by the BIOS of the motherboard. Many modern network cards come with such a PXE program pre-installed, and the use of the PXE protocol for network booting seems to be establishing itself as a de-facto standard. Due to this, as well as to the fact that it avoids the need for the low-level technical operation of writing a program into the EEPROM chip of the card, the implementation of the PXELinux strategy is much simpler. In addition to this the PXELinux alternative is quite a bit more flexible and easy to configure than the Etherboot alternative. We will therefore use the PXELinux alternative as our default here, keeping the Etherboot alternative for those cases in which for one reason or another the PXELinux strategy cannot be implemented.

There is another objective which should be accomplished in this phase of the process, namely the completion of the software of the node, which should be finished before we start cloning other nodes from the first one. What this software completion should include will depend on the particular needs of the group, of course. Therefore, no attempt can and will be done here at describing this completion in detail. While we will still go through the installation a few more packages of useful tools and utilities, both in the first node and in the server, there is no claim that this will make the software of the node complete for any particular cluster. What we do claim is that, once this second set of packages is installed in the node, most of its software will probably be in place

and any other things that may turn out to be necessary can be installed without too much trouble later, after all the nodes are cloned. So, if you know that there are other software packages that will be necessary, it is a good idea to install them in the first node in this phase of the process. In case you choose not to do this now, you will be able to do it later, using the clustering tools which will be described in further chapters. We will also install this second set of packages in the server itself, although in this case we are in no hurry to complete the software and we may just keep installing each thing as the need for it arises.

3.4.1 Testing the Node Filesystems

Let us start preparing the server for the operation of the nodes, then. As a first step we will install only the services required to enable us to test the structure and functionality of the filesystems of the nodes. If the server has two network cards, the first thing to do is to set up the second network interface. This interface will have the address associated to the name pmcs00 of the server in the private network. This name has already been recorded in the `hosts` file of the node, and now it is time to include it in the file `/etc/hosts` of the server. This file should be modified to contain at least the three lines below:

```
# JLdL 23Jul04.
#
127.0.0.1       localhost
172.16.129.30   fit.free.univ.com       fit
192.168.0.1     pmcs00.free.univ.com    pmcs00
```

An example of the complete version of this file can be found in the resources area, in the subdirectory `etc/`. With this done all that remains to be done is to configure the interface itself, which is to be done in the file `interfaces` of the server. The first network card `eth0` should already be configured in it, what we need to do is to add a second block with the configuration for `eth1`, which should look like this:

```
#
# The second network card.
auto eth1
iface eth1 inet static
        address 192.168.0.1
        netmask 255.255.255.0
        network 192.168.0.0
        broadcast 192.168.0.255
```

Note that there is no gateway entry in this block. An example of the complete file can be found in the resources area. With this in place one can raise the interface with the command `ifup eth1`. The command `ifconfig` can be used to show all the configured interfaces. Also, the command `route` will show the kernel routing rules

associates to each interface. You may also watch the interface at work issuing the command `ping pmcs00`.

Next we must activate the exportation of the filesystems through the NFS protocol, so we must install the package `nfs-kernel-server`. This is a support package for the NFS server which is internal to the kernel and which should already be compiled as a module called `nfsd`. The role of the NFS protocol in our cluster architecture is a very central one. It allows the diskless client to mount and have available to them filesystems which are actually located at the server, in its disks. The NFS server sends and receives through the network blocks of data resulting from the I/O activity at the node, and in this way acts as a kind of remote disk device for the node, which will function exactly as it would if it had a local disk device. The NFS protocol allows a single filesystem at the server to be exported to and mounted simultaneously by many hosts, and has mechanisms for resolving the possible file access conflicts that may arise. It also allows subdirectories of the filesystem to be mounted by the nodes, rather than just the whole filesystem. The NFS protocol will be used for both the system disks of the nodes and for making available at all the machines the centralized home disk with the user data.

The installation of the `nfs-kernel-server` package on the server is easily accomplished with the command

```
apt-get install nfs-kernel-server
```

This will enable and start the server, and you can verify that the `nfsd` module was automatically loaded into the kernel using the `lsmod` command. However, this is not all, we must still configure the NFS exports, that is, the exportation of the appropriate filesystems to the appropriate hosts. This is really a process of setting up the appropriate set of authorizations for other hosts so that the kernel NFS server will answer appropriately to remote mount requests sent in by them. This is configured in the file `/etc/exports` of the server, which at this time should be changed to look like this:

```
# JLdL 30Sep04.
#
# PMC system to the server itself.
/pmc        localhost(rw,async,no_root_squash) pmcs00(rw,async,no_root_squash)
/pmc/tmp    localhost(rw,async,no_root_squash) pmcs00(rw,async,no_root_squash)
/pmc/var    localhost(rw,async,no_root_squash) pmcs00(rw,async,no_root_squash)
/pmc/usr    localhost(rw,async,no_root_squash) pmcs00(rw,async,no_root_squash)
```

As usual, a copy of this file is available in the resources area. The options `rw` and `no_root_squash` shown in parenthesis tell the NFS server to allow both read and write operations to the exported directory tree, as well as to allow access to it with root privileges on the part of the client. The option `async` is meant to help speed up the NFS input/output operations, for details look up the exports manual page. For the time being we are exporting the filesystems of the nodes only to the server itself, for the purposes of the current tests, which will use loopback NFS mounts. Later we will add

to this file export entries for the first node, and later on to all the nodes, in this case using NIS netgroups. As you see, this exports file will have to be changed several times later, in fact it is one of the files which may require maintenance along the operational life of the server. After you are done changing this file, you should make the NFS server reload its data tables with the command

```
/etc/init.d/nfs-kernel-server reload
```

With this done we are ready to test the exports by mounting the exported filesystems back on a mount point in the server itself. The standard mount point for this kind of thing is the directory `/mnt/`, so you may now try the command

```
mount pmcs00:/pmc/0000 /mnt
```

in order to mount the root of the first node on `/mnt/`. The command `df` following this should show the NFS filesystem mounted on the mount point. For a few more details about the mount one can use the command `mount` with no arguments. The command `cat /proc/mounts` will give all the details about the mount. You may now mount the remaining filesystems of the first node on top of this one, using the commands

```
mount pmcs00:/pmc/tmp/0000 /mnt/tmp
mount pmcs00:/pmc/var/0000 /mnt/var
mount pmcs00:/pmc/usr/0000 /mnt/usr
```

for the `/tmp/`, `/var/` and `/usr/` filesystems of the first node. After this you should check out the whole set of mounts using `df` and `mount`. If all is well you may unmount the whole set in the reversed order, starting with the command

```
umount /mnt/usr
```

and so on for the other three filesystems. Note how you must unmount the `/usr/`, `/var/` and `/tmp/` filesystems before you unmount the root.

After all the test mounts are undone, we are ready to test the structure of the first node. If fact, this starts very simply, you just have to run the command

```
chroot /pmc/0000
```

which will start a new shell in such a way that this new shell thinks that the root of the system it is running on is `/pmc/0000/` instead of the root of the server. Once this new shell starts it will be running with an apparent default directory `/`, which in fact corresponds to the directory `/pmc/0000/`, so that only the files within this directory are accessible to this shell. Have a look around to verify this, the directory `/etc/` should

contain the files that you already modified for the node before, check it out. Note that the directories `/tmp/`, `/var/` and `/usr/` are empty, since the corresponding filesystems of the node are not yet mounted over the root. The same is true also for the `/proc/` directory, since the corresponding virtual filesystem not only is not mounted but does not in fact exist in the node, since there is no kernel running in the node itself. Note that the prompt still indicates that the shell is running on the server, the only thing that changed is the filesystem tree which is visible to this particular shell.

Since you emptied the file `/etc/mtab` of the node before, the commands `df` and `mount` will show nothing, although the content of the root is available to the shell. In order to correct this you can run the command

```
mount -f /
```

This is a fake mount, which will not mount anything but will add to the `mtab` file the appropriate entry for the root of the node. After this the commands `df` and `mount` will show the expected results. The root will appear to be mounted via NFS, as recorded in the `fstab` file of the node, although there is no actual NFS mount involved in this case. You can verify this exiting the chroot shell and checking out the actual mounts on the server. You can get back to the old shell on the server by exiting the current shell as usual, executing `exit` or typing a `[Ctrl]-[D]`. After you are back on the server you can check all the mounts which in fact exist in the kernel by means of the command

```
cat /proc/mounts
```

You should see that no NFS mounts appear in this table. Note that even if there were mounts within the node, these would not be shown by either the `df` or the `mount` commands on the server, these will only show the mounts of the server itself, as listed in the file `/etc/mtab` of the server. You may also have a look into the file `mtab` of the node to see that it is in fact not empty anymore.

Let us now go back into the node filesystem using again the `chroot` command. This time `df` and `mount` will show the root of the node "mounted" in place. In order to complete the filesystem structure of the node, we will now mount the other filesystems via loopback NFS mounts. This means that what in fact will be happening is that the server will be exporting certain directories to itself through the network, and will also be mounting these exported directories on other directories in its filesystem structure. From within the chroot shell, it will appear that the node has had its filesystems mounted just as it would be the case if it was really up and running. You may now mount the remaining filesystems of the node executing inside the chroot shell the commands

```
mount /tmp
mount /var
mount /usr
```

Once this is done, use `df` or `mount` to verify the mounts. It all looks exactly like it would look in an actual running node. If you are using the `tcsh` shell for root, you should execute the command

```
rehash
```

after the mounts, so that the shell can find all the executables which were made available by these mounts. We may now use all the usual Debian utilities for dealing with the packages of the node. As an example, we may take off the node the package `lilo`, since it does not make any sense to have this disk-oriented boot loader in it. You can do this running in the chroot shell the command

```
dpkg --purge lilo
```

You can then use the command `dpkg -l` to list the remaining packages in the node. A little searching of the resulting list with `grep` or the command `dpkg -s lilo` will convince you that the `lilo` package is not there anymore. You should now exit the chroot shell with `exit` or `[Ctrl]-[D]` and repeat these commands on the server to verify that the `lilo` package is still present there. While you are back at the server, use `cat /proc/mounts` again, to verify that the loopback NFS mounts for the `/tmp/`, `/var/` and `/usr/` filesystems of the node are there this time. Note that it is possible to unmount these filesystems from the server as well as from within the node. In the future you will in fact do this under some circumstances, but do *not* do it now. Since you mounted the filesystems from within the node and the mounts were recorded in the `mtab` file of the nodes, it is better to unmount them also from within the node when the time comes.

Before that, there is another thing that we can do, we can improve the software structure of the node, installing in it a second set of rather basic packages. In order to do this, before you go back into the node, you should get a copy of the file `common-package-list-2` which you will find in the `root/` subdirectory of the resources area, and put it in the `root/` directory of the node, that is, you should in fact put it in the directory `/pmc/0000/root/` of the server. With this done, you can get back into the node with the `chroot` command. Once in it, use the `df` command to verify that the mounted filesystems are still there. Since the chroot shell starts in the directory you changed the root to, that is, the root of the node, execute `cd` without arguments to go to the home of the root user, the `/root/` directory of the node. Once there, you can make sure that the Debian package databases are updated, with the command

```
apt-get update
```

This will access the same Debian mirror you chose before from within the filesystem structure of the node and work just as it would on the server. Let us recall here that while the sarge distribution is not yet released it is a good idea to run this command quite often, typically right before you do something else with the `apt-get` command. You may now install the second set of basic packages with the command

```
apt-get install ‘cat common-package-list-2‘
```

Differently from the first set of packages, these packages will ask you quite a few questions, but it is safe to answer all of them by simply pressing `[Enter]`, thus accepting the default answers. After a while you will have a node with approximately 370 packages installed in it. You should now run the `apt-get clean` command to clean the package cache. You can use the `dpkg` command and a reverse-search `grep` such as in

```
dpkg -l | grep -v ^ii
```

to check that all packages are correctly installed and configured. Any broken packages will appear in the output of this command after the headers. This may not always be an error condition, sometimes they are just packages which were de-installed by the Debian tools because you installed an alternative package with the same function. For example, the `telnet-ssl` and `telnetd-ssl` packages contained in our list will cause the `telnet` and `telnetd` packages which they replace to be de-installed in this way, if you have them installed. In this case you can get rid of these spurious entries by finishing the removal of the packages, with the command

```
dpkg --purge telnet telnetd
```

Besides checking for broken packages, you can count the number of installed packages with a similar command, using a direct search,

```
dpkg -l | grep ^ii | wc -l
```

Before you leave the node by exiting the chroot shell, do not forget to unmount its filesystems, you can use for that the simple command

```
umount -a
```

Once back in the server after exiting the shell, check that in fact there are no remaining loopback mounts in the `/proc/mounts` table.

3.4.2 Preparing the Server for Network Booting

We must now continue with the installation of the necessary services on the server. While the NFS server was the only thing needed to allow you to access the filesystems of the node in the way described above, other services will be needed in order for an actual node to boot and mount its filesystems through the network. The first thing to do is to add an entry for the first node in the file `/etc/hosts` of the server, which should now contain at least the lines

```
# JLdL 23Aug04.
#
127.0.0.1       localhost
172.16.129.30   fit.free.univ.com       fit
192.168.0.1     pmcs00.free.univ.com    pmcs00
#
192.168.0.10    n0000.free.univ.com     n0000
```

As usual there is a copy of this file in the resources area, in the subdirectory `etc/`, with the name `hosts.2`, in order to distinguish it from our first example for this file. Naturally, in order to use it you must change its name to `hosts`. In the NFS subsystem there is only an adjustment left to make, since we still have to export the filesystems to the first node. This requires a modification of the `/etc/exports` file of the server, which should now look like this:

```
# JLdL 30Sep04.
#
# PMC system to the server itself.
/pmc        localhost(rw,async,no_root_squash) pmcs00(rw,async,no_root_squash)
/pmc/tmp    localhost(rw,async,no_root_squash) pmcs00(rw,async,no_root_squash)
/pmc/var    localhost(rw,async,no_root_squash) pmcs00(rw,async,no_root_squash)
/pmc/usr    localhost(rw,async,no_root_squash) pmcs00(rw,async,no_root_squash)
#
# PMC system to the virtual compute node.
/pmc        n0000(rw,async,no_root_squash)
/pmc/tmp    n0000(rw,async,no_root_squash)
/pmc/var    n0000(rw,async,no_root_squash)
/pmc/usr    n0000(rw,async,no_root_squash)
```

There is a copy of this file as well, in the subdirectory `etc/` of the resources area, with the name `exports.2`. Remember to change its name to `exports` if you want to use it, and to reload the exports table of the NFS server, after every time you make any modifications to the `exports` file, with the command

```
/etc/init.d/nfs-kernel-server reload
```

Going back to the remaining services needed, let us start with TFTP, the trivial file transfer protocol. This service will be used to transfer the kernel of the node from the

server to the node, so it can boot it. The action of getting the kernel by means of this service will be taken by the network boot program, either the PXELinux program loaded on the node by means of the PXE protocol, or the Etherboot program which later on we will learn to write directly into the boot EEPROM chip of the network card of the node. Because the PXELinux program needs a TFTP server with support for an extended set of commands, we will use the `tftpd-hpa` package instead of the usual `tftpd` package. You may install on the server the `tftpd-hpa` package using the `apt-get` command as usual,

```
apt-get install tftpd-hpa
```

Like almost all packages containing network daemons, this one will make some automatic changes to the file `/etc/inetd.conf` of the server. This file contains lines which configure the functionality of each network daemon which can be started on-demand by the Internet super-server `inetd`. Since it changes according to the packages which are included in or taken away from the system, we do not have a complete copy of it in the resources area. What we do have there is a file containing a set of useful example lines for this file, kept in `etc/inetd.conf.LINES`. In the case of the TFTP daemon there are a couple of adjustments that we need to do in the corresponding line of the `etc/inetd.conf` file of the server. First, you should change the fourth entry of that line from `wait` to `wait.300`, so that the beginning of the line looks like this:

```
tftp dgram udp wait.300 root...
```

This will change, from the default of 40 to 300, the number of TFTP requests that the `inetd` daemon will agree to receive within a period of one minute. If it receives more than this number, the `inetd` daemon will declare the service to be looping and will disable it for a certain amount of time. The historical limit of 40 is excessively restrictive for modern machines and networks, and a PMC server with many nodes will easily surpass it at the time that all the nodes are trying to boot. The second change to be done is to add `-s /tftpboot` at the end of the line, which should look like this:

```
.../usr/sbin/tcpd /usr/sbin/in.tftpd -s /tftpboot
```

This is an option to the TFTP daemon which makes it change its root to the given directory, and is convenient for security reasons. The name `/tftpboot/` is traditional for the directory holding the files that may be obtained through the TFTP protocol. But we do not really have to have a directory there, it may be a symbolic link (also known as a soft link or as a symlink) pointing somewhere else. We will adopt a directory called `tftpboot` located inside the `/pmc/` filesystem, so you should now create that directory and make an appropriate link pointing to it in the root of the server, with the command

```
ln -s pmc/tftpboot /tftpboot
```

If you go to the root of the system with `cd /` and use the `ls -l` command, the link should show up something like this:

```
lrwxrwxrwx    1 root     root          12 Dec 29 02:10 tftpboot -> pmc/tftpboot
```

After you are done with all the changes to the `/etc/inetd.conf` file, you should make the inetd daemon re-read its configuration tables using the command

```
/etc/init.d/inetd reload
```

In order to test the operation of the TFTP server, you must put some file in the `/pmc/tftpboot/` directory. Any file will do; for example, make a copy of the kernel of the server itself, which you will find in the directory `/boot/`, in that directory with the name `vmlinuz`. This kernel will not actually work for a node, but it is good enough for a simple test of the server. Later on we will have to compile and to install in this directory a special kernel for the nodes. You can do a first test going to the `/tmp/` directory of the server and using the `tftp` command in this way:

```
fig:/tmp> tftp pmcs00
tftp> get vmlinuz
Received 645659 bytes in 0.3 seconds
tftp> quit
```

This will make a loopback connection to the server itself and get the `vmlinuz` file. After you exit the `tftp` command you can check that indeed a copy of the file now exists in the `/tmp/` directory. It is of course better to repeat this test on some other machine in your LAN if you have access to one, but this simple test suffices to show that the TFTP server is operating properly.

Another service you must have on the server is DHCP, the dynamic host configuration protocol. This one will also be used by the boot program, in this case to obtain from the server the network configuration parameters of the node. Since a diskless client has no permanent data stored within it other than the hardware address of its network card, in the first phase of the network boot process it must obtain from the network, using this single piece of information as its identification, all the other necessary information such as its own hostname, IP address and remaining network configuration parameters, as well as the name and address of the servers it should use for getting its kernel by means of TFTP and for mounting its root by means of NFS. After it mounts the root it has access to all the information stored within it, of course, so the DHCP protocol is only needed for this initial phase of the boot process. After the root is mounted the node will operate exactly like any other system, regardless of the fact that it has no local disks.

The DHCP protocol works by broadcast at the Ethernet level. The boot program will send to the Ethernet broadcast address `FF:FF:FF:FF:FF:FF` a message containing the Ethernet address of its own network card and a request for a reply from any server that has that address recorded in its DHCP configuration file, and therefore knows how to respond. In this response the server will inform the diskless client of its IP address, hostname, TFTP server and all other necessary network data. The boot program will then get the kernel from the TFTP server (which may or may not be the same as the DHCP server), and will load it in memory with appropriate boot parameters. At this point the boot program has done its job, it dies out and the kernel boots and takes over the machine. It detects all the hardware and mounts its root from the NFS server, which may or may not be a third server, different from the other two. The kernel may get the information about its NFS server in several ways. For example, it may have been loaded by the boot program with boot parameters containing that information, or it may get the information from the DHCP server, doing itself a broadcast call. The boot program may have obtained the boot parameter data via DHCP with the rest of the node information, via TFTP by means of a configuration file, or the parameters may be encoded in a boot block attached to the kernel file itself.

As you can see, the details of the complete network boot process can happen in many different ways. We will choose a particular strategy to describe and use here, possibly not the most up-to-date way to do things, but one that is simple, reliable and easy to maintain. We will start by giving the configurations for the DHCP service and, for the case of the Etherboot alternative, describing the tools for preparing a NBI kernel, a procedure which is not necessary for the PXELinux alternative. We will assume that all three services, DHCP, TFTP and NFS, are provided by a single server, our cluster server. In order to install the DHCP server, it suffices to run

```
apt-get install dhcp
```

The server will try to start but will fail due to the lack of a complete configuration file. There are two files that you must change in order to configure the DHCP server. In the file `/etc/default/dhcp` you will determine which network interfaces the server will listen to for the DHCP broadcast requests. In the case of the front-end server of a PMC, which has two network interfaces, you should configure the server to listen only to the interface to the private network containing the nodes it serves, that is, to the `eth1` interface. The file should look like this:

```
# JLdL 07Jun04.
#
# Defaults for dhcp initscript
# sourced by /etc/init.d/dhcp
#
# On what interfaces should the DHCP server (dhcpd) serve DHCP requests?
#       Separate multiple interfaces with spaces, e.g. "eth0 eth1".
INTERFACES="eth1"
```

The other file that must be modified is the `/etc/dhcpd.conf` configuration file of the `dhcpd` daemon. It must contain a subnet declaration for the interface `eth1`, informing the server about how to identify the broadcast requests, as well as several options and records identifying each one of the existing nodes. The information contained in this file, added to other information which may be contained in the kernel file (in the Etherboot alternative) or in a configuration file (in the PXELinux alternative) to be downloaded to the node, will not only allow each node to configure itself in the network, but will also give the node instructions as to how to obtain its kernel file and where to find its root filesystem. The `dhcpd.conf` file should look like this in the case of the PXELinux alternative:

```
# -*-shell-script-*-
# JLdL 09Jun04.
#
# dhcpd.conf for the PXELinux alternative.
#
# The subnet for eth1.
subnet 192.168.0.0 netmask 255.255.255.0 {
    option routers 192.168.0.1;
    filename "pxelinux.0";
    next-server pmcs00.free.univ.com;
    server-name "pmcs00.free.univ.com";
}
#
# The number crunching nodes.
host n0000 {
    hardware ethernet AA:AA:AA:AA:AA:AA;
    fixed-address n0000.free.univ.com;
}
```

In the case of the Etherboot alternative there should be an extra line, so the file should look like this:

```
# -*-shell-script-*-
# JLdL 22Oct04.
#
# dhcpd.conf for the Etherboot alternative.
#
# The subnet for eth1.
subnet 192.168.0.0 netmask 255.255.255.0 {
    option routers 192.168.0.1;
    filename "vmlinuz-nbi";
    next-server pmcs00.free.univ.com;
    server-name "pmcs00.free.univ.com";
}
#
# The number crunching nodes.
host n0000 {
```

```
    hardware ethernet AA:AA:AA:AA:AA:AA;
    fixed-address n0000.free.univ.com;
    option root-path "/pmc/0000,rw,nolock";
}
```

The string which appears in this extra line as the argument of the option `root-path` will be passed by the boot program to the kernel as the `nfsroot=<string>` boot parameter. This parameter has its complete syntax given by

```
nfsroot=[<nfs-server-ip>:]<root-dir>[,<nfs-options>]
```

As is the norm in this kind of informatics documentation, we use the notation that optional parts are within square brackets. In this example we are using only the NFSroot directory and a couple of options. The NFS server does not need to be given here because it will be also given in the `ip=<string>` explained below.

These are simple versions of the configuration file, with only a single node, the first one we are building. More complete files, with explanatory comments, can be found in the resources area, with names marked by terminations such as `PXELINUX.PMC`, `PXELINUX.TRM`, `ETHERBOOT.PMC` and `ETHERBOOT.TRM`. There are also examples of files mixing nodes using PXELinux with other nodes using Etherboot, marked with the terminations `MIXED.PMC` and `MIXED.TRM`. The first block contains not only the subnet declaration for the interface `eth1`, but a few options giving some of the addresses pertinent to that network. These addresses will be passed by the `dhcpd` daemon to the network boot program at the node, which will use them, as well as the information which is specific to the node, which is given in the second block, to automatically build the kernel boot parameter `ip=<string>`. The string in this parameter will contain all the relevant addresses that the kernel must have in order to auto-configure the network interface during the boot process, as well as the address of the NFS server it should mount its root from. Its complete syntax goes like this:

```
ip=<clip>:<srip>:<gwip>:<mask>:<host>:<ndev>:<conf>
```

where `clip` is the client's IP address, `srip` is the server's IP address, `gwip` is the IP address of the LAN gateway, `mask` is the netmask of the LAN, `host` is the hostname of the node, `ndev` is the network device to be used and `conf` is the IP configuration method. Several of these entries have reasonable defaults and need not be given. The server here is the RARP, BOOTP or DHCP server, where RARP and BOOTP are older protocols which play the same role as the DHCP protocol. Unless a different server is given as part of the `nfsroot=<string>` discussed before, it will be understood that this server is also the NFS server.

The second block uses the hostname of the node as an identifying label, and lists the Ethernet address of the network card of the node, the fixed address of the node (determined from its full name by a name resolution call), the name of the file to be

loaded, the name of the TFTP server to be used to download the file, and the name of the DHCP server itself. In the case of the Etherboot alternative the last option determines a string with an extra set of kernel boot parameters which is to be passed to the node. In the case of the PXELinux alternative this information will be in its configuration files. Only things that are different for each node should be discriminated in this way. The parameter seen here gives the path to the root of the node within the filesystem structure of the server, as well as a few NFS mount options. Later on we will improve this set of options in order to optimize the NFS mounts, both in this file and in the `fstab` files of the nodes, but for the time being it suffices to have this simpler set of options. Once the configuration files are properly written, you may start the DHCP server with the command

```
/etc/init.d/dhcp start
```

The server is now ready to answer DHCP requests from within the private network. Of course, there will be no such requests to answer until we actually build and run at least one node in the private network.

The next thing to do is to install the `syslinux` package, which contains the PXELinux boot loader, and to put in place the PXELinux configuration files, as well as to install the `etherboot` package and the `mknbi` utility for the case of the Etherboot alternative. The PXELinux configuration files will contain the remaining information which the boot program at the nodes needs, in addition to that found in the DHCP configuration file, in order to load the Linux kernel in memory with the appropriate boot parameters, so that it will know how to configure its network interface and how to find its root filesystem through the network. In the case of the Etherboot alternative, one uses the `mknbi` utility to add to the beginning of a normal kernel file a NBI (Network Boot Image) block. This is something like a boot block, containing the information just mentioned and parts of the boot loader program itself.

We will, of course, also need the kernel file itself, which has to be configured and compiled with the appropriate properties, in order that it be able to configure the network interface and mount its root by NFS after it gets loaded into memory. While in the case of the PXELinux alternative all that is needed is to compile the kernel with the appropriate configuration, in the case of the Etherboot alternative the production of an appropriate remote-boot kernel consists of two parts: the compilation of a kernel in usual way, and the transformation of the resulting kernel file into a network-bootable NBI image. On the other hand, the PXELinux alternative poses a security problem which must be solved by the application of a special patch to the kernel prior to compilation. This is due to the fact that the PXELinux boot loader allows you to pass boot parameters to the kernel to be booted, from its boot prompt. This represents a serious security problem, specially for machines located in public rooms, because the `init=/bin/bash` kernel boot parameter can then be used to gain root access to the system of the node without the need for a password. Since the Etherboot boot loader does not provide

such an interactive boot prompt, this is not a problem in that case. However, since the patch can be applied in either case without any problems, and is in any case a good idea from the security point of view, we will adopt it here as part of our standard kernel compilation procedure for remotely-booted systems. We will talk about the kernel compilation process momentarily, but for the time being let us just install the packages and configuration files mentioned above. In order to install the `syslinux` package we execute as usual the command

```
apt-get install syslinux
```

after which it you should not forget to run again the intrinsic shell command `rehash`, so that the shell will find the new commands introduced into the system by this operation. You should then put a copy of boot loader `pxelinux.0`, which you will find in the directory `/usr/lib/syslinux/`, in the directory `/pmc/tftpboot/`, so that the TFTP server will be able to find it. The PXELinux configuration files should be installed inside a subdirectory called `pxelinux.cfg/` in the `/pmc/tftpboot/` directory. There should be a different configuration file for each node, and the naming scheme is to use as the name of the file a hexadecimal representation of the IP address of the node. For example, if we use the IP address `192.168.0.10` for node `n0000`, then the name of the file is build in the following way: for each of the four numbers in the address one constructs a two-digit hexadecimal representation; `192` becomes `C0`, `168` becomes `A8`, `0` becomes `00` and `10` becomes `0A`, so the name of the file in this case will be `C0A8000A`. There is a `gethostip` utility in the `syslinux` package which you can use to build the names from the IP addresses. If you run it with the hostname or IP address of a machine as argument, it will return the IP address and its hexadecimal representation. Here is an example of the PXELinux configuration file for this node:

```
# JLdL 23Aug04.

# General settings.
SERIAL 0x3F8 38400
DEFAULT linux
DISPLAY pxelinux-banner
F0 pxelinux-banner.f0
F1 pxelinux-help.f1
F9 pxelinux.cfg/C0A8000A.f9
TIMEOUT 30
PROMPT 1

# Bootable images.
LABEL linux
  KERNEL vmlinuz
  APPEND root=/dev/nfs nfsroot=/pmc/0000,rw,nolock
  #APPEND root=/dev/nfs nfsroot=/pmc/0000,rw,nolock console=ttyS0,38400
  IPAPPEND 1
```

```
LABEL memtest
  KERNEL memtest86
LABEL local
  LOCALBOOT 0
```

Note that the value of one of the boot options present in this file, the value associated to the parameter `nfsroot`, is the same that appears in the `dhcpd.conf` file of the Etherboot alternative, as the argument of the option `root-path`. The other parameters will be in the NBI boot block of the Etherboot kernel file, as we will see later. In the case of the PXELinux alternative all the boot parameters are discriminated together, in this ASCII configuration file. Note also that the option `console=ttyS0,38400` should be used only for nodes, which have no video cards or monitors. It should *not* be used for remote-boot terminals, in which you expect the kernel messages do go to the VGA console. The file above contains two almost identical lines, one with this parameter and the other without. The line without the parameter is meant for use during the initial tests of the node on the workbench, which will be discussed in the next chapter, since in that case you do have a VGA monitor on the node and wish to see the kernel messages on it. The second line, which appears commented out above, is for the normal operation of the node.

You should now put copies of the appropriate files for your set of nodes inside the directory `/pmc/tftpboot/pxelinux.cfg/`. As usual, you can find a few examples of these files in the resources area, as well as a fully commented version of the file above. Note in particular the values used for the parameter `TIMEOUT` in these example files: this number gives, in tenths of a second, the time until the default boot alternative is tried. We use here a 3-second delay for the first node, a 6-second delay for the second one, and so forth. We are therefore staggering the delays, in order to avoid a situation in which all the nodes start trying to boot at the same time, potentially overloading the boot server and the network, for example when returning from a power failure. It is a very good idea to do this in all clusters, with at least a 3-second increase in the time between any two consecutive nodes, or more comfortably with a 5 to 6-second increase, specially if you have many nodes. It is very easy to implement this in the PXELinux alternative, since all we have to do is to configure a few ASCII files. One can also do this in the Etherboot alternative, but it is a lot more work and trouble, since in that case one must compile and install in each node a different image of the boot program, each one compiled with a different delay. One can devise an automatic multiple-compilation script without too much trouble, but there is still the work of programming each card with a different image and the trouble of keeping track and record of the whole thing.

You will also find in the same directory of the resources area a few examples of the banner and help files used in this configuration. You should make copies of the files `pxelinux-banner`, `pxelinux-banner.f0` and `pxelinux-help.f1` in the directory `/pmc/tftpboot/`, and also copy into the directory `/pmc/tftpboot/pxelinux.cfg/` the `Makefile` you will find there. This makefile is a configuration file for the `make`

utility, and should be left with the standard name `Makefile`. With this file in place you will be able to use the `make` utility to create, out of the PXELinux configuration files, extra help files with a `.f9` termination, for use from within the PXELinux boot loader. You need, however, to adjust this `Makefile` for your set of nodes, because the names of the configuration files of the nodes are listed within it. You may also need to adjust the contents of the configuration files themselves, since they contain their own names and the name of the root directory of the node on the server. Once you have written and adjusted all the configuration files you need, just go to the directory `/pmc/tftpboot/pxelinux.cfg/` and run the command `make` in order to create the corresponding help files. The PXELinux configuration files are written so that a banner with the essential explanations is shown when PXELinux is loaded, and so that the following keystrokes are available at the prompt of the boot loader:

- `[F10]`: displays a more complete version of the initial banner.
- `[F1]`: displays the available boot alternatives.
- `[F9]`: displays the active lines of the configuration file.

As explained in the banner file, for the serial console one must use certain keystroke combinations instead of the function keys: `[Ctrl]-[F]-[0]` instead of `[F10]`, `[Ctrl]-[F]-[1]` instead of `[F1]`, `[Ctrl]-[F]-[2]` instead of `[F2]`, and so on. You may want to write some extra help files and associate them to the remaining function keys. From the examples provided in the resources area it should be clear how to do this, but in case of any doubt you may also consult the Syslinux documentation included with the `syslinux` package, which you will find in `/usr/share/doc/syslinux/`.

The configuration files are also ready to allow you to boot the Memtest86 memory testing program, the same that you have in the server. Just as in the case of the Linux kernel, the format of the bootable file containing the program must be different in the PXELinux and Etherboot cases. The PXELinux boot loader is able to boot the usual files meant for booting from a disk or floppy, for example using the LILO boot loader, while the Etherboot boot loader requires other formats. Hence, in the case of PXELinux you use the usual `vmlinuz` kernel file, and you can use the `memtest.bin` file which you will get if you download the Memtest86 source distribution and compile the program. This file is available in the `memtest86` package of the Debian distribution, which you installed before on the server, where it is located in the directory `/boot/` and has the name `memtest86.bin`. On the other hand, in the case of Etherboot you must use the `vmlinuz-nbi` kernel file which is produced by the `mkelf-linux` utility, and you must use the ELF-format version of the Memtest86 program, which has the name `memtest` when compiled within the source tree of the Memtest86 (version 3.0 or later) distribution. Currently this file is not included in the Debian package, but you will find a copy of it in the resources area, in the subdirectory pmc/tftpboot/, with the name `memtest86-ELF`.

In order to install the `etherboot` package containing the EEPROM images of the Etherboot boot loader, we may now run

```
apt-get install etherboot
```

You should also install the corresponding documentation package, with

```
apt-get install etherboot-doc
```

The binary images containing the Etherboot boot loader will be found within the directory `/usr/share/etherboot/`, and the ones of interest for our case here, for the 3COM and Intel cards we will recommend for the nodes, are currently named `3c90x.rom`, `3c90x.zrom`, `eepro100.rom` and `eepro100.zrom`. The files with the `.zrom` suffix are compressed versions. In order to be able to write one of the `3c90x` images into the flash EEPROM of the 3C905C cards you will need the `cromutil` tool. This can be found in the source tree of the Etherboot project, but seems not to be included in the `etherboot` package. Getting and compiling it is a straightforward matter, and you will find an improved version of the program in the resources area, in the subdirectory `usr/local/src/cromutil/`, including a bit of documentation with the compilation and usage instructions. For the Etherboot alternative we also need the `mknbi` utility in addition to this, so we also run

```
apt-get install mknbi
```

With the tool installed, we will now install in the `/pmc/tftpboot/` directory another configuration file for the `make` utility, which records how to use the tool within the scheme of our cluster architecture. This file should also have the name `Makefile` and should look like this:

```
# JLdL 22Oct04.
#
# Rule to make the NBI kernel image; the first rule is for tests
# _ on the workbench, the second one is for normal operation.
vmlinuz-nbi: vmlinuz Makefile
        mkelf-linux --rootdir=rom --ip=rom \
                --append="root=/dev/nfs" --output=$@ $<
#       mkelf-linux --rootdir=rom --ip=rom \
#               --append="root=/dev/nfs console=ttyS0,38400" --output=$@ $<
```

Note that the white space at the beginning of the last line should be a single `[Tab]` character and *not* a set of 8 spaces. You will find a copy of this file in the resources area. You can see here the boot parameters which are present in the PXELinux configuration file. As was mentioned before, the parameter `console=ttyS0,38400` should be used only for compute nodes, not for remote-boot terminals. Therefore, the first rule of the

two alternative rules you can see within the file should be used when making tests on the workbench, in which case a video card and a monitor are temporarily installed on the node. The second rule is meant for the normal operation of the node.

With this file in place and the kernel file `vmlinuz` of the nodes available, in order to produce the NBI image needed for the Etherboot alternative, all one has to do is to go into this directory and execute the command `make`. This will create within the directory a file called `vmlinuz-nbi`, which is the *target* listed in the makefile, containing the network-bootable kernel and its NBI boot block. This will depend, of course, on the existence in the same directory of the file `vmlinuz`, which is the *dependency* listed in the makefile. This may be a copy of the kernel file which you will compile for the nodes, or it may be a symbolic link pointing to such a file, as we will describe in a little while when we discuss the compilation and installation of the kernel.

The `mkelf-linux` options `--rootdir=rom` and `--ip=rom` which appear in the rules will cause the boot program to assemble and pass to the kernel the boot parameters `nfsroot=<string>` and `ip=<string>`, respectively, using the information it gets from the DHCP server. The string in the option `--append` in the last line of the rules contains a few more kernel boot parameters, in this case those parameters which are the same for all the nodes. Since these will be recorded inside the boot block of the NBI kernel file, parameters which are different for each node should not be included here, so that a single NBI kernel file may suffice for all the nodes. These other node-specific parameters should be included in the `dhcpd.conf` file instead. In the case of the PXELinux alternative this information is contained in the ASCII configuration files, making the whole configuration structure a lot easier to manage.

3.4.3 Preparing the Node for Network Booting

We come now to the issue of kernel compilation. This is a task that should eventually become part of your basic system administration skills. The compilation in itself, as well as the installation of the kernel, are neither complex nor difficult, as you will see. The only somewhat difficult thing is to *configure* the kernel, choosing what device drivers and what internal properties to include or not to include in it. Since there is an enormous number of possibilities, configuring a kernel from scratch can be a lengthy task and requires a significant amount of knowledge about this embarrassingly large number of possible alternatives. You should not be afraid to configure and compile a kernel, it is not all that difficult, it can be fun and it is very instructive. It does not have to be done as root and is also completely safe up to the point that you install and boot it. Even then, so long as the compilation was completed without errors, it is a fairly foolproof procedure and it is easy to take steps to make it completely safe.

Let us first describe briefly the complete procedure for obtaining, configuring, compiling and installing the kernel for the nodes. The kernel will be the one piece of software for which we will not use the available Debian packages. Instead, we will obtain the

source code from the kernel distribution site or one of its mirrors, and will open it up inside the directory `/pmc/usr/0000/src/`. We will then configure the kernel for compilation but, instead of configuring it from scratch, we will use the kernel configuration files which are available in the resources area, in the directory `pmc/usr/0000/src/`, with names such as `Config-2.4.27-JLdL`, which were written to work correctly for a compute node. They assume that the CPU of the node is at least a classic (pre-MMX) Pentium, so they should be appropriate for all modern nodes, with either recent Pentium or Athlon CPU's. They include support for many 100 Mbps and 1 Gbps PCI network cards, including not only the 3COM and Intel cards listed in the examples, but also many other common cards and on-board chips. They do not, however, include support for dual-CPU nodes or for high-memory nodes, that is, nodes with 1 GB or more of RAM.

Since most compute nodes are otherwise very similar, these kernel configurations should be appropriate for your nodes without need of any adjustments. However, later on you may want to learn to use the kernel configuration tools, in order to clean up these configurations, taking off all devices which to not exist in your nodes. It is always better to have a clean kernel configuration, customized for your particular hardware, containing only what is strictly needed, since the kernel will be more efficient and more robust that way. You may also include in the kernel additional capabilities you may need, such as support for dual-CPU nodes or for nodes with large amounts of RAM. It should be noted that for dual-CPU nodes you are advised to use the 2.6 kernels, because you may have trouble booting a 2.4 kernel on a dual motherboard through the network. For the time being, in what follows we will compile the kernel and its modules and install them in the appropriate places in the filesystem structure of the server, so it may be used by the nodes, using the standard configurations found in the resources area. Prior to the compilation we will apply to the kernel source tree the security patch which can be found in the resources area, as explained below.

The complete procedure is started by obtaining through the Internet the tar file with the sources of the latest stable version of the kernel. The version number of the kernel has three parts, such as 2.4.27, where the first number is the major version, the second is the minor version and the third is the bug-fix level. In the stable (that is, non-development) versions the second number is always even. We are currently using major version 2 of Linux, and there are two minor versions available, 4 and 6. The procedure which we will describe here is valid for the 2.4 version of the kernel. Currently the latest version of the 2.4 kernel is 2.4.27, which is the one we will use here. Hence, the first thing to do is to obtain from the primary kernel site or one of its mirrors the file `linux-2.4.27.tar.bz2`, a compressed tar file containing the complete kernel source code tree. At this primary site, which has the address `http://www.kernel.org/`, you will be able to find a list of the mirror sites from which you will be able to download the file by either FTP or HTTP. Lacking a local mirror near you, it is always possible to download the file directly from the primary site. You can download the kernel via

FTP from the central site using the URL

```
ftp://ftp.kernel.org/pub/linux/kernel/v2.4/linux-2.4.27.tar.bz2
```

You can do this using a browser in some X11-capable machine you have access to, or you may do it from your new server, for example using the command `wget` like this:

```
wget ftp://ftp.kernel.org/pub/linux/kernel/v2.4/linux-2.4.27.tar.bz2
```

Having obtained the kernel source file, put it in the directory `/pmc/usr/0000/src/`, which you will have to do as the user root. In principle it is possible to compile the kernel as a common user, only to install it in the system root privileges are needed, but for simplicity we will describe the whole process assuming that you will do it as root. Now go to that directory and make sure that it does not already contain a subdirectory named `linux-2.4.27/`. In case it does exist either delete it or change its name to `linux-2.4.27.OLD/` or something like that. This makes room for you to open the new kernel source tree, which you will do with the command

```
tar -xjvf linux-2.4.27.tar.bz2
```

This will create a new directory `linux-2.4.27/`, recording in this way the complete version of the whole source code tree inside. Due to the use of the `-v` flag of the `tar` command, many messages will be printed on the terminal, one for each file extracted from the tar file, giving you an idea of the size of the whole thing. Since it is standard practice to have the kernel source tree inside a directory named just `linux/`, you should make a symbolic link with this name pointing to the actual directory, with the command

```
ln -s linux-2.4.27 linux
```

The lack of this link might disturb other kernel-related compilations that you may want to do in the `src/` directory. In addition to this, you should fix the user and group ownerships of the kernel source tree, including the directory and all its contents, using the command

```
chown -R root:src linux-2.4.27/
```

Next you should copy into the directory `/pmc/usr/0000/src/` the file containing the security patch mentioned before. The file is named `disable-init-param-2.4.27.diff` and can be found within the subdirectory `pmc/usr/0000/src/` of the resources area. You will also find there versions of this file appropriate for other versions of the kernel.

As its name indicates, the file contains a patch which disables the boot parameter `init=` of the kernel. This is a very simple change which has to be done in the file `init/main.c` inside the kernel source tree. The parameter `init=/bin/bash` is useful

as a way to recover the structure of the disk content of severely broken systems. Since in the case of a remotely-booted system all the disk content is actually on the server, this kind of operation can and should be done at the server, so that this boot parameter has no function in such a system. In the case of the server, which has all the disk content, neither should you disable this parameter, nor is it a problem. This is so because the server uses the LILO boot loader, which allows for a password to be set to protect a boot done with the inclusion of parameters at the boot prompt and, in any case, all servers should be locked in a room to which only authorized personnel have access. In order to apply the security patch to the kernel source tree you just created, you should now run, in the `/pmc/usr/0000/src/` directory, the command

```
patch -p0 < disable-init-param-2.4.27.diff
```

We are now ready to start the configuration and compilation process. You should now put inside the directory `linux/` (which is the same as the directory `linux-2.4.27/`, of course) a copy of the configuration file `Config-2.4.27-JLdL` which you can get from the subdirectory `pmc/usr/0000/src/` of the resources area. After this, get into the `linux` directory with `cd linux/` and make there another copy of the configuration file, this time using its standard name `.config`, which is the name expected by the compilation structure. You may now use this file to configure your kernel executing the command

```
make oldconfig
```

The keyword "oldconfig" is a target of the makefile structure of the kernel source tree, which will cause it to configure the whole tree according to the file `.config` that you put there. A lot of message lines will appear on the screen, scrolling very fast and finishing with a message to the effect that the kernel has been configured without any errors. Next you should run the command

```
make dep
```

This is another makefile target, which will determine and record all the dependencies among files within the kernel tree. Once again you will see many message lines scrolling very fast on the screen. We are now ready to do the actual compilation of the kernel. You may now create a compressed kernel image with the command

```
make bzImage
```

This one will be a much lengthier process, as all the parts of such a complex program as the Linux kernel get compiled and linked together. The last step of the compilation itself, before we start the installation, is to make all the additional drivers which were configured as modules, with the command

```
make modules
```

Each one of these two last commands will take a while to complete. Keep an eye on the extensive output which is shown in the screen and make sure that the each one of them completes without error before trying the next one. In case there is an error, the `make` command will abort the compilation and stop at the point where the error occurs. If this ever happens take note of the error messages. Such errors are very rare and it is extremely unlikely that there really is something wrong with the kernel, usually any errors that eventually happen are a consequence of one of the following three things: the absence of some part of the development environment in your system; the inclusion of an inconsistent set of options in the kernel configuration file; hardware problems in your system, since a kernel compilation is a fairly severe test of the hardware. The first two causes should not happen if you followed correctly all the steps to this point. If you discover that you have a hardware problem, possibly a bad memory module, you must see to that before you continue, of course.

If all went well up to this point, we are now ready to start to install the kernel and the modules in the appropriate places. We will start by the modules. In order to put them in the correct place, which is the directory `/pmc/0000/lib/modules/2.4.27/` inside the filesystem structure of the node and *not* the corresponding directory of the server itself, which is the default location encoded in the makefile, you must now define a special environment variable with a certain value, using the `tcsh` command

```
setenv INSTALL_MOD_PATH /pmc/0000
```

You should be very careful and never forget to do this whenever you are compiling a new node kernel on the server, otherwise your node modules may get mixed up with the modules of the server, with results that cannot be good. Having done this, and while still in the compilation directory, you may now install the compiled modules in their proper place using the command

```
make modules_install
```

This command will create the directory `lib/modules/2.4.27/` inside the prefix directory `/pmc/0000/` defined by the environment variable, and will put all the compiled modules in appropriate subdirectories inside it. It will also run the `depmod` command in order to record all the dependencies among modules. Next you should install the kernel symbol map in the boot directory of the node. While still in the compilation directory, you may do this with the command

```
cp -p System.map /pmc/0000/boot/System.map-2.4.27
```

Following this you should copy the compressed kernel image inside the `tftpboot` directory, using the command

```
cp -p arch/i386/boot/bzImage /pmc/tftpboot/vmlinuz-2.4.27
```

Note that the name of the kernel file should be `vmlinuz` and not `vmlinux`, indicating that this is a compressed kernel which will automatically decompress itself during the boot process. Observe also that we have recorded the version of the kernel in the name of the file. You should now go to the boot directory of the node and delete from there any old versions of the files `System.map` and `vmlinuz` that may be there, having been inherited from the server. Since the node is going to boot via the network, it does not need any copies of the kernel file in this directory. If you are using the Etherboot alternative, you should go next to the `tftpboot` directory in order to make the NBI kernel image. First you should make inside this directory a soft link `vmlinuz` pointing to the actual kernel file, using the command

```
ln -sf vmlinuz-2.4.27 vmlinuz
```

This will make the link `vmlinuz` which is expected both by the PXELinux configuration files and by the makefile you copied before inside this directory, overwriting an older file of the same name that may already exist there. This link will serve to record the version of the kernel which is in use for the nodes, and completes the task for the case of the PXELinux alternative. All that remains to be done now for the Etherboot alternative is to run the command `make` inside this directory, which will create the `vmlinuz-nbi` file containing the NBI kernel image. This completes the kernel compilation and installation process. Please note that what we have described here is not a standard kernel compilation, valid for a generic machine, but a modified procedure, appropriate for remote boot nodes. The scheme adopted is not the only possible one and we will in fact use a slightly different one for the remote boot terminals. The installation of a kernel on the server itself, as well as on any other machines booting from disk, will be somewhat different and will be described in detail later.

Now the only thing still missing so that an actual node can boot through the network is the boot program residing in the boot ROM (Read-Only Memory) chip of its network card. There are several different ways in which remote booting can be accomplished in the i386 architecture, depending on the hardware that you have. We will describe in detail, here and in what follows, two different approaches, one based on the PXE protocol, which seems to be establishing itself in recent times as the de-facto standard for network booting, and one based on the more traditional Etherboot program. In the latter case we will discuss only a single case, which is a particularly convenient one, based entirely on free software. On the other hand, it is valid only for the 3COM line of model 3C905 network cards. In fact, in practice it will probably only be used for the current model of this line, the 3C905CX model which comes with an on-board flash EEPROM (Electrically Erasable Programmable Read-Only Memory) chip. This is the one of the models of network card which we will be recommending when we discuss the hardware of the nodes in the next chapter, the other being the Intel EtherExpress Pro/100. The PXE-based alternative works for both these cards.

Before we proceed, let us discuss briefly the available alternatives for the hardware and for the software involved in remote booting. True remote-boot systems could be characterized by being completely devoid of any kind of magnetic or optical media and by having a small network boot program recorded in an on-board chip. These systems do not have any moving mechanical parts other than cooling fans, which is one important reason contributing to their good reliability and ease of maintenance. It will be even better in that happy day in the future when one will be able to do away with the cooling fans as well. In truth one should go a bit beyond that in this definition. It is possible that in the near future one will be able to boot a system from a USB memory stick or from a full-fledge solid-state disk, both of which have no magnetic or optical media and no moving parts. However, these are solid state storage technologies with high and persistent storage capacities, in which one would typically have full operating systems or user data, and which can be read from or written to just like a normal disk. Large capacity and constant read and write I/O operation imply the need for software maintenance of the data content.

The idea of remote-boot clusters is that the permanent data content of each node is reduced to a bare minimum, a *small* program used in read-only mode to do only a very definite thing: boot the system from a remote server which has all the actual data content. To establish what is meant by small here, we may mention that the typical size of a binary image containing a (possibly compressed) remote boot program is around from 16 kB to 64 kB. In this way the node never needs any local software maintenance, and all such maintenance work in centralized on the server. Therefore, in such systems the boot program will always be in some small ROM chip on-board the hardware of the node. Let us overview the existing possibilities for the nature and operation of this kind of chip.

- **PROM:** Programmable Read-Only Memory; a chip that can be programmed (written to) only once; it usually comes soldered to the board of the network card, and is typically used by manufacturers to install in the card their own proprietary boot programs; sometimes cards which usually would come with a truly programmable boot chip will actually come with such a PROM chip in their O&M versions, which therefore cannot be re-programmed; unless the chip comes with some program that can be used to boot Linux, as we will discuss later on, network cards with this kind of chip are not appropriate for our cluster architecture.

- **EPROM:** Erasable Programmable Read-Only Memory; a chip that can be erased by exposition to ultra-violet light and then re-programmed by means of a special piece of equipment, an EPROM recorder, also known as a programmer or burner; the network cards which use this kind of chip come with a socket where you can insert it after you use the EPROM burner to program it; many older network cards use this kind of chip, in fact it is the traditional way of doing remote booting; many recent network cards still use this technology, usually the cheaper ones; the

Etherboot project distributes boot program images for many cards of this type, new and old; you may be able to buy chips pre-programmed with Etherboot for your network card, thus avoiding the need for an EPROM burner.

- **EEPROM:** Electrically Erasable Programmable Read-Only Memory; a chip that can be erased by electrical means and then re-programmed, in general by means of an EEPROM burner, which also does the erasing, and which usually can also deal with EPROM's; this technology eliminates the need for UV lights, but usually still depends on the availability of special equipment for erasing and programming the chips; usually the cards which can use EPROM's can also use EEPROM's, in the same type of socket; although it is in principle possible to have a network card capable of programming a chip of this type which comes soldered on-board, we currently know of no examples of this.

- **Flash EEPROM:** a modern, faster type of electrically erasable programmable read-only memory; it comes in a much more compact shape than the traditional chips, either soldered to the board or inserted into a small socket on it; it can be programmed on-board the card, by means of appropriate software, and eliminates the need for a special chip burner; this kind of chip is also used for the BIOS of many motherboards; the current tendency is for this kind of chip to supersede all others for this type of purpose; modern network cards such as the 3COM 3C905CX and the Intel EtherExpress Pro/100 come with a chip of this type, or with a socket for it; in the case of the 3C905CX card it is possible to program the chip with a freely available open-source program that runs under Linux, and this is one of the alternatives which we will describe in detail in this book.

So much for the hardware. In terms of the software involved, there are basically two practical alternatives, the use of the Etherboot boot program or the use of the PXE protocol which was introduced by Intel and is now available in many network cards. While the Etherboot program was designed to be able to boot the Linux kernel directly, this is not the case with PXE programs, which cannot boot the Linux kernel directly. The traditional way to do remote booting for Linux is to use the Etherboot program, but now there is a practical and easily implementable way to use the PXE protocol, by means of the PXELinux program, and this is the approach we will adopt as our standard recommendation here. However, once more let us overview the main alternatives before we proceed.

- **PXE and PXELinux:** PXELinux is a network boot loader which is distributed as part of the Syslinux project; it can be loaded by the node by means of the PXE protocol, if the node has a PXE-enabled card, containing a manufacturer-supplied PXE boot program; since the PXE protocol is now available in many network cards, this may be the ideal solution in many cases; all the main network cards in the market support the PXE protocol, in particular the 3COM model

3C905CX and the Intel EtherExpress Pro/100, which are the ones we recommend elsewhere in this book; it is quite convenient because it requires only a rather simple configuration setup on the DHCP server; instead of being located within the `dhcpd` configuration file, all the node-specific configurations will be located in a set of PXELinux configuration files in the TFTP server.

In this case one configures the DHCP and TFTP servers to send to the node the `pxelinux.0` binary image, instead of the kernel directly; the `pxelinux.0` program then downloads from the TFTP server a node-specific configuration file which contains all the information it needs in order to download from the server (again via TFTP) and load into memory the Linux kernel and its boot parameters; the idea of downloading an ASCII configuration file via TFTP is a brilliant one, which simplifies everything and makes for much more flexibility; the resulting structure is highly configurable and easy to configure.

- **Etherboot:** the traditional procedure was to get the appropriate Etherboot program written into an EPROM chip on the network card; while this was not a problem with the old EPROM's or EEPROM's to be inserted in sockets, so long as a standard EPROM or EEPROM standalone programming equipment was available, it might represent a problem for on-board flash EEPROM's due to the lack of the appropriate tools for writing the Etherboot program into the on-board chips; tools of this kind for running under Linux are still very rare, and the more common standalone DOS programs distributed by the manufacturers of network cards will not always be usable for the Etherboot binary image.

  For the case of the 3C905CX card there is an open-source program to do this under Linux, which is the ideal situation; in some other cases there are standalone DOS programs distributed by the manufacturers to do this kind of job; however, they might refuse to write a program that, from their point of view, is unknown; usually they are distributed to write only the proprietary program of the manufacturer, they will probably check the image before writing it and, as they will not be able to identify it, they may refuse to proceed; sometimes it is possible to write the program into an EEPROM chip on the motherboard instead, using the flash BIOS maintenance program distributed by the manufacturer on the motherboard.

- **PXE and Etherboot:** the Etherboot program can also be prepared so it can be itself loaded through the network; it can then be chained with PXE, that is, PXE can be used to download the network version of the Etherboot image, which then downloads the kernel and proceeds as usual.

  The major drawback of this approach is that it requires a more complex DHCP configuration of the server, because it will receive from the same node two different sets of DHCP requests, one from PXE and another one from Etherboot; since it must answer in different ways to each set of requests, it must be able to identify

the program sending the requests.

If you are going to use the PXELinux alternative, you should make sure that you have at least version 2.0 of PXE on the network card, since some 1.0 versions may not work with PXELinux. It is usually possible to upgrade an older card with tools supplied by the manufacturer and freely available in its web site. Usually the card manufacturers also distribute standalone DOS configuration tools which you can use to configure the cards, doing things such as enabling and disabling the boot ROM, choosing the PXE protocol among several possibilities, and other things. This may be necessary regardless of the alternative you choose to adopt, so you should always get from the manufacturer's site the available tools for the card you decide to use. Usually these tolls can also be found in a CDROM media which should be supplied with the card, which contains also drivers for several operating systems and hardware documentation.

The PXELinux alternative has many advantages, specially from the point of view of ease of implementation. Since it does not involve writing any images into the boot ROM of the network cards, a whole set of technical problems is avoided altogether. Besides, the whole configuration structure is much simpler and easier to set up and maintain, both in regards to the `dhcpd` configuration and the ASCII configuration files which are specific of the PXELinux program. The use of ASCII configuration files is a major advantage, which makes the PXELinux program flexible and easy to configure, eliminating the need for recompiling and reinstalling either the boot loader images or the NBI kernel image in order to change configuration parameters. You can configure things like booting timeout delays, the use of the serial console, and several different boots including a local boot, simply changing the PXELinux configuration files. You can also manually pass parameters to the kernel during boot and configure useful help messages and banners. Since most Gbit cards also are being released with PXE support, it is likely that the PXELinux alternative will be instrumental in facilitating the migration of nodes from 100 Mbps to 1 Gbps, which will happen in the near future. In fact, it is already known that PXELinux works just fine with the 3COM model 3C2000 card, a 1 Gbit card for a 32 bit, 33 MHz or 66 MHz PCI bus, which looks like quite a good choice for a Gbit node.

All this makes the PXELinux alternative unbeatable from the point of view of the end user, so long as the network cards to be used on the nodes support the PXE protocol natively, as they come from their manufacturers. The Etherboot alternative can only be recommended in cases where the PXELinux alternative will not work for one reason or another. It can be used without trouble for older network cards, which do not support flash EEPROM's, so long as a standalone EEPROM burner is available. In the case of modern network cards the continued use of the Etherboot program in native mode, written into the flash EEPROM's of the cards, will depend on the development of appropriate utility programs to do the writing, preferably running under Linux, such as the `cromutil` utility. The possibility that card manufacturers will start distributing standalone DOS programs to do this anytime soon seems very remote, but

free standalone DOS programs of this type could also be developed. The possibility that some day card manufacturers will distribute their cards with something like Etherboot pre-installed seems even more remote.

Currently one can only say that there is a strong tendency for the PXE protocol to become the de-facto standard for remote booting. In this case the most promising standard for the future seems to be the use of the PXELinux boot loader in Linux clusters. If this tendency realizes, it is unlikely that the Etherboot solution will survive on the long term. This is often the case in the informatics revolution, when an open protocol becomes a de-facto standard: all other solutions tend to disappear in favor of those which adhere to the de-facto standard, independently of the intrinsic technical merit of each solution. One says that the de-facto standard becomes a "giant component" of that particular sector of the market, dwarfing all others.

Still, one should keep in mind that all this is technology in development and that, in order to find the best solution for any given situation at any given time, the best thing is to keep in contact with the free software projects involved. One of the best sources of information on this subject is the Etherboot project, which is located at the URL

`http://etherboot.org/`

or, since the project is hosted at SourceForge,

`http://etherboot.sourceforge.net/`

The project offers plenty of documentation and a mailing list. The site of the Syslinux project is also very useful, it is located at

`http://syslinux.zytor.com/`

and the page about the PXELinux network boot loader can be found at

`http://syslinux.zytor.com/pxe.php`

Some known hardware problems with PXE cards which affect PXELinux, as well as possible workarounds, can be found in

`http://syslinux.zytor.com/hardware.php`

There is also the Netboot project, which is in fact the oldest one created for supporting network booting on Linux. Its software is similar to that of the Etherboot project, providing ROM images for network cards and using the same NBI architecture. Unlike the Etherboot project, which develops its own drivers for the network cards, the Netboot ROM images use the DOS drivers for the network cards. In some cases these drivers are not free and hence cannot be distributed by the project. The project is hosted at SourceForge and has its web site at the address

`http://netboot.sourceforge.net/`

Finally, there is the Nilo project, a new project which evolved from the Etherboot and Netboot projects. Both this new Nilo project and the older Netboot project provide ROM images to be written into the boot ROM of network cards, including PXE-compliant images for most cards which have a driver under Linux or FreeBSD. The home page of the Nilo project is located at

`http://www.nilo.org/`

or, since this project is also hosted at SourceForge, at

`http://nilo.sourceforge.net/`

There is also plenty of information included in the corresponding Debian packages, which will be available in your system as soon as you install them. Although finding a practical network boot solution for a particular situation may sometimes be a somewhat tricky thing, there is plenty of documentation available and plenty of remote help can be found through the Internet, so if you persevere you will eventually find your solution despite any difficulties which you might encounter along the way.

3.4.4 The First Network Boot With PXELinux

From the point of view of the software structures on the server and in the filesystem of the node, by now we are essentially ready to try remote-booting a real node. We may even venture trying this immediately, so long as we use the PXELinux alternative. For the Etherboot alternative we must still write the boot program into the network card, and the issue of the writing of EEPROM's will be delayed until the next chapter, after we discuss the hardware for the nodes. Besides this, there are still a few relevant hardware issues on the node, which will only be discussed in detail in that chapter, and which may impair the network booting process in either case. In the case of the PXELinux alternative, however, with a bit of luck, we may be able accomplish our first network boot at this stage, which will help to establish that all the basic structural elements are in fact soundly in place on the server.

This first time around the remote booting game usually involves great expectations, and even temporary failures can be a bit traumatic. Keep in mind that there is the possibility that quite a bit of confusion will be present the first time you try this. Hence the best strategy is to take it easy and do things slowly and one at a time. Having reached this point, however, the temptation of immediately hitting the power button on the node to see what happens may be irresistible. Well, do that if you must, but be prepared to try the whole thing more than just once. Living dangerously is part of the free software hacker's life style, so do feel free to play around with this. If you get

into too much trouble here, there is no reason for despair, but it may be a good idea to wait until after the hardware of the nodes is discussed in the next chapter, before trying again. There is another subsection about booting the first node in Chapter 4, in the latter part of Section 4.1.

In order to try network booting a node you must have some hardware for the node, of course, but it does not have to be one of your definitive compute nodes or remote-boot terminals. No matter what final use you intend for your cluster, right now all you need is a spare motherboard and a few more parts you can use temporarily for a few tests. In order not to have to worry too much about special hardware settings, we will do the tests using a video card and a monitor installed on the node, as well as a keyboard. This is, in short, the first time you will use your *workbench*, which may be just some table space somewhere, near the necessary power and network connections, and what you will boot on it will be a primitive version of a remote-boot terminal. If your node filesystems are set up for compute nodes, a few special temporary settings will be needed on the server in order to fully support the remote booting tests we are about to undertake. This is due to the fact the system settings for compute nodes do not support the use of a video card and monitor.

Let us start by listing the hardware you will use for the tests. What you need here is a motherboard with the processor and RAM assembled on it. Any CPU of the i386 architecture starting with a classical (non-MMX) Pentium will do, and any amount of RAM above 32 MB will be more than adequate. The motherboard may or may not be assembled in a cabinet, but you need a power supply and cables, of course. You will also need a network card, a video card, a monitor and a keyboard. All that is required of the network card is that it be a 100 Mbps PCI card and that it have an operating PXE boot program on its boot PROM. Any video card will do, either PCI or AGP, with any amount of video RAM, because we have no intention of running the X11 graphical system on this node, and the monitor will be used only in text mode. The monitor can be any standard VGA monitor. Assembling this set of components is a fairly simple task, not unlike assembling the server, as described in Chapter 3. A few more comments on this issue can be found in the hardware sections of Chapters 4 and 5.

Besides defective hardware, which is a real possibility if you are using old spare part for the tests, the most common reasons for remote booting failure are that the motherboard may not be set up to boot from the network, and that, although the network card does have a boot PROM, it is disabled in the configuration of the card. It may also happen that the PXELinux boot loader will not itself be loaded through the network due to the version of the PXE boot program on the card being too old. Modern cards usually come with a recent version of PXE enabled by default, so there should be little problem with that. Older ones, however, may give you a bit of trouble until you manage to make them work, if they have PXE at all. Let us now go through the sequence of operations involved in performing the network boot tests.

First of all make sure the server is up and running in multi-user mode, with all

daemons operating and, therefore, all services available. You will need the DHCP, TFTP and NFS network services. The configuration of the boot command line of the kernel should be *without* redirection of the console to the serial port, that is, without the parameter `console=ttyS0,38400`, as shown in the examples in the previous subsections for the PXELinux configuration files within the subdirectory `pxelinux.cfg/` of the `/pmc/tftpboot/` directory of the server. If your node filesystems were configured for compute nodes, you must temporarily undo the reconfiguration of `etc/inittab` file in the root filesystem of the node. You should save the modified file with a different name such as `inittab.CURR` and put back in place the original file you saved before with the name `inittab.ORIG`.

Next you should assemble the test node on the workbench, including the video card, the monitor and the keyboard. If you are doing the boot test in the external LAN, connect the network card of the node to it. If, on the other hand, you are doing the test in the private LAN of a PMC, make sure that the second network card of the server is connected to the switch of the private network, and connect the node to it as well. With the assembly of the node completed, turn it on, look intently at its monitor and hope for the best. Depending on various circumstances, it is just possible that the network boot will happen, in this case you should see on the screen the reset of the hardware (POST), then some messages saying something about a search for a DHCP server. IF this is the case you are in luck and you can skip the next few steps, going directly to the step where you reconfigure the DHCP server in order to record within it the Ethernet address of your card.

More likely than not, however, the system will not boot. If it ends up stopping with a message warning that no system was found, reboot the hardware of the node with `[Ctrl]-[Alt]-[Del]` and while the POST runs again, enter the setup program of the BIOS of the motherboard, pressing whatever key is needed for than in your case, probably the `Del` key. There is a single thing to be done in the BIOS at this stage, which is to configure the motherboard to try to boot through the network. Find the boot menu of the setup program and check whether or not a network boot alternative is available. Older motherboards may not have such an alternative, in this case the presence of a network-boot enabled card in the system should be enough to force its use instead of any boot options configured via the menu. If the motherboard does not have a network alternative in its boot menu and the machine did not try to boot through the network, then probably either the boot PROM of the card or its PXE program is disabled. Most recent motherboards will not only have a generic network boot alternative, they will also detect the particular boot program at the network card and include it in the menu. If such a motherboard does not detect the boot program of your card, this may also mean that the boot program of the card is somehow disabled.

If the boot menu does have a network alternative, either generic or specific, activate it as the first boot device and try the network boot again, rebooting the hardware after you save the new BIOS configuration. If this time it does try to boot you are finally in

luck and you can skip directly to the step where you reconfigure the DHCP server in order to record within it the Ethernet address of your card. If you still cannot see the machine trying to boot through the network, you must configure the network card itself for remote booting. Network cards may be configurable either by means of a standalone configuration program, typically run by booting from a DOS floppy, or by means of an on-board setup program. If there is such a setup program, a message should appear on the screen during POST telling you what keys to press in order to enter it. For example, Intel cards such as the EtherExpress Pro/100 use the keystroke combination `[Ctrl]-[S]`, while 3COM cards with that manufacturer's MBA (Manageable Boot Agent) software, such as the 3C905CX, use `[Ctrl]-[Alt]-[B]`. Sometimes the setup program does only some things, and other must be done by means of a standalone program.

The standalone configuration programs can usually be obtained from the manufacturer's web site. If you have trouble obtaining them and then producing a suitable bootable DOS floppy containing them, it may be the time to accept that you are out of luck and to go on to read the details about all that in Chapter 4. If you are able to run a configuration program of some sort, there are two things to look for: a menu for activating the on-board boot PROM, and a menu for activating the PXE boot loader, possible among other network-booting alternatives. This last one may come also in the guise of "enabling network boot" or making it the default. If you are able to do these things, your node should now be able to boot through the network, barring the possibility of defective hardware and possibly an old version of the PXE boot program. For example, the PXE version 1.0 in Intel cards is known not to work with PXELinux, which will require you to do an upgrade of the software on the card. This can usually be done with tool downloaded from the manufacturer's site, but if you get to this point and do not know how to do that, once again it is time to go read Chapter 4.

Assuming that all went well with you so far, you should try again to boot your node, once you have saved the new configurations of your network card. Once the machine starts trying to boot via the network and the messages about the search for a DHCP server show up, the Ethernet address of the card should be clearly visible on the screen. You should now write down this hardware address and then record it in the `dhcpd.conf` file of the server. Then restart the DHCP server with the command

```
/etc/init.d/dhcp restart
```

After this you should reboot once more your node, and this time you should see the PXELinux banner come up after the POST and the search for the DHCP server. If this still fails to happen, even after you check carefully all the relevant elements on the server, then you may have some hardware problem or an old version of the PXE boot loader. The moment you get the PXELinux boot menu on the screen, however, several elements have been proven to be in working order, including the network card itself, the

network connections, the DHCP server and the TFTP server. The actual boot of the kernel will show whether or not the remaining elements are correctly in place.

Assuming that your PXELinux configuration files follow the examples given before and hence that the default boot alternative within it is to boot Linux through the network, hitting the `[Enter]` key at this point will start the boot of the kernel. You should see a couple of lines reporting that the kernel is being loaded and decompressed, then the kernel boot messages should appear. They should be similar to the ones you already saw when you booted your server. If the kernel messages do not show up in the monitor, this is indicative that you mistakenly have the boot parameter `console=ttyS0,38400` in your configurations. The kernel should detect all the hardware and then mount its root via NFS. If you get a message saying that there was a kernel panic and that it was unable to mount its root, something is wrong with the NFS service of your cluster server. In this case you should check carefully all the relevant configurations, including the kernel boot parameters in the PXELinux configuration file. Check also that all other necessary files and services are correctly in place.

If all goes well the system setup routines should execute to the end and then you should see a login banner and prompt on the screen. If this does not happen, this is indicative that you have the modified `/etc/inittab` file in your node, rather than the standard one. Once everything is working properly, log into the node using its keyboard and monitor, and look around the system. You will have to do this as root, since so far we have no other users registered within the system. Use `df` to see the system mounts via NFS, including the system root. Do a `ps aux` to see what is running on the node, as well as a few `ls` commands executed here and there. Do a `dpkg -l` to look at the installed software packages. What you have here is a remote-boot node functioning as a simple text terminal. Congratulations, you have just established control over the central technology which will be one of the foundations of the architecture of your cluster.

3.4.5 Preparing for Full Diskless Operation

As should have been established in practice by the booting experiment undertaken in the previous subsection, at this point all the basic technical elements needed for remote-booting a real node are in place, at least in the case of the PXELinux alternative. Even in the case of the Etherboot alternative not much is missing, just the technical challenge of writing the boot program into the EEPROM, to be undertaken in the next chapter. However, even if we do boot a PMC compute node right now, it still will have no direct access to the Internet from inside the private network. This would prevent us, for example, from updating the software package databases of the nodes and from installing new packages in them running the `apt-get` utility on the nodes, as well as from using any network services provided by machines in the external LAN. Besides, a few additional services would also be missing, preventing us from operating the nodes in full multi-user mode, with actual users able to use the cluster for production purposes.

While in the case of the installation of software packages on the nodes the unavailability of access to the Internet could be circumvented by the use of the `chroot` command on the server, other network services on the external network, such as the importation of a user home disk from an external server, may be essential for the operation of the nodes. Although in many cases it is possible to circumvent the unavailability of the Internet by the installation of the necessary services in the server itself, it is often inconvenient or wasteful to do so, so that it is usually very useful to have a way to access the Internet from within the private network. In any case, usually this is a problem only for the compute nodes of a PMC, since they are typically in a private network. Remote boot terminals will usually be on the external LAN and therefore directly on the Internet, although they also could be configured within a private network. We will now discuss and put in operation each one of the additional structures and services needed for the nodes to be able to operate as fully as possible.

Let us first talk about the *masquerading* structure on the front-end server of a private network such as the one usually employed for a PMC. This is a packet-manipulation structure which operates in the kernel of the server. Since the PMC nodes are in a closed local network which is not connected directly to the external LAN and therefore not connected directly to the Internet, the only way in which a node in the private LAN can exchange network packets with a machine in the outer LAN is through the intermediation of the server. Let us recall that in this case the server will have two network interfaces, one to the internal LAN and the other one to the external LAN. In order that the nodes be able to communicate to machines on the Internet the server must be able to pass packets from one interface to the other and hence to function as a *router* between the private network and the Internet, reading and identifying the destination addresses on the packets in order to forward each packet to the appropriate interface.

This is, however, not all that is needed, because packets containing addresses of a private network may not travel on the Internet. In order to establish communication between a node in the private network and a machine on the Internet, the server will have to *masquerade* the packets it receives for forwarding from the private network, before it sends them out to the Internet, making them look like they are packets of its own, using its own Internet address as the source (sender) address. This involves changing addresses in the headers of the packets, and also involves keeping track of the *connection* to which each packet belongs, so that when an answering packet is received from the Internet, the server may identify it as such, re-address it and forward it to the node that sent the original packet. In this way each node will be able to communicate transparently to machines in the outer LAN, essentially as if the private LAN was part of the Internet. From the point of view of the machine on the Internet, it will appear that the communication is going on with the front-end server instead of the nodes.

The one limitation of this scheme is that only *answering* packets can be forwarded from the Internet to the private network, so that machines on the Internet cannot initiate

connections to machines in the private network. Connections can only be initiated by machines in the private network. By and large, this is all that is needed in order for nodes in the private network to be able to use services provided by machines on the Internet. The nodes will not be able to export services to the Internet, but it is seldom the case that this is necessary, due to the nature of the role of such nodes. In fact, it is actually possible to circumvent this limitation by means of extra routing structures on the server, which may allow exporting to the Internet network services which are installed on a node, making it look like they are services installed on the server but, again, this is seldom necessary and will not be discussed now. Only if you decide to install remote-boot terminals on a private network it is conceivable that things like this may become necessary on occasion, and a few examples will be discussed in later chapters.

In order to set up the masquerading structure three things are needed: kernel modules, a support package and a startup/configuration file. The standard Debian kernel package which you installed on the server during the initial installation of the system already includes all the necessary modules. The internal kernel structure which is used for this kind of routing has the name *IP tables*, and includes modules for all the basic functions needed: *packet filtering*, for identifying the packets which are to be manipulated; *network address translation* or NAT, for editing the addresses in the packets; *connection tracking*, for keeping track of the connections and the corresponding network ports, so that multiple dynamic connections can be handled simultaneously, without mixing up all the packet traffic.

Some network protocols, such as FTP, require additional packages in order to work under the masquerading structure, usually due to the fact that they involve an incoming connection as part of their normal operation. In the case of the FTP protocol, the client (in this case, the node) starts a control connection to the FTP server (in this case, some machine on the Internet) but, later on, when a file transfer is started, the corresponding data connection is usually started by the server. Without some additional structure this incoming connection would not be completed by the masquerading structure, hence the need for a special kernel module. Some other protocols and services also present this kind of behavior and require special kernel modules or user-space programs running on the front-end server, but none of them are of any interest to us here. The support package provides programs that enable you to manipulate the internal kernel data structures involved in all this. It is called `iptables` and you may now install it with the usual command,

```
apt-get install iptables
```

With this in place all that is still missing is a startup script to set up the masquerading structure during boot. It will also double as a kind of configuration file for your masquerading setup. Here are the active lines which should be in this file, with an explanation of the function of each one, in our case here, in which we assume that the

interface `eth0` connects the server to the external LAN and hence to the Internet, while the interface `eth1` connects the server to the private network in the internal LAN. First, the following lines load the extra kernel modules enabling NAT and connection tracking for the FTP protocol, since these modules, unlike the others we will need, are not loaded automatically by the kernel:

```
modprobe ip_nat_ftp
modprobe ip_conntrack_ftp
```

The following line will act on the "nat" IP kernel table, adding a new rule to its POSTROUTING chain of rules, in order to enable the masquerading of any network packets which are forwarded from any input interface to the output interface `eth0`, using the IP address of the server as the source address:

```
iptables -t nat -A POSTROUTING -o eth0 -j SNAT --to-source 172.16.129.30
```

The following line will act on the default "filter" IP kernel table, adding a rule to its FORWARD chain, in order to enable the forwarding of any packets from the input interface `eth1` to the output interface `eth0`, that is, going from the private network to the Internet:

```
iptables -A FORWARD -i eth1 -o eth0 -j ACCEPT
```

The following line will also act on the "filter" IP kernel table, adding a rule to its FORWARD chain, in order to enable the forwarding of any packets which are part of a known connection, from any interface to any other interface:

```
iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
```

The following line will also act on the "filter" IP kernel table, defining the default policy rule to reject all other packet forwarding operations:

```
iptables -P FORWARD DROP
```

Finally, the following line will edit a kernel parameter in the `/proc/` filesystem, in order to actually turn on its IP forwarding capability:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

This set of packet manipulation rules will implement masquerading as described before. A complete and fully commented version of this file can be found in the resources area, in the subdirectory `etc/rc.boot/`. You should now create the directory `/etc/rc.boot/` in your server and copy in it the file `set-ip-masquerading` which you will find in the resources area. Note that this file should have its execution access bits turned on, which you can ensure with the command

```
chmod +x set-ip-masquerading
```

All the executable files found in the directory `/etc/rc.boot/` will be executed once during the system boot, right after the scripts linked from the directory `/etc/rcS.d/`. In this way the masquerading structure will be automatically set up at every system boot, even if you boot in single-user mode.

We tackle next the configuration of the Network Time Protocol (NTP). All the packages involved in this were already installed during the initial system installation. This protocol has the objective of keeping the local clocks of all machines on the network in tight synchronism. Apart from the obvious convenience of having all system clocks in a local network showing the same time, the implementation of a good level of clock synchronization is made necessary by the sharing of files among the systems. If the clocks are not very well synchronized, programs in various machines across the local network may get confused by the fact that the files they access through the network appear to bear time stamps in the future of the local system time, due to clock skew. Programs that use the time stamps of files in their operation may fail to operate properly. The `make` utility, for example, is affected by this. All time records which are kept internally by the system, as well as the time stamps of files, use the coordinated universal time (UTC), not the time of the local timezone. The hardware clock should always be kept in UTC and it is this reference time which is in fact synchronized across the whole Internet.

The NTP system works in a client/server architecture. The clients access the servers in order to adjust their local clocks in accordance with the clocks of the servers. The servers are organized in a hierarchical structure in which each level of the hierarchy is called a *stratum*. It is also possible to have two or more systems on the same stratum work as peers instead of in a client/server relationship, but it is far more common to use a plain client/server architecture within a LAN. The machines within a certain stratum may work as servers for other machines in stratum below them, and use as their own servers machines in a stratum above them. So one sees that each machine may function as both a client and a server, in this hierarchical fashion. Within a cluster one usually uses the cluster server as a NTP server for the nodes, while the cluster server itself uses as NTP server one or more external machines. For example, it is possible that the institution you are in maintains a central NTP server which you can use to synchronize your cluster server. There are also publicly available servers.

In each of your systems, either nodes or servers, you will have a daemon called `ntpd` running. Its behavior will be determined by the configuration file `/etc/ntp.conf`. In a typical client such as a compute node or a remote-boot terminal, this file will contain a line such as

```
server fit.free.univ.com
```

or, in the case of a PMC node, you could use

```
server pmcs00.free.univ.com
```

telling the daemon to use the cluster server as a NTP server. In the server itself the file will contain at least two lines such as

```
server clock.isc.org
broadcast 172.16.129.255
```

or, in the case of a PMC server,

```
server clock.isc.org
broadcast 192.168.0.255
```

The first line tells the daemon of the cluster server to use an external machine as a NTP server, and the second line enables the NTP service to all machines in the local network server by the cluster server. Note the use of the broadcast address of the LAN. Fully commented example configuration files are provided with the packages you already installed, including reasonable defaults, so it should not be difficult to set up the NTP servers and clients. There are also examples of these files in the resources area.

There is a second program called `ntpdate` involved in the NTP time synchronization system. It can be used to set the clock of the machine during boot and is useful in case the local clock is really quite far away from proper synchronism. In general the NTP daemon does not deal very well with very large shifts in time, and the use of this extra program corrects this by suddenly bringing the local clock into at least approximate synchronism during the system boot. It should use the same NTP servers used by the `ntpd` daemon, and is configured by means of a file called `ntp-servers` located in the directory `/etc/default/` of each system. There are examples of this file in the resources area. You may also use this program manually during the operation of the system if the need arises, but before you use it you must remember to first shut down the `ntpd` daemon with

```
/etc/init.d/ntp-server stop
```

because they both use the same NTP network port for their connections and trying to run both simultaneously will result in a conflict. After you are done using the `ntpdate` command to correct your clock, just turn the `ntpd` daemon back on for a finer and continuous adjustment of the local hardware clock.

The last essential service we will discuss here is the all-important network information system (NIS). This is a structure that is able to distribute through the network the user and host databases that each system needs in order to be able to function in true multiuser mode. Each system needs to know the usernames, passwords and other data relating to each user which is authorized to use it, as well as the names and addresses of other hosts it must access through the network. Usually this data is contained in local files in the `/etc/` directory of the system, such as the `passwd`, `group` and `hosts`

files, among others. These are all common text files which must be maintained manually by the system administrators, and it is obviously very inconvenient to have to maintain in proper synchronism copies of these files in each machine of a large local network. The NIS system will allow you to keep a single copy of all the necessary files in a single system, the NIS *master server*, and will automatically distribute all the information to all other machines on the local network.

The NIS system also provides the very useful concept of *netgroups*, symbolic names for sets of hosts or of users, which we will use for various functions relating to access control, as will be seen later. A NIS master server is associated to a NIS *domain*, a name which will identify that NIS service and which clients must use when requesting a connection to the service. This domain name could be anything, but it is a good idea *not* to use the Internet domain name of the institution for this purpose, because there may be more than one NIS service in its local network. It could even be the hostname of the master NIS server, or something like it. Just like the nickname of the cluster, it also should be something that clearly identifies the group of users. In order to make things definite we will use here the name "fityp" for the NIS domain name. Since the traffic generated by the NIS services is not encrypted and carries sensitive authorization information, the use of a NIS domain should be restricted to the local network, for security reasons.

This is another network service which works according to a client/server architecture, there are NIS clients and NIS servers. The servers include a single master server and possibly also slave servers, but for the time being we will not talk about slave servers, so you will have just one server, the master server. Typically the cluster server will be the NIS master server, unless you already have some other machine playing this role on your local network. In this case all the PMC compute nodes and remote-boot terminals will be clients of this one server. If you already have a NIS master server installed in some other system in the LAN, then all the machines in the cluster, including the cluster server, may be clients of this other system. However, it is important that this other server be running a recent version of NIS, at least as recent as the one in the sarge distribution of Debian. Versions of NIS on traditional Unix systems and older Linux versions may not be appropriate for use by the cluster, and some older Unix versions may even be incompatible with the version used in Linux. The NIS system running within the cluster should be well integrated with two other components which are important from the point of view of system security: shadow passwords and MD5 passwords. Older versions of NIS may lack one or both of these properties.

In a system with shadow passwords the encrypted passwords usually contained in the file `passwd`, which must be readable by all users for reasons unrelated to the passwords, are transferred to another file called `shadow`, which is readable only by the root user and by the group "shadow". The system commands which must consult the passwords are in the shadow group and able to read the file, while the normal users are not. In this way no users except root have access to the set of encrypted passwords in use

within the system, which makes it much harder for anyone to obtain them and try to crack the encryption in order to obtain the passwords and hence gain unwarranted access to the system. A system with MD5 passwords uses for the passwords a much stronger form of encryption than the traditional "crypt" of the Unix system. With this strong encryption system it becomes much harder to break the encryption and, most importantly, it becomes possible to use longer passwords than the 8-character passwords of the traditional system. While in the traditional system it is usual to require passwords from between 4 and 8 characters, in the MD5 system one may require that the passwords have a minimum of 8 and up to 16 characters.

These two security components are very important to ensure the safety of the system and should not be neglected. The system structure of the sarge distribution supports both components and has them well integrated with NIS and with the library of Pluggable Authorization Modules (PAM) which is the basis of the user certification and authorization subsystem used by Debian. The sarge distribution comes with MD5 passwords by default, and you will recall that in the initial installation of the system we chose to turn on password shadowing. When you install the NIS package it will fit smoothly into this system, it can handle MD5 encryption and it knows about the `shadow` file, which will be distributed to all the NIS clients. With all this in place, your whole cluster will have a completely satisfactory degree of password security.

In the NIS installation and configuration sequence that follows we assume that the cluster server will act as the NIS master server of your cluster. If some other system already has this role, then you should use it and hence you no not have to do the part relating to the NIS server in what follows, but only the part relating to the NIS client. This means that you do *not* have to worry about the files `default/nis`, `ypserv.conf` and `ypserv.securenets`, that there is no need for the initialization procedure and that the directory `/var/yp/` will play no role. On the other hand, you *do* have to check and fix the files `defaultdomain`, `nsswitch.conf` and `yp.conf`. You can start the installation of the NIS server by installing the `nis` package in the cluster server, with the usual command

```
apt-get install nis
```

During the configuration phase of the package installation you will have to type in the NIS domain name you chose. By default the package will assume that your machine is a NIS client of some pre-existing server. At the end of the process it will try to start the NIS client daemon `ypbind`, but this will fail if there is no NIS server for the domain you entered. It may take a few minutes until the package installation tool gives up and backgrounds the daemon, simply wait until the shell prompt returns. Once that happens, stop the daemon executing immediately the command

```
/etc/init.d/nis stop
```

This will avoid annoying NIS timeouts while you configure everything. The configuration of a NIS master server can be a bit tricky and all the necessary tasks should be done in the correct order, to avoid hangs and general confusion. It is a good idea to read the Debian NIS HOWTO which comes with the package, and possibly to print it and have a hardcopy at your side during the process. This compressed text file is at the location shown in the command below, and you can read it with

```
zless /usr/share/doc/nis/nis.debian.howto.gz
```

Although it may not be completely up to date, most of the important information and the order of the operations is available there. The first thing to do after the installation of the package is to check the file `/etc/defaultdomain`, which should have been written during the process of installation of the `nis` package and should already contain the NIS domain name you chose. If it does not, fix it. Next, you should configure the NIS subsystem of the cluster server to work as the master NIS server, editing the file `/etc/default/nis`, where you should change the default entry `NISSERVER=false` to `NISSERVER=master`. Since we are using shadow passwords the file `/etc/ypserv.conf` should not need any changes. However, you should change the file `/etc/ypserv.securenets` to limit access to the NIS server. Only machines in your local network should be able to use the NIS server, so you should change this file to look like this:

```
# JLdL 26Sep04.
#
# securenets    This file defines the access rights to your NIS server
#               for NIS clients (and slave servers - ypxfrd uses this
#               file too). This file contains netmask/network pairs.
#               A clients IP address needs to match with at least one
#               of those.
#
#               One can use the word "host" instead of a netmask of
#               255.255.255.255. Only IP addresses are allowed in this
#               file, not hostnames.
#
# Always allow access for localhost
255.0.0.0       127.0.0.0
#
# This line gives access to the PMC private network.
255.255.255.0   192.168.0.0
```

In this example the netmask/network pair in the last line limits access to the server from within the private network of the PMC. In case the server should also serve the external LAN, another line can be added with the netmask/network pair for that network, such as

```
255.255.255.0   172.16.129.0
```

This completes the basic configuration of the NIS server. However, it must still be *initialized*, in order that it will create the binary database files it will use to distribute the user and host information to the clients. In order to initialize a master server you should run the command

```
/usr/lib/yp/ypinit -m
```

and follow the instruction which will appear on the screen. Note that this executable is not on the path and that therefore you must use the full pathname. When you run the command, it will write to the screen a bunch or error messages due to the fact that the NIS server `ypserv` is not yet up and running. You may ignore those error messages, because despite this somewhat inconsistent behavior, the binary databases should have been correctly created. In principle you need to do this only this once, during the installation of the server, but we will do it again in a little while, after the `ypserv` server is up and running, to make sure that there will be no more error messages. You should now have a look at the structure of the directory `/var/yp/`, where the server puts all the NIS databases. The command `ls -l` within it should show something like this:

```
-rw-r--r--    1 root     root        16781 Apr 18 17:49 Makefile
drwxr-xr-x    2 root     root         4096 Jun 17 08:32 binding
drwxr-xr-x    2 root     root         4096 Jul 22 20:00 fityp
-rw-r--r--    1 root     root          207 Jan  1  2003 nicknames
-rw-r--r--    1 root     root           26 Jun 17  2003 ypservers
```

The directory `binding` contains information about the `ypbind` daemon. The directory with the name of the NIS domain, `fityp` in this example, contains all the binary files with the databases. If this directory is not there or is empty, something went wrong and you should try to initialize the server again. If a directory `(none)` shows up here, it means that, due to some mistake, the NIS server was started without the prior definition of the domain name; you may delete such a directory and its contents. The makefile contains the rules to rebuild the NIS databases. Every time you change some file which gets read in order to build these databases, such as the `passwd`, `group`, or `hosts` files, you should come to this directory and run the command `make`. With the NIS server correctly configured, we must also configure the NIS client on the cluster server, for usually the NIS server will also be a client, bound to itself, as indicated in the file `/etc/default/nis` discussed above. You should add NIS entries to the file `/etc/nsswitch.conf`, which defines the way in which the system will obtain information about users and hosts, and which should look like this:

```
# JLdL 22Jul04.
#
# /etc/nsswitch.conf
```

```
#
# Example configuration of GNU Name Service Switch functionality.
# If you have the ‘glibc-doc’ and ‘info’ packages installed, try:
# ‘info libc "Name Service Switch"’ for information about this file.

passwd:         compat nis
group:          compat nis
shadow:         compat nis

hosts:          files nis dns
networks:       files nis

protocols:      db files nis
services:       db files nis
ethers:         db files nis
rpc:            db files nis

netgroup:       nis
```

Note that we use a somewhat unusual lookup procedure for hosts, looking them up first in the NIS databases and only later in the DNS system. The other file to change in order to configure the NIS client daemon is the file `/etc/yp.conf`. This will determine which server the `ypbind` daemon will try to bind to and prevent it from trying to find its server by doing broadcast on the LAN. The file should look like this:

```
# JLdL 22Jul04.
#
# yp.conf       Configuration file for the ypbind process. You can define
#               NIS servers manually here if they can’t be found by
#               broadcasting on the local net (which is the default).
#
#               See the manual page of ypbind for the syntax of this file.
#
# IMPORTANT:    For the "ypserver", use IP addresses, or make sure that
#               the host is in /etc/hosts. This file is only interpreted
#               once, and if DNS isn’t reachable yet the ypserver cannot
#               be resolved and ypbind won’t ever bind to the server.
#
# ypserver ypserver.network.com
ypserver pmcs00.free.univ.com
```

Note the warning within this file about the fact that the server listed within it should also have an entry in the `hosts` file of the system. Since in our case here that is the cluster server itself, this should not be a problem. This file and a similar one for remote-boot terminals on the external LAN can be found in the resources area. With all this in place we should be ready to start the NIS server and client daemons on the cluster server. However, the NIS structure is a bit idiosyncratic, any error in the sequence of tasks can mess things up and if this happens the commands using the NIS system are

liable to hang or go through long timeouts. Therefore, as a measure of safety, so there will be less chance of loosing your terminal to such a hang, this time we will run the command in the background,

```
/etc/init.d/nis start &
```

Wait until the job ends, a message should appear on the terminal when that happens, or you can check it with the command `jobs`. After that you should check the state of the NIS subsystem of the cluster server: the command `domainname` should return the NIS domain name you chose; the command `ypwhich` should return the NIS server that the client is bound to, in our case here it should return the hostname of the server itself. You should also test the NIS databases which were created, using the command `ypcat`. This command consults the NIS system and displays on the screen the contents of each database. Try, for example, `ypcat passwd`, `ypcat group`, `ypcat shadow` and `ypcat hosts`. If all these commands return the expected outputs, all should be well. As a final verification step, run again the command

```
/usr/lib/yp/ypinit -m
```

This time there should be no error messages, all the binary databases should be simply regenerated. You may try also running the command `make` within the directory `/var/yp/`.

There is one last task to be done on the cluster server before we go on to install NIS on the node. In order to force the use of longer passwords, you should change a line in the file `common-password` in the directory `/etc/pam.d/`. Instead of the default `min=4` and `max=8` options, you should have `min=8` and `max=16`, so that the line looks like this:

```
password required pam_unix.so nullok obscure min=8 max=16 md5
```

This will force the use of passwords with at least 8 characters. The option `md5` at the end of the line determines the type of encryption used for the passwords. This gives you a basic and probably sufficient measure of password security, but there is more that you can impose, if you wish to, to ensure that passwords that are too simplistic are not accepted by the system. Install the package `libpam-doc` and read the PAM documentation, which you will find in the directory `/usr/share/doc/libpam-doc/` and which is available in several formats. But be careful not to be too strict with this and end up making the life of your users too miserable.

Setting up a NIS client is a fairly simple task, once you have the server available. We will do this on the first node we created before, so you should start by using the `chroot` command to go to its root,

```
chroot /pmc/0000
```

and then mounting the necessary filesystems, as before, with

```
mount /tmp
mount /var
mount /usr
```

You can then install the `nis` package, just as you did on the server,

```
apt-get install nis
```

Once more you will have to enter the NIS domain name you chose. Since the `/proc/` filesystem is not mounted on the node, the `ypbind` daemon should not get started, but if by any chance the daemon *does* get started, immediately after the `apt-get` command returns you should stop the daemon with

```
/etc/init.d/nis stop
```

Remember that this daemon would be really running on the server, not on the actual node, and starting another `ypbind` like this might have bad consequences, such as making the whole NIS system hang. With the software installed you can unmount the node filesystems with `umount -a` and then exit the chroot shell. You should then fix on the filesystem of the node some of the same files which you already fixed on the server. These will be in the directory `/pmc/0000/etc/`, of course, and you do not need to change any files having to do with the NIS server, only those which configure the NIS client. You should check the contents of the file `defaultdomain`, and change the file `nsswitch.conf` exactly as you did for the server. The file `yp.conf` should also be exactly like the one in the server. Note that the hostname of the server, which appears in this file, should be correctly listed in the `hosts` file of the node. If you are using the example files in the resources area, this should already be correctly done. The file `common-password` within the subdirectory `pam.d/` should also be changed to become identical with the one in the server. Finally, there is a couple of modification that you did not have to do on the server. The file `passwd` on the node should have the token

```
+::0:0:::
```

as its last line. Similarly, the `group` file of the node should have as its last line the token

```
+::0:
```

These tokens indicate that the NIS databases should be searched for the required information if it is not found in the files. There are files with the appropriate tokens in the subdirectory `pmc/0000/etc/` of the resources area, marked with the termination `NISTOKEN`.

This completes the installation of the NIS system on both the server and the node. We are now ready to go on to get and assemble the hardware of the first node in order to actually boot it. As a finishing touch to this phase you might want to install on the cluster server the second set of packages, as we already did on the node. In order to do this you may use again, this time in the server, the command

```
apt-get install ‘cat common-package-list-2‘
```

Remember to run the command `apt-get clean` after you finish this. As explained before, it is not really necessary to do this now, but since this is such a basic set of packages it does not hurt to do it right away. The basic system software installation is now finished for both the server and the first node. Later on there will be additional things to install in the case of the remote-boot terminals, which will need an X11 server running in their monitors. The installation of additional packages will also be needed as we introduce further structures and services into your cluster.

Chapter 4

Compute Nodes

4.1 Back to Hardware Issues

With the cluster server up and running, the filesystems of the first node ready to operate, and both systems configured for network booting, it is now time to shift attention back to the actual hardware of the compute nodes. So we must repeat here the cycle of choosing hardware components, configuring the hardware, and assembling it in its final form, which we already went through in the case of the server. Since most of the general issues regarding the choice and care of the hardware are the same as before, we will concentrate here on the issues which are particular to compute nodes. Later on this cycle will be repeated once more, for the case of remote-boot terminals. Since the hardware of remote-boot terminals is a superset of the hardware of compute nodes, many things we will discuss here apply to them as well.

There is no need for describing once more the state of the main elements of PC technology here, since this has already been done for the case of the servers in Chapter 3. Both the servers and the nodes will draw their parts from the same world of off-the-shelf components. In this chapter we will have once more the opportunity of describing the use of the workbench, which you may have already used for the network-booting tests described in Section 3.4. What we mean by this term is a kind of hardware-oriented workstation, which you should regard as a permanent fixture of your informatics infrastructure. We will use it here to configure the motherboards and network cards of the nodes, but it is also very useful for diagnosing and fixing the various hardware problems that are bound to appear in your machines from time to time.

4.1.1 Choosing the Right Parts

We must now discuss the hardware for the compute nodes of the PMC, in order to establish the guidelines for the choice of components. The objective of a PMC is to provide the users with as much batch-mode processing power as possible. This can be achieved either by the installation of a larger number of compute nodes or by the installation of nodes which are individually more powerful. Besides the speed of the CPU, the amount of memory available at each node is also an important issue here. There should be enough memory to accommodate the operating system and the largest program or set of time-sharing programs that is expected to be running on a node during its operational lifetime. Another important issue may be the performance of the network, particularly if the parallel programs intended for the machine are strongly dependent on that.

The hardware of a compute node is very minimal and simple, it consists of only five parts, plus the necessary interconnection cables and physical support structures: a motherboard with an appropriate processor and memory, the network card and a standard power supply. As for the physical support, you may or may not want to use a standard computer cabinet to assemble each node in, as will be discussed later. Besides the usual preoccupation with quality, the main issue regarding the choice of the

components is the resulting price/performance ratio. Usually what is wanted is as much performance as possible within a given budget allocation for purchasing and assembling the complete cluster.

Both the measurement of the performance and the calculation of the price can be more complex than may seem at first sight. One may use as the measure of performance the consolidated speed of the nodes in Mflops (millions of floating point operations per second), according to some standardized benchmark, but this will of course ignore all the possible parallelization issues, including the performance of the network. Since in practice there is very little flexibility available in terms of the performance of the network, using the consolidated speed may still be the best you can do. The performance of a cluster may vary wildly, depending on the nature of the scientific or technological problem at issue, and also from program to program among several different programs written to solve the same problem. It must be realized that the use of a cluster of compute nodes to solve a certain problem may require that both the approach to the problem and the programming techniques be chosen and adjusted to take advantage of the potentialities of the available hardware. Since the limitations relating to the performance of the network are currently more severe than those relating to the processing speed of a node, a successful parallel-programming approach will usually have two basic characteristics:

- Scalability to a fairly large number of nodes, anywhere from tens of nodes to hundreds of nodes; if for any reason the program cannot be made to run efficiently in parallel on a large number of nodes, then it is intrinsically limited in speed, given the inevitable limitations on the speed of individual nodes.

- A coarse-grained parallelization strategy, so it becomes less dependent on the performance of the network; this means that the program should be divided into separate parallelizable tasks such that the tasks running on the nodes will execute for a significant amount of time before they need to interchange data messages with any other nodes.

However, parallel processing is not the sole use of a processing cluster. One may also use the cluster for throughput processing, in which a large number of independent jobs runs in the nodes of the cluster. Here too the cluster is the most effective way to provide processing power, in terms of the price/performance ratio. In many cases the user has to run a single program over and over again, for example for many different values of some set of parameters. So long as each run does not depend on the results of previous runs, this type of operation is a kind of trivial way to parallelize the overall computer effort, and clusters can perform this role very well. In other cases there is a large number of users, each one running a small number of jobs, and once again the cluster may be able to satisfy their needs very well.

While it is true that the way in which the machine will be used does not really matter much from the point of view of its construction, installation, management and

maintenance, this wide range of possible utilizations makes it impossible for us to give here precisely fine-tuned recommendations for the details. While you may base your decisions on the ideas discussed here, you should try to adapt and customize the details for your particular case. If you want to keep things simple, a basic criterion of performance, which is fairly relevant in most cases, is the consolidated processing power of the set of nodes measured by a standard benchmark such as the *Linpack* benchmark. One should use the results of the Linpack 100×100 test for this purpose, for it is the best predictor for the speed with which a single node will run the typical program of the average technical or scientific user. By and large, recent processors with frequencies around 2 GHz have been getting quite close to about 1 Gflop per node for at least some practical programs. The results of the Linpack benchmark for many machines and processors are published regularly at the site `http://performance.netlib.org/`. The benchmark results are listed at the URL

`http://performance.netlib.org/performance/html/PDStop.html`

These performance listings also include measurement of the speed of parallel-processing clusters, which may be of interest to you. You will also find there the list of the 500 fastest machines in the world, at the URL

`http://performance.netlib.org/benchmark/top500.html`

Besides being affected by the difficulties in gauging the performance, the calculation of the price/performance ratio are also made somewhat more complex due to the existence of constraints among the various parts of a node, as well as due to the underlying infrastructural issues. The basic scaling method for making a cluster more powerful may be increasing the number of nodes, but more nodes implies not only more processors, but also more motherboards, more memory, and so on, while using faster processors will not incur in the expense of these other parts. The amount of memory in each node is determined by the needs of the intended applications, and since memory cannot be shared among nodes, that given amount of memory must exist in each and every node. The price of a node must include all the necessary parts, of course, and the price of the CPU chip in itself is probably only about half of it, possibly less.

By and large, on a typical node, the processor and the motherboard will contribute approximately the same for the price, while the memory will contribute anywhere from less than half to a full measure of that amount, depending on how much memory is needed. The network card comes next, contributing approximately from a quarter to one half that same amount, depending on whether you use a 100 Mbps or a 1 Gbps card. In the case of the 100 Mbps card, the price of the card should be about the same as the proportional cost of a port on the switch it is connected to, considered as a fraction of the price of the switch itself. For a 1 Gbps card, currently the port on the Gbit switch probably still costs more than the card. Other indirect costs due to the infra-structure,

which should be considered when you increase the number of nodes, are the possible costs of air and power conditioning, as well as of the physical space required, including shelves or cabinets as the case may be. The cost of the power supply and of the cables is much smaller and can be ignored on a first analysis of the problem.

In short, the main contributors to the price are the processor and the motherboard. The memory comes next and depends on the needs of the applications. The last large contribution comes from the network card, which is tied to the switch it is to be connected to. These are the primary contributions to be considered on the initial analysis of this optimization problem. The other contributions may have to be taken into account when you refine that analysis, with appropriate consideration of the infra-structural items relevant for your particular situation. Given all these guidelines, one finds that it is usually advantageous to use on the nodes a processor which is somewhat, but possibly not a whole lot, faster than those used on the servers. The processor probably should be the single most expensive part of a node.

After you consider these guidelines in order to determine approximately the configurations which are most likely to work for you, you should get quotations for several such configurations, within a reasonable range of variations. From these you will be able to estimate the total performance you will get in each case and to calculate in detail the cost and the price/performance ratio of each solution. Proceeding on these lines you should be able to find the optimal solution for your case. Let us now list some basic technical recommendations regarding each one of the main parts of the nodes.

Choice of motherboard: the motherboard should be as clean as possible, without any expensive devices on-board; it should *never* have an on-board video card, because it will not use any video device at all and it is probably not possible to disable this on-board video device; therefore, you should require a motherboard which *does* have an AGP slot; it should not have an on-board network card unless it happens to be a very good device, such as a 3COM or Intel chip; other on-board devices may be inevitable but are not a problem so long as they can be disabled.

The PCI bus speed issue turns out to be trivial for a node. A standard 33 MHz, 32 bit PCI bus, with its nominal throughput of 133 MBps, is quite sufficient for either 100 Mbps or 1 Gbps network cards, since the former has a nominal throughput of 12.5 MBps and the latter a nominal throughput of 125 MBps. Also, one should recall that in a compute node the PCI bus will be in use for the network card and nothing else, since it will be the only active input/output device.

A word should be said about the use of dual-processor motherboards on the nodes. This may be a solution to look at if you are short of physical space or network ports. However, since you will probably have to double the amount of memory in such a node, so that each processor can still use the amount of memory the applications typically need, and since dual motherboards tend to cost approximately twice as much as single-processor ones, it may not be too advantageous in terms of the price/performance ratio.

Another problem is that dual motherboards are usually meant for servers and may have expensive SCSI devices on-board. You must avoid those, otherwise the price of a node will go right through the roof.

Besides, in a dual node the two processors will have to share the same network channel, which makes for a very inhomogeneous connectivity of the complete set of processors. While the two processors on the same motherboard will be very well connected, each one will have, on average, only half the bandwidth of the network connection to communicate with processors on other motherboards. Bonding two network cards in each motherboard will also imply doubling the investment on the switches, which will probably eliminate completely any financial advantages of this solution. Besides, using bonding may significantly complicate remote booting as well, as will be discussed later in this Subsection. Going from a 100 Mbps network to a 1 Gbps network will also be expensive at present, but should become feasible in the near future.

A further problem is that the simplest form of remote booting, as it is described in this book and used as the default for our cluster architecture, will not work on a dual motherboard with 2.4 versions of the Linux kernel, up to 2.4.27. As a consequence of some internal kernel problem, apparently involving the RTC clock, a kernel compiled with support for multi-processor motherboards and booted through the network in the usual way will hang during the boot. In order to solve this problem one may be forced to migrate to the 2.6 kernels, which do not present this problem.

Hence, this kind of solution should only be considered if you are really short of physical space and, unless you are willing to do bonding of network cards or go to a Gbit network, only if the applications are not too dependent on the performance of the network. You should also be willing to face a somewhat more complex technical challenge.

Choice of processor: you should use on the nodes the same class of CPU's that you used on the server, but with a somewhat higher clock frequency. It is usually a good solution to use a CPU which is moderately faster than those used on the servers. If you simply pick the fastest processor available, it is likely that it will be so expensive that the overall price/performance ratio of the cluster will end up being too high.

Choice of memory: the speed of the memory is determined by the CPU chosen; you should use the fastest RAM supported by the CPU; it is necessary that the motherboard you chose support both the CPU and the RAM, of course; you should have at least 256 MB of RAM per node on a generic machine, more if the intended applications require it; each node should have enough memory to hold comfortably the program or set of time-sharing programs meant to run on it, in their entirety; do not consider making large savings on RAM, it is almost certainly not worth it; exaggerate its amount if you can, get as much RAM as possible and reasonable within your budget.

Some comment is necessary here about the issues of paging and swapping. Both

are related to virtual memory, a method to increase the available space in memory beyond the physical RAM of the machine, by using a permanent storage device such as a disk in the role of additional memory. An intensive processor meant to run things fast should *never* do demand-paging of programs running on it, because this is disastrously damaging to the effective performance of the machine. Doing swap is less damaging, but should also not have any use on such a processor, because an intensive job should never be idle for long. The action of swapping idle processes out of memory should always be avoided if possible, and on a compute node this is certainly the case.

Demand-paging is a method which can be used to allow a program which does not fit into physical RAM to still run, although very slowly. The kernel will write idle parts of the RAM image of a running program to a storage device, usually to a swap partition on a disk, and read them back in when they are needed. Since the speed of input/output operations to a disk, even in the case of the fastest disks available, is several orders of magnitude smaller than the speed of input/output operations to RAM, if an intensive process starts paginating with any appreciable frequency the performance of the machine will typically degrade to a crawl, possibly including the performance for any other time-sharing processes which may be running simultaneously on the machine and which are not themselves paginating. In later chapters we will see how to set limits to the memory used by user programs, in order to ensure that only programs which do fit into the physical memory of the machine can run on it.

Swapping is a method which can be used to allow idle programs to reside in virtual memory as others run, when each individual program fits into physical RAM but the set of simultaneous time-sharing program being executed by the machine does not. A program which is idle will be written out to the swap partition completely, where it will wait to become active again. If the program is completely idle and the residence in the swap area is sufficiently long, swapping it out and in again will cause no sustained performance degradation of the machine. This may be useful on servers, possibly also in remote-boot terminals, but seldom on compute nodes. It should be mentioned that it is possible to do swap over the network, it is slow on a 100 Mbps connection, but will not be a problem so long as the network is technically solid, that is, reliable and error-free. Over a Gbit connection I/O the speed should no longer be an issue, since the network is likely to be faster than the hard disk at the far end. This may be of some interest in the case of remote-boot terminals, possibly also for remotely-booted specialized servers.

Choice of network card: it is important that you use a good high-performance network card; currently such a card for a 100 Mbps network can be obtained for about US$ 40, possibly less; cheaper cards, which you can find for as little as US$ 10 to US$ 15, will typically have only a quarter to a third of the actual performance, and are *not* to be recommended; examples of high-performance network cards are the 3COM 3C905CX and the Intel EtherExpress Pro/100 models; do not consider saving on the network cards, it is probably not worth it.

A few comments on the bonding of network cards are in order. Bonding is a method by which two or more network cards existing on a given machine can be joined together by software, so that they can be seen and used as a single network interface by the operating system. In this way one can create faster network interfaces using several standard network cards. This has to be done on both ends of the network connection, of course, and in the case of a cluster that means that it must be supported by the switch as well. If the switch does not support it, one can get around the difficulty using several switches in parallel; if one bonds two network cards on each node, two switches are needed, one to interconnect the first card of each node, the other to interconnect the second set of cards.

Usually one can nearly double the network performance by bonding two network cards in each node. Since one needs twice as many network cards and twice as many ports on the switches, the cost of the network hardware will exactly double. While the cost grows linearly with the number of cards in each node, up to the 4 or 5 one may put in the available PCI slots of the typical motherboard, the performance will not grow in such a linear fashion. With two or three network cards per node one may expect reasonable returns in performance, but above that a law of diminishing returns sets in and the price/performance ratio degrades rapidly.

You should only use bonding if it is the only solution for your problem, for example if you absolutely need a network bandwidth of 2 Gbps and there are no cards above 1 Gbps available in the market. If there are faster cards available, it is typically better to use them instead of bonding slower cards. Bonding is usually more expensive and less performatic if compared to the adoption of a faster network standard. With the current prices, probably it already is better to go to a Gbit network than to bond cards on a 100 Mbit network, if a faster network is really needed.

In addition, using bonded cards on the nodes may make remote booting considerably more complex and difficult, because the boot programs do not support bonding and it will be necessary to use the network connections in non-bonded mode during boot, switching to bonded mode after the kernel boots. Whether or not this is possible on a remote-boot cluster, and with what level of difficulty it can be achieved, is currently unknown to us. The only obvious solution is the use of two independent networks, one without bonding for booting, the other with bonding for actual operation, which makes the whole thing even more complex and expensive.

Choice of power supply: in order to choose the power supply you must know the power consumption of a node; most of the power used by the node is dissipated as heat on the CPU; in recent motherboards some power is dissipated on the chipset too, but in much smaller quantities; nodes based on Pentium CPU's tend to be more economic than those based on Athlon CPU's in terms of their power requirements; a node with a current Athlon XP CPU will require approximately 150 W of power, when in full operation; older Athlon CPU's of the Thunderbird class, the ones with the largest

power requirements and potential overheating problems, may require some 200 W in full operation; idle nodes require less power than nodes in full operation, that is, with 100 % CPU load.

Unless you are using a dual motherboard, a standard 300 W power supply is probably more than what is needed for a node, but they cost very little compared to all other parts, and it does not matter that we do not use their power capacity fully. For a dual node a 300 W power supply is probably just right, but you may want to use a 400 W unit, just in case. The important thing is that using an individual power supply for each node is a simple solution which is very easy to implement. Since there is relatively little power overhead dissipated by the power supply itself, this sub-utilization of the power capacity of each power supply has no relevant impact on the overall power consumption and therefore on the price of the no-break used to feed the complete set of nodes.

If you are going to install the nodes on shelves rather than in computer cabinets, you will not have a reset or power button available for each node, and in this case it is a good idea to obtain power supplies with an on/off switch on their back panels, so that you may easily power-cycle a node that misbehaves for any reason. In all probability you will only have to buy separate power supplies if you are mounting the nodes on shelves rather than standard computer cabinets, since cabinets will often come with a power supply already assembled in them.

Motherboard:	Soyo model SY-KT333 Dragon Lite single-processor, with 3 DDR memory slots, with a 32-bit, 33 MHz PCI bus.
Processor:	Athlon XP 2800+, with ≈2.3 GHz, with appropriate heat sink and cooling fan.
Memory:	DDR with 256 MB in a single DIMM, for a 333 MHz memory bus.
Network card:	3COM model 3C905CX or Intel model EtherExpress Pro/100, 100 Mbps, for a 32-bit, 33 MHz PCI slot.
Power supply:	run-of-the-mill ATX unit with 300 W and an on/off power switch.

Table 4.1: Typical discrimination of a remote-boot compute node.

The remaining parts: the necessary power cables are not a concern, since they should come with the power supplies or with the cabinets you will buy; however, you may have to provide a dedicated set of power outlets in order to connect a large set of nodes; the network patch cords needed to connect the nodes to the switch should be Cat5 or Cat6 as the case may be; you may either buy them or make them using a pair of crimping pliers; the CPU fans should come with the CPU's, you should use only fans which are certified by the CPU manufacturer; you may need to get 5-inch fans (which

have a 130×130 mm square frame) for the shelves or 3-inch (80×80 mm) fans for the cabinets, as the case may be, since not all cabinets come with fans installed; shelves are the least expensive way to install the nodes and constitute no concern in terms of the total cost; standard computer cabinets do cost a little more, but still you should be able to get each one for US$ 25 or so, and therefore they do not have a relevant impact on the total cost either; the issues relating to the physical assembly of the nodes will be discussed in more detail in the last part of this section.

This completes the discussion of the criteria which should guide the choice of the main elements of the hardware for the nodes. In the way of an example you will find in Table 4.1 a typical discrimination of a remote-boot compute node for a general-purpose processing cluster. It should be pointed out that this description is being put together in the beginning of the year 2005, and should be scaled or changed appropriately to fit the state of the technology at later times.

4.1.2 Configuring the Hardware

Next you must configure your hardware, which includes setting up the BIOS of your motherboards in the appropriate way and setting up your network cards for remote booting, possibly including programming it with the binary image of a network boot program. All this is to be done on the workbench, which is just some table space somewhere, with network and power connections available, and with a few spare parts which you will use temporarily in each motherboard you have to configure. These spare parts should include a video card, a monitor, a keyboard and a mouse, so you can configure the BIOS of the motherboards of the nodes, which lack such parts. You may also want to have there a spare motherboard, a floppy drive and a power supply, possibly older ones you may have lying around, which you can use to configure and program network cards.

Let us start with the setting up of the BIOS of the motherboard. If you are going to use standard computer cabinets, you may want to assemble the motherboards on the cabinets before using the workbench, but leave the covers of the cabinets off. Otherwise just assemble each node in turn on the table, over an insulating base, such as a cardboard box or some other non-conducting surface you happen to have handy. Beware of the grey foam which sometimes comes with the motherboards, since it is probably anti-static and therefore *conducting* foam, which should not be used. The same warnings already given and cares already discussed in Section 3.2 apply equally here. In order to do the first node you should assemble the spare video card on its motherboard and connect the monitor, power supply and keyboard to it. If you are not using a cabinet, make sure that the weight of the video cable does not tend to pull the video card out of its socket.

The network card is not necessary at this time, unless you want to take the opportunity and configure it too. If you are going to assemble it now, it is a 3COM model

3C905C or 3C905CX card with a flash EEPROM, and you want to use the Etherboot alternative, then be sure to verify and record the type of flash EEPROM chip on the card before you assemble it on the motherboard. You will need this information later, in order to use the `cromutil` utility to write the Etherboot program to the card. The chip is approximately 11 mm by 14 mm (a little under 1/2-inch by a little over 1/2-inch) in size, and stands about 3 mm (approximately 1/8-inch) high when soldered to the card. Look for the codes written over it, you should find something like "AT49BV512", "SST39VF512" or "SST29EE020", or possibly some variation of these. The first two are chips with 512 kbit which are written byte-by-byte, the other is a 2 Mbit chip which has to be written in a 128-byte page mode; since this last one is faster to erase and program, it is sometimes called a "super-flash" chip. If you find some flash EEPROM chip other than these in your card, you may have to access the sites of the manufacturers in order to determine its size and type. The most relevant sites, both of which allow you to search by part number, are

`http://www.atmel.com/products/`

for the Atmel chips and, for the SST chips,

`http://www.sst.com/products/`

If you are not using a cabinet, since the network card will be just inserted in its socket and otherwise loose over the motherboard, make sure there is no appreciable length of network patch cord hanging from it. If you are using simple shelves to assemble the nodes, this will be a permanent situation, and you should plan on strapping or otherwise fastening the patch cord to the shelf near the network card, in order to avoid stresses on the card. It may also be a good idea to take off the metal plate of the network card, if it is possible to do so non-destructively. Since you will not be able to use it to screw down the card, it only contributes its weight to the stresses on the card. With these cares taken, you will find that, so long as the nodes are locked in their room out of the reach of curious hands, the network cards can simply sit happily in their sockets without any further physical support.

You should now turn on the power supply and press the `[Del]` key in order to enter the setup program of the BIOS of the motherboard, just like you already did in the case of the server. Setting up the BIOS goes mostly as it did before, but some of your choices must now be different. You can start by doing a "load factory defaults", and also a "load failsafe defaults" if it is available and you want to play it safe. If, on the other hand, you want to play around with the optimization of the hardware settings, you may want to try a "load optimum defaults" or some equivalent thing, if it is available. If it turns out that the motherboard starts to misbehave after a few trial runs, you should go back to the safe defaults. The list below contain the essential settings, which must be set as described so your motherboard can act as a remotely booted compute node.

- Set the hardware clock to UTC, the universal time used by all computers on the Internet. It is the same as Greenwich mean time (GMT).
- In the main menu there should be an option determining what the BIOS should do if there are errors during POST. You should set it to "halt on no errors", so the BIOS will not abort the boot due to the missing keyboard, video card and floppy drive.
- Enable and configure the two serial ports, the first one should be at I/O address 3F8 and IRQ 4, the second one at address 2F8 and IRQ 3; these ports are necessary for the serial console.
- Configure a single boot device, the network; sometimes, in older motherboards, the network boot will come in regardless of what you do here, if a boot program exists in the network card.
- Set the state of the power supply after a power failure to ON, so that the node may come back unattended from such an event.
- Configure the power button to turn the power supply off only after being held pressed for 4 or 5 seconds.
- Disable all other items relating to power management, suspend or sleep modes, wake-on-something, and so on.
- Disable all the unwanted on-board devices. These may include sound cards, modems, on-board SCSI devices, the two IDE (ATA) devices, the USB devices, the parallel port, on-board network cards, the PS/2 mouse, etc.

Note that you should *not* set a password for access to the BIOS setup program, because this is neither necessary nor convenient for a PMC compute node. You might want to look also at these additional non-essential items:

- Verify that the first floppy drive is configured correctly and that the on-board floppy device is enabled; you may want to use a floppy drive to boot for testing purposes.
- If you do the above, then reconfigure the boot devices in proper order: first the floppy drive, then the network, and no others.
- Turn off all memory allocations for video and system shadows. Linux will not use this in any case and you save a little bit of memory.
- Disable allocating IRQ's for USB and for the video card, which are not going to be used.

- Disable the ECC checking of cache memory, this will make your machine run a bit faster.
- Check the speed of the fans connected to the motherboard, as well as the temperatures of the motherboard and of the CPU.

Now it is time to set up the network card, so turn off your node and assemble the network card on it, if you have not already done so. Assuming that you are using a modern 100 Mbps card with a flash EEPROM, there are three things you might find necessary to do to it: to use a manufacturer-supplied card configuration tool to activate the boot PROM and/or the PXE capability of the card; to use a manufacturer-supplied flash EEPROM writing tool to install or upgrade the PXE software of the card; use the free flash EEPROM writing tool to write an Etherboot image to the card. The first two are the relevant ones in the case of the PXELinux alternative, the third one is relevant only in the Etherboot alternative, and then only for the 3COM models 3C905C cards.

Older cards and lower-quality cards are typically configured by means of a standalone configuration program, but most newer high-quality cards have a configuration setup program on-board, which usually will allow you to perform at least the first one of these three tasks, possibly more. In this case a message should show up on the screen during POST, telling you what combination of keys should be pressed in order to enter the setup program of the BIOS of the network card. Most cards by Intel use that manufacturer's network boot software and have a setup program that can be entered with the `[Ctrl]-[S]` keystroke combination. The cards by 3COM usually have the MBA (Manageable Boot Agent) network boot software and a setup program that can be entered by means of the `[Ctrl]-[Alt]-[B]` keystroke combination. In both cases this is true not only for 100 Mbps cards, but for Gbit cards as well.

If your network card does not have an on-board setup program that suffices to solve your configuration problems, you will have to use a standalone configuration program. The tools supplied by the manufacturer are typically meant for use in the DOS operating system. Since we do not have this operating system installed in our machines, it will be necessary to make a couple of bootable DOS floppy disks containing each one of the relevant programs. Fortunately it is possible to do this using a free version of the DOS operating system rather than a proprietary one, so that we are able to include an image of such a bootable DOS floppy disks in our resources area. This is a 100 % DOS-compatible open-source operating system written and distributed by the FreeDOS project. You can learn about it browsing the site

`http://www.freedos.org/`

or, since the project is hosted at SourceForge,

`http://freedos.sourceforge.net/`

The documentation relating to the project can be found at

```
http://fd-doc.sourceforge.net/
```

You will find a copy of the binary image of a minimalistic 1.44 MB FreeDOS boot floppy, appropriate for our purposes here, in the subdirectory `usr/local/FreeDOS/` of the resources area, with the filename `fdosboot.dd`. Let us now explain two methods by which you can use it to produce the floppies you need. First of all, you must obtain the necessary programs and, possibly, other files associated to them, so that you have a copy of them available in your server. You can do this either by downloading them from the manufacturer's site, or by copying them from a CDROM which might have come with the card. Usually they will be in a ZIP file which you can open up with the `unzip` utility. In order to make the utility available, install in the server the `unzip` package, using the command

```
apt-get install unzip
```

Then you can try to open up the file with `unzip <file>`. You may try this even if the file you downloaded is a DOS executable with a `.EXE` termination, since the `unzip` utility may be able to find a ZIP image inside the executable file. Next you should obtain a copy of the file `fdosboot.dd` and put it somewhere in your server. You may then create a bootable FreeDOS floppy inserting a blank floppy in the floppy drive and running the command

```
dd if=fdosboot.dd of=/dev/fd0
```

Make sure that you are using a new floppy, or one you know is in good condition. If you run into trouble with this command, you may want to low-level format the floppy before you try again, with the command

```
superformat --superverify /dev/fd0
```

The option `--superverify` will cause the program to do a detailed verification of the media. This is a slow process but you should do it this way nevertheless, since these floppies are prone to presenting media surface errors. Since floppies are such an unreliable medium, you might want to format it in any case, even before you try to use it, as a safety measure. If the `superformat` command fails to complete without error more that once, you should discard the floppy and try another one. The bootable floppy you make in this way contains only two files, the FreeDOS kernel and the FreeCom DOS shell. We must now copy into it the program and files relating to the card. In order to do this you can mount the floppy in the `/mnt/` directory with the command

```
mount -t vfat /dev/fd0 /mnt
```

and then copy all the necessary files into it with

```
cp <files> /mnt
```

Depending on the sizes of the programs and their associated files you may be able to put both the configuration and the upgrading programs in a single floppy, or you may need to make two. Since floppies are not to be trusted, it may be a good idea to keep a backup image of the floppy in the server, in case you need to use the floppy again later. You can make such an image using again the `dd` utility,

```
dd if=/dev/fd0 of=netcard-config.dd
```

using some appropriate name for the file, possibly recording in it the brand and model of the card it is intended to. With the floppy available, you can now attach a floppy drive to the motherboard of the node, which you have in your workbench, insert the floppy into it, and boot the node from the floppy. The FreeDOS operating system will be started. You can just hit `[Enter]` in answer to its request for the current date. You will be in the `A:` drive, as a `DIR` command will show. You will then be able to execute the program you put in the floppy in order to configure or program your network card.

There is another, more elegant way to make the boot floppies you need. It has the advantage that you do not need an actual floppy drive and media in order to produce the images, although you will of course need them in order to make an actual floppy you can use to boot the node. You can do this using the loopback mount available in Linux, which allows you to mount a file containing the image of a filesystem, as if it was an actual block device. In order to do it this way, you start by simply making a copy of the bootable floppy image,

```
cp fdosboot.dd netcard-config.dd
```

You can then mount this image in the directory `/mnt/` using

```
mount -t vfat -o loop netcard-config.dd /mnt
```

Use `df` to see that the mount is in effect. A `ls` in `/mnt/` will now show the contents of the filesystem which is within the file `netcard-config.dd`. You may now copy the relevant files into it just as you did before,

```
cp <files> /mnt
```

You may now unmount the image with `umount /mnt` and you are finished, you have your new filesystem image. You can now make an actual floppy using `dd` to write from the file into an actual floppy inserted in the floppy drive,

```
dd if=netcard-config.dd of=/dev/fd0
```

The same techniques can be used if you need to make any other bootable configuration floppies, for things such as doing a BIOS upgrade of your motherboard, for example. Such BIOS upgrades are usually distributed by the motherboard manufacturers in their sites and may be needed to make an older motherboard recognize new models of CPU or RAM chips, to make them accept larger quantities of RAM, or in order to solve other problems. It is a good idea to store and keep in your server binary images of all boot floppies that you make.

This completes the task of setting up the network card in case you are using the PXELinux alternative. If you had trouble before trying to boot through the network as described in Subsection 3.4.4, you should be now in a position to try once more to boot a test node on the workbench. However, for the Etherboot alternative, in case you are using a 3COM model 3C905C or 3C905CX card, things are slightly more complicated. You will have to boot Linux on the node from a floppy, then use the `cromutil` program in order to write the appropriate Etherboot image to the flash EEPROM of the card. In order to do this you first have to compile the utility found in the resources area and put copies of the executable and of the appropriate Etherboot image in a directory accessible to the node, for example the `/root/` directory of the node, which is the `/pmc/0000/root/` directory of the server. In order to compile the utility, copy into this directory the files `Makefile` and `cromutil.c`, from the subdirectory `usr/local/src/cromutil/` of the resources area, and run `make` to make the executable. Also copy the `3c90x.rom` Etherboot image from the directory `/usr/share/etherboot/` into this directory.

Having prepared the necessary files on the server, we must now boot Linux on the node using a bootable Linux floppy. You can use, possibly after some adjustment, the floppy image `lilo-boot.dd` you will find in the subdirectory `pmc/` of the resources area. Note that this file is actually a symbolic link to another file, which has a name that records the version of the kernel contained within it. You may also use this image to produce, on the server, a new version of the boot floppy, for example with a newer version of the Linux kernel. It is a good idea to have this bootable floppy image available in any case, as an alternative way to boot a node. In order to adjust the image or make a new one, get a copy of that file and write it into a floppy media with

```
dd if=lilo-boot.dd of=/dev/fd0
```

Take the usual precautions with the media, of course. This operation will produce a bootable floppy which you can use to boot Linux on your node, mounting the root via NFSroot, exactly as it would happen in a complete network boot. However, the floppy contains a configuration file listing the address of the node and other network addresses, which may need adjustment before you use it. In order to examine the contents of the floppy you may now mount the floppy in the directory `/mnt/` with

```
mount /dev/fd0 /mnt
```

This is not usually necessary at this stage but, if you want to install a new version of the kernel on the floppy, next you should copy the kernel of the node into the floppy, using

```
cp -p /pmc/tftpboot/vmlinuz-2.4.27 /mnt
```

You may have to delete the old version manually before you do this, because the floppy probably does not have enough room to hold both. Because they contain many drivers for network cards, the standard kernels in the resources area are quite big, close to the maximum size that can fit into a floppy. When you compile your own kernels, you should strip them of all unneeded drivers, and then they will be lot smaller. Now go to the `/mnt/` directory and remake the symbolic link to the kernel, called `vmlinuz`, with

```
ln -sf vmlinuz-2.4.27 vmlinuz
```

Note that the floppy contains a configuration file `lilo.conf` for the `lilo` utility. This file contains two lines of the form `append="<string>"` where the string contains the same boot parameters found in a similar line in the PXELinux configuration files, such as `root=/dev/nfs`, `nfsroot=/pmc/0000,rw,nolock` and `console=ttyS0,38400`. The second one of these two lines is commented out and contains this last parameter, while the other does not. Just as in the case of the PXELinux configuration files, the first of the two lines is meant for the tests of the node on the workbench, while the other one is meant for the normal operation of the node. These lines contains also the boot parameter

```
ip=192.168.0.10:192.168.0.1:192.168.0.1:255.255.255.0:::
```

which in the cases of both PXELinux and Etherboot is generated automatically by the boot loaders. It contains the minimal network information needed for the node to boot, namely the IP address of the node, the IP address of the server, the IP address of the LAN gateway and the netmask of the LAN. The addresses are those used here as examples for a PMC node. If you are using other addresses or want to use this image to create a boot floppy for some other remotely-booted system, you should edit the `lilo.conf` file and adjust the addresses appropriately. All that remains to be done after this is to run `lilo` using this configuration file, which you can do like this, while you are in the `/mnt/` directory:

```
lilo -C lilo.conf
```

This will remake the MBR of the floppy and make it bootable for the new kernel you put in it, and must be done every time you install a new kernel on the floppy or make any changes to the `lilo.conf` file. If, when you run this command, you get a message saying that there is not enough space in the floppy, try deleting manually the file `map`

within the floppy and then try again. You may now get out of the `/mnt/` directory and unmount the floppy with `umount /mnt`. It is a good idea to use `dd` to make a backup copy of the bootable image on the server, using a filename which records the nature of its contents, such as `n0000-lilo-boot-2.4.27.dd` or something like that.

You are now almost ready to boot the node using this floppy, but first make sure that both the server and the node are connected to the switch of the LAN or of the private network, as the case may be. There are also a few verifications and adjustments to make. Make sure your server is running in multi-user mode, with all services available, since you will need the NFS network service. The configuration of the boot line of the kernel should be *without* the parameter `console=ttyS0,38400`, as shown in the examples in Chapter 3, either in the PXELinux configuration files or in the `Makefile` for the Etherboot NBI kernel. If your node filesystems are configured for compute nodes rather than remote-boot terminals, you must temporarily undo the reconfiguration of `etc/inittab` file in the root filesystem of the node, just as was discusses in Subsection 3.4.4. Save the modified file with a name such as `inittab.CURR` and put back in place the original file, which you saved before with the name `inittab.ORIG`.

In order to boot the node, just insert the floppy into the floppy drive, which should be connected to the motherboard of the node, turn the node on and boot the motherboard from the floppy. You should get a LILO menu and prompt just as you did on the server, just hit `[Enter]` to boot immediately. When the kernel boots you should see the usual set of kernel messages, but with a few differences, as the kernel looks for and mounts its root through the network. If the kernel stops with a panic message because it failed to mount its root, it is likely that there is something wrong with the network or with the NFS service of the cluster server. It may also be that the kernel has no support for the network card you are using. Go back, check everything very carefully, fix any problems you find and try again until you get the system boot process to complete successfully.

Once the system stabilizes in multi-user mode and you get a login prompt on the monitor of the node, log into the node as root, using the same password you used for the server; since the node was cloned from the server, the password was copied from it and is the same. Once you get in, you might want to start by having a look around, checking the mounted filesystems, the hardware detected by the kernel and so on, just like you did on the server just after its installation. If you are using the Etherboot alternative and a 3C905C or 3C905CX card, now it is time to write the boot program into the flash EEPROM. You can program the network card using the `cromutil` utility that you copied before into the `/root/` directory of the node. Generally speaking, the way to use this program goes like this:

```
./cromutil <ioaddr> <command> [(>|<) file]
```

Here `ioaddr` is the hexadecimal I/O address of the card in the PCI bus of the machine, in a form such as `0xA123`. The `command` can be one of those explained in what follows, and in some cases you need input/output redirection from/to a file. You can

get the I/O address of the card using the commands `lspci -v`, `cat /proc/pci` or `dmesg | grep -i 3C905C`. Note that this address depends on which PCI slot of the motherboard you inserted the card in, and possibly on the existence or not of other cards on the bus, among other things. Therefore, if you make *any* changes in the hardware, you must check this address again, as it may have changed. You can also get on-line help for the command executing it like this:

```
./cromutil 0x0000 help
```

where in this case the address `0x0000` is just a dummy parameter and could be anything. Besides this `help` command, the internal commands available in this utility are the following:

`id`	get the ID numbers of the card;
`read > file`	read the contents of the ROM into a file;
`read128 > file`	read the contents of the ROM into a file;
`erase`	erase the whole ROM to the 1 state;
`prog < file`	write the contents of a file into the ROM;
`prog128 < file`	write the contents of a file into the ROM.

The `read` and `prog` commands are to be used if the card has a traditional 512 kb (64 kB) flash EEPROM chip, such as the AT49BV512 or the SST39VF512. The `read128` and `prog128` versions are for cards with a 2 Mb (128 kB usable) page-write flash EEPROM chip, such as the SST29EE020.

Be *very careful* when using this program, you can very easily erase the contents of the flash EEPROM chip. The first thing to do after you get the I/O address of the card is to use the utility to read from the card and store in a file the current contents of the flash EEPROM chip. This will be a backup copy of the boot program currently installed on the card, so that you are able to put it back in if that ever becomes necessary. You will do this backup using the `read` or `read128` command, as the case may be, for example with a command line such as

```
./cromutil 0xA123 read > original.rom
```

Warning: note that the address `0xA123` is only an example, do *not* use it when running this program on your node, use the correct I/O address of your card instead. Since this file is a backup, it is a good idea to disable all write access to it, using the command `chmod -w original.rom`. Next you should erase the flash EEPROM with a command such as

```
./cromutil 0xA123 erase
```

This will erase the whole chip to the all-ones state. You can check that this was done dumping again the contents of the chip to a file, with

```
./cromutil 0xA123 read > erased.rom
```

but make *very sure* that you do not accidentally overwrite your backup file when you do this. If you look at the contents of this `erased.rom` file with the `emacs` editor, you should see a continuous string of EOF characters, which are shown by the editor as the character `ÿ`. If you use the editor to look at the `original.rom` file, being *very careful* not to change it in any way, you should be able to see some strings identifying it as a PXE image or some other form of boot loader. You are now ready to write the Etherboot image to the card, with a command line such as

```
./cromutil 0xA123 prog < 3c90x.rom
```

You will see in the screen some progress report output, as the program writes the image to the card. As a final verification that the image was written correctly, you may read it back out with

```
./cromutil 0xA123 read > verify.rom
```

being careful once more about not overwriting any existing files. The file `verify.rom` should not be different from the original file `3c90x.rom` except for the size, since the `verify.rom` may be padded with binary ones at the end. You can edit it with `emacs` and cut out some of these all-ones `ÿ` characters at the end, until it is exactly the same size as the original image, then compare the two files with the `diff` command,

```
diff 3c90x.rom verify.rom
```

If the sizes are identical, then there should be no differences between them at all, so the `diff` command should return no output. This is the final test that certifies that you have successfully programmed your network card.

With this verification done you are finished programming your network card and ready to use the newly installed boot program to really boot your node through the network. But do not yet undo the adjustments you made to the configurations of the server and the node, since at first we will try it still with the video card and monitor installed on the node.

4.1.3 Network-Booting a Node at Long Last

At this point you should finally be ready to boot the node completely through the network, both in the PXELinux alternative and in the Etherboot alternative. If you are using the PXELinux alternative you probably already did that, but in the case of the Etherboot alternative we must still try it for the first time to check that everything is working correctly. After you do this, it is time to return the configurations to their permanent settings, in the case that yours is a cluster of compute nodes. We will do it in two stages, rebooting the node at each stage, in order to take the opportunity to see in practice the effect of the configurations on the behavior of the node. You may now reboot your node using the Etherboot boot loader. Just take the Linux boot floppy off the floppy drive and reboot the system with the command

```
shutdown -r now
```

Once the Etherboot program is executed and starts to search for the DHCP server, after a short delay, the Ethernet address of the card should be clearly visible on the screen. You should write down this hardware address and register it in the `dhcpd.conf` file of the cluster server. Then restart the DHCP server with the command

```
/etc/init.d/dhcp restart
```

Reboot the node once more, and when the Etherboot prompt comes up, just hit `[Enter]` on the keyboard connected to the node in order to boot immediately. If the kernel fails to load then there may be something wrong with the DHCP or the TFTP services on the cluster server, check it all out carefully. In particular, make sure that the Ethernet address of the card is correctly registered in the configuration file of the DHCP server. Once the kernel loads, since you already booted the node before with the boot floppy, there should be no further problems and the node should boot to the end and stabilize in normal multi-user mode, showing a login banner and prompt on the screen. You may want to log into the node and look around its system.

If the node is to be a compute note rather than a remote-boot terminal, we will now reset the configurations of the server and the node to their permanent settings and reboot the node a couple of times. You should do this regardless of whether you are using PXELinux or Etherboot. The first thing to do is to return to its place the file `/etc/inittab` of the node, which you saved before with the name `/etc/inittab.CURR`, once again saving the original version of the file with the name `/etc/inittab.ORIG`. You may do this in your login session on the node itself. Having done it, reboot the node once more and watch its monitor to see what happens. You should see the hardware reset cycle, the prompt and messages of the network boot loader, and the kernel messages as it boots.

However, by the end of the boot the terminal will freeze and there will be no login prompt on the monitor. This is so because, with the current configuration of the

`/etc/inittab` file, the `getty` daemons for the virtual terminals `tty1` to `tty6`, which are associated to the VGA console, are disabled. The only available ways to log into the node now are through the serial ports and through the network. Since we do not yet have a serial console, we must use the network. Therefore, log into the server as root and, from there, use the `ssh n0000` command to log into the node as root through the network. Look around the system of the node once more, except for the lack of login prompts in virtual terminals on the monitor, everything should be as before.

Returning to the server, you should not reset the PXELinux configuration file or the `Makefile` of the Etherboot NBI kernel, as the case may be, to include in the kernel command line the boot parameter `console=ttyS0,38400`. In case you are using PXELinux you should change the configuration file which corresponds to your node within the directory `/tftpboot/pxelinux.cfg/`. In case you are using Etherboot you should change the file `/tftpboot/Makefile` and then run the command `make` within the same directory. Having done this, reboot the node once more, either logging into it through the network and using the command `shutdown -r now` or pressing the keystroke combination `[Ctrl]-[Alt]-[Del]` on the keyboard connected to the *node*. Be careful not to mistakenly do this on the keyboard of the server and thus reboot it unintentionally.

Once more you will see on the monitor of the node the hardware reset cycle and the prompt of the network boot loader. There will be a message saying that the kernel is being loaded, but this time you will see no messages as the kernel boots. Just as before, you will not be able to log in using the VGA console, since it is not active, and again you will have to log into the node through the network, which is the standard operating situation for a compute node. The set of messages which are generated during the boot of the system are now being sent to the first serial port instead of to the VGA console, and and therefore invisible to you right now. Once we put in operation a serial console later on, you will be able to see the messages again, this time within a terminal emulator running on the X11 system, for example in another node acting as a remote-boot terminal. Note that for a compute node it is essential to have a timeout in the boot loader, after which the boot happens automatically, because in such a node there is no keyboard to initiate the boot.

We are finished with the basic setup of the first node, you may log out of it and shut down its system if you wish. We must now continue the work towards assembling and cloning larger numbers of such nodes. You may now use your first node as a workbench machine to set up and program all the network cards of your compute nodes and/or remote-boot terminals, making sure that they work correctly and registering their hardware addresses in the configuration file of the DHCP server. Since it is the network card which determines the address and the hostname that each machine will correspond to, remember to label each network card with the corresponding hostname. If you are using cabinets, do that on the metal plate of the card, so that it is visible from outside. Otherwise, you can stick a label on the card itself. The workbench should

also be used to set up the motherboards of all the compute nodes, of course.

4.1.4 Assembling a Set of Nodes

When you finish setting up the motherboards and network cards of all your nodes, you will be ready to assemble them together into a computing cluster. As we saw above, the hardware of each node is fairly simple, but there is also the problem of how to assemble together a large number of them, which presents its own challenges. A sufficiently large number of nodes may pose non-trivial problems with regard to the necessary infra-structural items, such as power and air conditioning, as well as with regard to the physical space required. The infra-structural items just mentioned will be discussed in more detail in a later chapter, here we will be concerned only with the problem of physical space and with some of the details of the assembly.

There are basically two viable alternatives for the nodes: the use of standard computer cabinets and the use of simple shelves in a structure like a bookcase, over which you can assemble the nodes in a variety of ways. In what follows we will refer to these alternatives as the cabinet solution and the bookcase solution. The cabinet solution is simpler to implement, but it is more expensive and will require about twice as much physical space as the bookcase solution. Specialized rack solutions tend to be too expensive to be worth considering for most small to mid-size clusters. The use of cabinets may introduce some concerns regarding good ventilation for the CPU inside, which precludes the use of the smaller sizes. On the other hand, you will be able to use the standard power and reset buttons of the cabinet, while with simple shelves you will have to rely on power-cycling the nodes by turning off the power supply, in case you have any trouble with them. Also, in the cabinet solution you will have the power LED and the beeper as possible diagnostic tools, which may be absent in the bookcase solution, except if the motherboard has an on-board beeper or power LED, as many modern motherboards do.

Even if you use cabinets, you will still need shelves, in order to stack them vertically. But in this case you will need only a few widely spaced shelves. For the bookcase solution you will need many narrow-spaced shelves, about three times as many as for the cabinet solution. In order to establish an approximate but concrete notion of the sizes involved, let us mention that a typical tower-shaped cabinet of the appropriate type will have approximate dimensions for width, height and depth given, in this order, by 20 cm $\times$ 43 cm $\times$ 50 cm, which in inches translates to $8 \times 17 \times 20$. In comparison to this, the shelf space needed for a node mounted directly over the shelf is given approximately by 13 cm $\times$ 33 cm $\times$ 50 cm, or $5 \times 13 \times 20$ inches. From this one can calculate that the cabinet solution requires about twice as much volume as the bookcase solution. If one uses the same maximum height for the sets of shelves in either case, this will translate to about twice the floor space in the room in which you intend to install the nodes.

The shelves should be about 50 cm (20-inch) deep in either case, although you may manage to get by with 40 cm (16-inch) shelves. In the case of the cabinet solution one should consider stacking up at most 5, but typically 4 shelves of cabinets, so that

they are within easy reach. In the case of the bookcase solution it should be at most 15 shelves high, but typically one should have 12 shelves of nodes. This means that typically one should have 5 shelves in the cabinet solution, 4 for cabinets plus a covering top shelf, and 13 shelves in the bookcase solution, 12 for nodes and the covering top shelf. In either case the top shelf will be about as high as a standard door. If we take as an example a set of 100 cm-wide (3-foot+4-inch) shelves, with a 100 cm × 50 cm (3-foot+4-inch × 1-foot+8-inch) footprint, we will verify that we can install 20 cabinets on 4 shelves, possibly a bit tightly, and that we can install comfortably 36 nodes on 12 shelves. Remember that you must fit a round number of either cabinets or motherboards within the width of the shelves. For the cabinet solution shelf widths of 80 cm, 100 cm and 120 cm, which are common standards, will usually do fine. For the bookcase solution you will be able to use shelf widths of 70 cm, 100 cm and 140 cm. Of course, these are only approximate examples and you should consider in detail the dimensions of all the parts you purchase, so that they match correctly.

In any case, you must also remember that you need walking space both in front and behind the shelves. In the case of the cabinet solution you need access to the power and reset buttons in the front, and all power and network cables will be on the back. In the case of the bookcase solution the power cables and power supplies will be on one side, the network cards and network cables on the other. You also need room on both sides of the shelves in order to allow for the air flow needed for ventilation, for cooling purposes. This means that neither the front nor the back of the shelves can be against a wall. But the shelves do not need to be a in completely free-standing structure, you may screw one of its sides to a wall or some other firm structure, as a way of stabilizing and strengthening the structure of the shelves. Let us now go through the basic characteristic which are required of the computer cabinets and of the shelves.

Choice of cabinet: in order to allow for good ventilation and ease of assembly and maintenance you should *not* use the small so-called mini-tower models, in which the power supply partially covers the motherboard; use only the somewhat larger so-called *midi-tower models*, in which the power supply leaves completely free all the space over the motherboard; the cabinet should have two independently removable side covers, not a single overall cover for both sides; it should have space for one 3-inch fan on the back panel, in the immediacy of the CPU chip; it should be of reasonably good quality, fairly robust and with no internal or external sharp metal edges.

The assembly of the motherboard in the cabinet goes as usual, you may proceed just as you did in the case of the server. You may have to buy separately the 3-inch, 12 V fan to install on the back panel, plus the appropriate screws. The back fan should drive the air to the exterior of the box, just like the power supply fan which is close to it. Most cabinets also have space for another 3-inch fan in the front panel, which you may also want to use; this fan should drive the air to the interior of the box. Alternatively,

in order to improve ventilation you might want to leave off the plastic and/or metallic covers of the 5.25-inch bays on the front; you might even consider leaving off the whole front plastic faceplate, unless this interferes with the power and reset buttons or with the power LED.

Choice of shelves: you may use either metal or wooden shelves; metal shelves are usually the cheapest solution; metal shelves may be made of painted sheet metal, but wooden shelves should have their upper surfaces finished with a hard plastic laminate such as Formica, since paint and wooden finishes probably will not stand the action of the double-sided adhesive tape you will probably use on them, if you ever need to take it off; the height of the shelves should be adjustable at 5 cm (2-inch) intervals; the structure should be sturdy and of at least reasonably good quality in terms of materials and workmanship.

Many shelf structures include the need of cross-braces in order to be sufficiently robust and stable in a free-standing situation. These cross-braces are not a problem on the sides, but may be difficult to install in the front or back without interfering with the cables or with the access to the cabinets or motherboards. The alternative is to leave them off and screw one side of the structure to a wall, as was mentioned above. If you have to use more than one shelf structure, you can screw each to the next one in order to make a longer structure, and still screw only one side of the whole thing to the wall.

In the case of the bookcase solution you will need to get a set of 5-inch, 12 V fans for ventilation purposes. Both the power supplies and these fans can be simply glued to the shelves by means of double-sided adhesive tape, of the type made with a thick foamy material. Once in place it can stand considerable stresses, and in order to take it off you can just pass a sharp cutter through it. The fan should be mounted beside the power supply, glues to it and to the shelf below. Use one of the connectors of the power supply to connect the fan. The extra fan should drive the air in the same direction as the fan of the power supply.

Mounting the motherboards to the shelves is an entirely different problem. If you are using non-conducting wooden shelves you might use the double-sided adhesive tape here too, gluing them directly to the shelves, but you incur on the risk of damaging the motherboards with the cutter when you have to take it off for some maintenance operation. Hence for the case of motherboards better solutions are needed. One solution which can be used is to screw the motherboards to the shelves by means of a set of 1/8-inch self-threading screws with spacers, in order to maintain a minimal distance between the motherboard and the shelf. Appropriate cylindrical spacers can be made out of 1/2-inch segments of 1/4-inch-diameter aluminum tubing, which you can easily cut off with a hacksaw. You may use flat-headed screws which are about 3/4-inch or 1-inch long, for metal and wooden shelves respectively.

Another solution is to cut 1/2-inch-long segments out of a 1/2-inch-diameter plastic

rod, made of a plastic such as PVC. You can bore small holes through their axis, to receive short (1/4-inch to 3/8-inch long) self-threading screws you can use to screw down the motherboards. On the other side you can use other screws on metal shelves or, for either type of shelf, you can simply glue the base of the plastic cylinders to the shelves with double-sided adhesive tape. If you are able to find them for sale, you might also use standard motherboard supports made of nylon or some other plastic. You might get some of those with the motherboards you buy, but probably not enough. You can trim the foot of these supports with a cutter in order to obtain a flat surface, taking off a small knob meant to be inserted into slots on the motherboard plate of the cabinet. Then you can glue them to the shelves with double-sided adhesive tape. The other side has a catch that you will insert in the screw holes of the motherboards.

If you are boring holes on the shelves, first make a thin plywood template with the position of the holes of the motherboard, and use that as a guide to bore the holes in the shelves in their appropriate positions. In this way you may avoid destructive interactions between your power drill and the motherboards. You will have to bore the holes before you assemble the shelves, otherwise you will not have enough space for your power drill. If you are using plastic cylinders glued do the shelves, first screw them to the motherboard, then apply the adhesive tape and just set the motherboard carefully in its proper position over the shelf. Proceed in a similar way in the case of the standard nylon supports. In either case you will need a short screwdriver in order to take the motherboards off the shelves and screw them back in again, given the limited vertical space available over the shelves.

The top shelf is an appropriate place to put the switch of the private network of the PMC. You can glue it there with the double-sided adhesive tape. You will have to fasten the network cables in bundles to the structure of the shelves, using nylon straps, as they run from each node to the switch. If your computing cluster is large and the distances the cables have to cover are considerable, or if you want to make each patch cord exactly the size it needs to be, you may be better off making your own patch cords using a pair of special crimping pliers, crimpable RJ45 connectors and rigid Cat5 or Cat6 cable, rather than buying standard flexible-cable patch cords of some standard length. The rigid cable is both cheaper and a better conductor of the network signals and, if it is going to be strapped in place, it does not really need to be flexible.

You will also have to fasten to the structure of the shelves a set of bars of power outlets such as line filters, in order to be able to power up all the nodes and the switch. You may screw them on or glue them with double-sided adhesive tape. Nylon straps may be a solution here too, both for the outlets and for fastening the power cables in place. It is a good idea to label each power supply with the hostname of the node it serves, so that you may know exactly what you are doing when you have to power-cycle a certain node in the middle of them all. As you can see, assembling a set on nodes using the bookcase solution is a nice proposition for a do-it-yourselfer. You may just use your imagination to create a practical and neat assembly for your powerful computing

cluster.

In short, if you can afford the extra cost and the physical space, and you want to keep your life as simple as possible, go for the cabinet solution. If, on the other hand, you cannot afford either the cost or the space and have no fear to handle a small power drill and a few tools, then the bookcase solution may be right for you.

4.2 Cloning Additional Nodes

With the cluster server up and running and the hardware of all the compute nodes ready to start operating, we must now clone the filesystems of our first node in order to provide the system software for all the other nodes. We have available a set of scripts ready to do that but, since it is important that you understand in detail how the whole scheme works, we must first go through a detailed explanation of the ideas and techniques involved. In order to do that we must review some of the characteristics of the structure of the filesystem under Linux. Let us start, however, by stating clearly the problem which is to be solved.

We already have a set of filesystems for the first node, and we know that the node can mount and use them through the network. It is clear, therefore, that in principle all we must do in order to have more nodes operating is to repeat the recipe for each node, changing only the name and address of each node. This will stumble, however, on the problem of the amount of disk space needed, as a quick calculation will show. A typical compute node will have disk occupation in its root, `/var/` and `/usr/` filesystems of 20 MB, 40–80 MB and 1 GB respectively. Considering that the filesystems should be no more than about 2/3 full, in order to provide space for content fluctuations, and that the `/tmp/` filesystem should be about the same size as the root filesystem, we arrive at minimum filesystem sizes of 32 MB, 32 MB, 128 MB and 1.5 GB for the root, `/tmp/`, `/var/` and `/usr/` filesystems, respectively. If we consider a computing cluster of 48 nodes, we conclude that we must have total disk space in the amount of about 80 GB available on the server. For remote-boot terminals the situation gets much worse, since the typical occupations are, in the same order as above, 40 MB, 200–300 MB and 3.5 GB, implying minimum filesystem sizes of 64 MB, 64 MB, 512 MB and 5.5 GB. If we consider a set of 24 terminals, this implies the need for a total of about 150 GB of disk space on the server. For larger numbers of compute nodes or remote-boot terminals the disk space requirements grow linearly with the number of machines and may quickly become impossibly large, of the order of hundreds of GB.

Not only this is a lot of expensive disk space, but its content would be highly repetitive in nature. Since all the 48 compute nodes will have almost identical disk content in their filesystems, most files will be repeated 48 times in the disk content of the server. This is a huge and unnecessary amount of redundancy. The same is true for the remote-boot terminals, even if they may be somewhat more heterogeneous in terms of their hardware, so long as we have the same set of software packages installed in each one of them. If we analyze the situation filesystem by filesystem, we quickly see that the filesystems with node-specific content are the root, `/tmp/` and `/var/` filesystems. Since the `/tmp/` filesystem is completely volatile and has no permanent content, it is of no concern to us in this discussion. The `/usr/` filesystem can be exactly the same for all compute nodes or remote-boot terminals, and the root filesystem has only from a few to a few tens of files which are specific to each system. This leaves the `/var/`

filesystem as the only one with a significant amount of node-specific content, although most of it may still be shared.

The problem is, therefore, how to eliminate this huge amount of redundancy among the nodes in order to save disk space, in such a way that each node still has a complete filesystem structure available for its operation. Furthermore, we would like to do this in such a way that the usual Debian package manipulation tools can still be used on a node in the usual way. Since the `/usr/` filesystem can be completely shared, that is, is exactly the same for all nodes and is used by them in read-only mode during normal operation of the system, in this case there is an obvious solution: it suffices to export the `/usr/` filesystem to the set of nodes and mount it in each node so that its contents become available to each node. In fact, it is quite possible to do this and clusters have operated in this way for long periods of time with few problems. We will, however, adopt a different solution as our default here, for reasons which will become clear later.

A similar shared-filesystem solution can be used in the case of the `/tmp/` filesystem, which has no shared content at all. In this case one can have a single filesystem on the server which is exported to all the nodes, although a different directory of this filesystem must be mounted and used by each node, in order to avoid interferences among them. For the root and `/var/` filesystems, which have content that can be partly but not completely shared among the nodes, a special structure must be employed, to allow for the existence of some files which are not to be shared among the nodes. In fact, we will employ this structure in the case of the `/usr/` filesystem too, because it a more general and robust solution, as we will discuss later on. In these cases there will also be a single filesystem on the server, which is exported to all the nodes, and within which there are subdirectories for the use of each node, but this time there will be some special relations among these subdirectory trees.

This is where the details of the structure of the Linux filesystem come in. This is one of the most important structures that Linux inherited from Unix. The concept of the filesystem in Unix or Linux is a very simple yet very powerful structure, as will be shown by its utilization to solve this problem within the architecture of our cluster. It must be noted that the name "filesystem" is used with two related but different meanings in these systems. One meaning refers to the global, abstract filesystem, which is the set of all files in the system organized in a hierarchical way. The other meaning refers to the binary organization of the data written in a physical medium such as a partition of a hard disk, which allows it to store files and be part of the general structure. These two aspects of the filesystem are connected via the concept of a *mount*, by means of which the root directory of a physical filesystem is associated to some subdirectory of the global filesystem, so that the content of the physical filesystem becomes a branch in the oriented tree of the global filesystem.

One of the most elegant characteristics of the filesystem is that usually the end user does not have to interact at all with the physical aspects of the filesystem, he may think only in terms of the global filesystem. Here we will, however, have to discuss

in more detail than usual the structure of a physical filesystem under Linux, such as the second extended or ext2 filesystem. There are seven different types of files which one can have in an ext2 filesystem. These types of files are different from the point of view of their role in the structure of the physical filesystem, not due to the nature of their contents. One has directories, which are files containing information pertinent to the oriented tree which organizes all the files in a hierarchical fashion. Then there are character and block device nodes, which are handles for the device drivers in the kernel. Two others, the names pipes or FIFO's (first-in, first-out) and the sockets, are handles relating to communication between processes within a system and through the network, respectively. There are also symbolic links, which are files containing the name of other files they point to. You will find examples of these six special types of files in the `/dev/` directory of your server. In order to have a look at these examples you can use the command `find`, which will be very useful in the cloning scripts to be discussed in the next section of this chapter. You can use it like this:

```
find /dev -type <type> -exec ls --color -ld \{\} \;
```

where `<type>` is a single letter indicating the type of file to search for. It should be `d` for directories, `c` for character devices, `b` for block devices, `p` for FIFO's, `s` for sockets and `l` for soft links.

None of these six types of files has any appreciable amount of content, and hence none of them need concern us in this first analysis of the problem of disk space occupation, except perhaps for the space allocated to the directory structure on very large filesystems formatted with a large block size. In fact, almost all the disk occupation is located in the common files, the last of the seven possible types of files. Although they are usually named text files, these files can contain anything, from ASCII text to binary images of executable programs. These are the files which we must deal with in order to limit disk occupation. We must now discuss in a little further detail the structure of these common files in the filesystem. The files must be distinguished from the names which may be given to them. In terms of their content the files are identified within the filesystem by a number, a simple integer, which is named the *inode* of the file. A name given to the file is a character string kept in a table which associates it to the inode. The short name of the file is called its *basename* and is associated to a certain directory in the hierarchical directory structure of the filesystem. By following the path backward from each subdirectory to its parent directory until one reaches the root of the global filesystem one reconstructs the complete name or *path* to the file. An example is in order here. Consider the NIS configuration file of our first compute node. The name (basename) of the file is `yp.conf`, and its complete path within the filesystem of the server is `/pmc/0000/etc/yp.conf`. If you look at it with the command

```
ls -li /pmc/0000/etc/yp.conf
```

you will get an output like this:

```
3392327 -rw-r--r--    1 root     root          574 Jul 22 15:09 /pmc/0000/etc/yp.conf
```

The number at the beginning of the line is the inode number of the file. A central concept in this structure is that a single file can have many names or *links*, which are usually called *hard links* to distinguish then from the "soft" symbolic links. This means that a single inode can have several names associated to it in the table of names. These may be more than just different basenames, all located in the same directory, they may be also names located at several different directories of the physical filesystem, thus corresponding to several full paths pointing to the same file. The one limitation is that all the links must be located within the same physical filesystem and, therefore, the part of the path which lies outside the physical filesystem must be the same for all the links. In the example above this means that the directory `/pmc/` must be the first part of all paths pointing to the file. Let us elaborate on our previous example, creating in the server the directory `/pmc/1000/etc/`, which you can do all at once with the command

```
mkdir -p /pmc/1000/etc
```

Then you can create a second link to the file in our example using the command `ln`. If used without the option `-s` this command will create a hard rather than a soft link, so you can use it like this:

```
ln /pmc/0000/etc/yp.conf /pmc/1000/etc/yp.conf
```

Note that the *first* argument is the *existing* link and the *second* is the *new* link to be created, not the other way around. You may now look at the two links with

```
ls -li /pmc/0000/etc/yp.conf /pmc/1000/etc/yp.conf
```

which should return an output such as

```
3392327 -rw-r--r--    2 root     root          574 Jul 22 15:09 /pmc/0000/etc/yp.conf
3392327 -rw-r--r--    2 root     root          574 Jul 22 15:09 /pmc/1000/etc/yp.conf
```

The second number from the left to the right on the example above is the number of links to the file, note that now it is 2, while it was 1 before we created the second link. Note also that the access mask, ownership, date and size of the file relate to the inode, not to the links, and therefore they are reported as the same in the two lines of the output above. Here we have an example of a file with two links, both with the same basename, but in different directories, which is what we need in order to solve our disk occupation problem.

The solution to the disk occupation problem will be giving many different names to the same file, each name appropriate for use by a definite node. Note that in the example

above, since the directory `/pmc/` is exported to all the nodes via NFS, we could have the directory `/pmc/0000/` mounted by one node and the directory `/pmc/1000/` mounted by another. Each node would have, therefore, access to all the files or, more precisely, all the links, which reside inside the tree starting at the directory it mounted through the network. In this way each node would have access to a given link to one and the same file. Note that there might be other files in each directory which are not hard-linked to each other and which may be, therefore, different for each node. Therefore, shared files and node-specific files can coexist quite peacefully in the filesystems of a node. If a node deletes a shared file, it in fact deletes only its link to the file, so that this action will not affect the other nodes. If it creates a new file, a new inode will be created and the file will not be shared, so this also does not affect any other nodes. The only way in which a node may affect another node is by *overwriting* a shared file, but this very rarely happens during normal operation of the system.

We may, in fact, use this overwriting mechanism, either on the server or on a node, as a means of doing simultaneous system maintenance of all the nodes. Suppose, for example, that we need to make a change to the file `yp.conf` of the example above. Instead of editing the file directly, which could unlink (which is the same thing as to delete) the old version of the file, so that the new file would be specific to the node we are working on, we could copy it to a new temporary file `yp.conf.N`, edit this file and then use its contents to overwrite the old file, with a command such as

```
cat yp.conf.N >! yp.conf
```

This will change the contents of the old file without changing its inode, so that the changes will be seen on all nodes that have links to that inode. We may then delete the temporary file, and the change in the contents of the file will have been done without changing the link status of any files in any of the nodes.

We see here that, in order to do system administration of the nodes, we will have to keep in mind always the difference between links and inodes, quite unlike the normal mind-set of a regular user of the system. We must consider how each of the file-manipulation command deals with the inodes and links, keeping in mind that changing a link affects only the node to which it belongs, while changing the contents of inodes affects all nodes. Most of the usual commands have the reasonable and expected behavior regarding inodes and links, so most of the time it is not difficult to keep track of what is going on. In some cases you must determine, possibly by experimentation, how a tool or utility behaves. For example, the `emacs` editor will create a new version of the file it changes, changing the name of the link that points to the old version, which is kept as a backup, while the otherwise `emacs`-compatible editor `jed` will do the exact opposite of this, it will actually overwrite the current file with the new content and create a new file with a backup copy of the original content.

Using the scheme described here we manage to have what is essentially a single image of the node filesystems on the server, while each node has its own set of links to the

files in this image, contained within the directories it mounts from the server. From within the node this set of links looks like a complete and individual filesystem image, which can be managed within the node with the usual Debian package administration tools, plus a few scripts to do a bit of occasional maintenance on these special filesystems, linking together identical files which the package administration tools may have installed separately within each node. This works without any trouble because the Debian tools do the right thing when they have to exchange a file for a new one, they unlink or move to a new name the old link and create a new file with the same name and the new content.

There is no limit to the number of links you can make to the same file, except for the availability of space in the tables holding their names within the structure of the physical filesystem. Since in our special filesystem each file will typically have many more names than is usual, we must have larger tables than usual. Hence the use of the options `-i 1024` and `-i 2048` which, by increasing the maximum number of inodes in the filesystem, indirectly increase the size of these tables. The disk occupation is not exactly constant as one increases the number of nodes, because there is always some overhead for the creation of a new set of links. The most important of these is the disk space allocation needed for creating the new directory structure, specially on very large filesystems, which are formatted with a large block size, typically 4 kB blocks rather than 1 kB blocks. Since each directory must be allocated at least one block, this may cause a considerable amount of disk space allocation for a large directory structure. On a very full remote-boot terminal, with close to 2000 installed packages, this may come to about 50 MB per node in the `/usr/` filesystem. Since the `/var/` filesystem contains a non-shared part, it too will have a considerable overhead per node, in this case because of the actual disk content. These considerations are reflected in the formulas given before, in Section 3.1, for the calculation of the disk space required for each filesystem.

You may now delete the directory `/pmc/1000/` and all its contents, going to the directory `/pmc/` and doing a recursive removal with `rm -r 1000/`. If you are using our standard shell configuration for the root user, this command should ask you for confirmation before each one of its actions. This will include the removal of the file `/pmc/1000/etc/yp.conf` you created there, which is just another link to the file pointed to by the link `/pmc/0000/etc/yp.conf`. After you finish this recursive removal, go check that this other link to the file is still there. A file is actually deleted, in the sense that its content is lost and its disk space allocation freed for other uses, only when the last link pointing to its inode is unlinked. So you see that, in a typical Linux or Unix filesystem such as the ext2 filesystem, which includes the concepts of inode and hard link, one does not really delete files, unless they happen to have a single hard link pointing to their inode; one deletes only hard links.

4.2.1 Cloning Scripts

We will now examine the cloning scripts that are available in the resources area, which you will need in order to clone the filesystems for all your compute nodes. There are separate scripts for each one of the four filesystems of a node, and they should reside in the root of each filesystem in the server. Hence, you will find four scripts named `make-new-netboot-node` in the resources area, in the subdirectories `pmc/`, `pmc/tmp/`, `pmc/var/` and `pmc/usr/`. They are all quite similar to one another, but there are differences since the structure of each filesystem has its own peculiarities. We will examine in more detail the script of the root directory and comment on the main differences found in the other ones.

These are `tcsh` scripts which use crucially the `find` program in order to ensure good performance levels. The shell programming used has some limitations, which are already dealt with in the scripts. The performance of these scripts could be increased with improvements in the programming and possibly with re-programming in C, but the use of shell scripts is convenient since it is simple and easily accessible for modifications by the user. Anyone wishing to modify and redistribute these scripts is welcome to do so, since they are all distributed under the GPL. In fact, you will certainly have to modify them at least a little bit, in order to customize them for your particular situation.

What each one of these scripts does is to create a new subdirectory in the corresponding filesystem, with the appropriate contents for the use of a new node. It does so by copying the structure of an existing subdirectory, called the *source* directory. The new subdirectory which is created is called the *target* directory. The scripts take two arguments, for the source and target directories, in this order. By default the source directory is the `0000` directory and the target directory if the `9999` directory. If no arguments are given in the command line, the scripts will ask for them and, if empty answers are given, use the defaults. In order to use one of these script one first goes to the root of the corresponding filesystem and then executes something like

```
./make-new-netboot-node 0000 0001
```

This example creates the node for the host n0001 out of the node for the host n0000. If one does this in the root filesystem `/pmc/`, then the script will also descend into the subdirectories `tmp/`, `var/` and `usr/` and execute the scripts of those filesystems as well, using the same arguments. Therefore, in order to create the complete set of filesystems for a new node, it suffices to run the script in the `/pmc/` directory.

Let us now go through the program contained in the script for the root filesystem, to see how it works. It is recommended that you make a copy of the complete program found in the resources area and that you edit it with the `emacs` editor, using our standard configurations for the root user, so that you will have available a fully colorized image of the program, which you can peruse while you read this. The first lines of the code define a few variables used in the rest of the code, such as `nick` for the nickname (first

part of the hostname) of the nodes, `ntip` for the number to be added to the number of the node (last part of the hostname) in order to generate the last part of its IP address, and `snet` for the subnet within which the nodes are to be configured. Note that in order to use this script you must put all your nodes is a contiguous range of addresses within their LAN. Here the default source and target nodes are also defined. Then there are blocks of code which ask for and acquire the source and target nodes, in case these were not given as command line arguments. The existence of the corresponding directories is also verified: the source directory must already exist, the target directory must not exist yet. The variables `source` and `target` contain the source and target directories, respectively. With these checks performed, the first action is taken and the target directory is created. Next the source directory is entered and three blocks of code are run within it. The first of these is

```
find . -type d  -exec mkdir -p ../$target/\{\} \; \
                -exec chmod --reference=\{\} ../$target/\{\} \; \
                -exec chown --reference=\{\} ../$target/\{\} \; \
                -exec touch --reference=\{\} ../$target/\{\} \;
```

This block of code uses the `find` command to find all directories (this is what the option `-type d` determines) within the tree starting at the source directory. It then runs, for each directory found, four commands, in order to create the corresponding subdirectory within the target directory and to fix its ownership, mode and date to be the same as those of the directory found. Note that the structure `\{\}` is exchanged by each object found by the `find` command. In order to see how this works just play around with the `find` command without the `-exec` options, so it will find things but take no further action. Reading the manual page of the command may help as well. One sees that this block of code reproduces completely and exactly the directory tree structure of the source node. The next block of code moves the task along by copying all non-text files,

```
find . -not -type d -and -not -type f -exec cp -a \{\} ../$target/\{\} \;
```

By finding all files which are not a directory and not a common file (which is the meaning on `-type f`), this command will find all files of all other types existing within the source node. It will them copy them within the target node. The option `-a` of the `cp` command makes it preserve all the properties of the object being copied. With this done we are left with the common files to deal with. The next block of code does this,

```
find . -type f -exec ln \{\} ../$target/\{\} \;
```

It finds all the common files within the source node and creates new links to them within the target node. Since no new files are being created here, we do not have to worry about ownership and permissions. This completes the construction of the new node. The script returns now to the parent directory and the next block of code checks the copy with the `diff` command, in order to detect whether something failed to get copied or linked,

```
diff --brief -r $source $target |& \
    grep "^Only in $source" | \
    tee `basename $0`.err
```

The result is copied into a file with the name of the script plus a termination `.err`. The construction `$0` returns the full path with which the program being executed was called, and the command `basename` extracts from it the basename of the file containing the program. This `diff` may take a little while to run in the case of the `usr/` filesystem, if it is very large. Next the script changes the link with the name of the host, pointing to its root, which we traditionally have in all systems, then edits automatically the crucial files which define the new host. This is done with the `sed` command, a stream editor meant for this kind of use. The files changed are all within the `etc/` directory of the target node, they are the files `hostname`, `fstab`, `interfaces` and `exim.conf`. Here is the code for one of them as an example,

```
rm -f $target/etc/hostname
cat $source/etc/hostname | sed -e "s|$source|$target|g" > $target/etc/hostname
```

Here the old file is removed in the target node, then the contents of the file of the source node are passed through the editor, which exchanges the old numerical part of the hostname by the new one and creates a new file in the target node. All the other files are treated in a similar way. The first three define the hostname, system mounts and IP address of the new system, the last one defines it as a client of the mail system of the cluster. If you do not yet have this part of your system structure set up, you may just comment out the corresponding part of the script. Note that in the case of the `interfaces` file there is a bit of integer shell arithmetic involved, in order to define the IP address,

```
@ stemp = $source + $ntip
@ ttemp = $target + $ntip
```

and that the nickname variable `nick` is used only in the editing of the `exim.conf` file and in the modification of the soft link to the root, both of which may not be necessary in your case. The last part of the script of the root filesystem simply executes the similar scripts in the other three filesystems, entering each subdirectory to run each script and coming back up to the root directory. The script of the `tmp/` filesystem is the simplest of all, since there is nothing to copy into the new node. It has the same code to handle the command line parameters and then it simply creates the target directory and fixed its mode. The script for the `usr/` filesystem is similar to the one for the root, except that it has no need of some of the initial variables, has no files to edit, and no further scripts to execute. The script for the `var/` filesystem contains some significant differences, because not all the content of this filesystem is to be shared among the nodes. Only the subdirectories `cache/` and `lib/` of the `var/` filesystem are to be shared among the nodes, the rest must be node-specific. Therefore the script starts by using the command `tar` to simply copy the independent part of the tree,

```
tar -C $source --exclude ./cache --exclude ./lib -cf - . | \
    tar -C $target -xpf -
```

This pipeline of `tar` commands will copy over the tree of the `var/` filesystem, with the exclusion of the `cache/` and `lib/` subdirectories. The code then proceeds essentially as before, but limiting the searches with the command `find` to the two subdirectories which were excluded, in order to reproduce the shared part of the tree.

In conclusion, running the `make-new-netboot-node` script in the `/pmc/` directory is all that is needed for the creation of a new node. You should do this for each one of your actual compute nodes, possibly keeping the node `0000` as a virtual one. The nodes will be created with the basic adjustments already in place for them to be able to function as the intended hosts. This should be enough for compute nodes, but in the case of remote-boot terminals there may be more to be done, as we will see later. You will note that the scripts for the root, `tmp/` and `var/` filesystems finish quickly, while the one for the `usr/` filesystem may take a little while to complete. The first three should take of the order of tens of seconds to run, while the last one may take from a few to several minutes, depending on the size of the `usr/` filesystem.

4.2.2 The Shared /usr/ Alternative

Since the `usr/` filesystem of the nodes can be completely shared among them, so long as they all have exactly the same set of installed packages, it is possible to treat it in an alternative way, without the subdirectories for each node and the need for building a set of links and a complete directory structure for each node. Here we comment on this alternative and point out its advantages and drawbacks, as compared to the scheme adopted above as the standard. In this alternative scheme one simply exports the `/pmc/usr/` filesystem to all the nodes and, in each node, one directly mounts it as the `usr/` filesystem of the node. In this way all the nodes will be sharing not only the same set of inodes of common files but also the same set of hard links to these files, as well as the same directory structure.

The changes needed in order to do this are simple, one simply eliminates the subdirectories `0000`, `0001`, etc, and puts a single copy of the contents of the filesystem directly in the root of the filesystem. One can do this executing in the directory `/pmc/usr/0000/` the command `mv * ../`, then going up with `cd ../` and removing the directory with `rmdir 0000/`. If there already are any other subdirectories cloned in there, one can delete them recursively using with great care the command `rm -rf 0001/`, and so on. However, if you decide to do this *be very careful*, because errors when using a forced recursive remove such as this one can be *disastrous*. One also changes the file `/etc/fstab` of each node, in order to mount in the mount point `/usr` of the node the NFS filesystem `pmcs00:/pmc/usr` rather than `pmcs00:/pmc/usr/0000`, if we use the node n0000 as an example. In this scheme there is no need of the script

`make-new-netboot-node` for the `/pmc/usr/` filesystem, of course, or of another script which will be discussed later on, to be used for hard-link maintenance purposes.

This scheme has been used by several clusters of both compute nodes and remote-boot terminals for several years, with few problems. This works so well because the use of `/usr/` filesystem by the operating system is completely static under normal operation, meaning that the filesystem is usually not changed. Files are read from it, but nothing is written to it, and no files are deleted. The only situation in which there is write activity to this filesystem is when one installs or removes a package, or when one compiles something inside the `/usr/src/` directory. Occasionally there may be some small problem during the removal of a package in a cluster which uses this scheme, but they are usually easy to deal with. The typical problem is a failure during the removal of a package. Occasionally the pre-removal script of a package will want to make use of one of the files of the package in the `/usr/` filesystem one last time before all the files are removed. Since all the file of the package are removed when one executes the removal in the first node, the removal operation in the second and subsequent nodes may fail. In order to deal with this problem one will have to make adjustments in the collective package manipulation scripts to be discussed in the next section. The basic idea is that, instead of simply running in each node, for example, a command such as

```
apt-get --purge remove <package>
```

in order to completely remove a package, in each node one first runs the command

```
apt-get --reinstall install <package>
```

to recover any missing files in the `/usr/` filesystem, and then runs the previous command to purge the package.

The main advantage of this alternative scheme is that there is no need to run either the `make-new-netboot-node` or the filesystem maintenance script in the `usr/` filesystem. All the scripts take a lot longer to run in the `usr/` filesystem than in the others. While one should expect run times of the order of tens of seconds for the root and `var/` filesystems, it may go up to anywhere from a few to several minutes for the `usr/` filesystem, depending on its size. While this fairly long run time is not such a big problem when creating a new node, which is a one-time operation for each node, it may become bothersome for the regular use of the filesystem maintenance tool. Of course one does not have to run that tool too often either, so this running time issue is not really a very serious one. The other advantage is that one may save up to about 3 or 4 GB in the `usr/` filesystem, which usually is not enough to make the use of this alternative scheme a real necessity.

There are several advantages of the standard scheme over this alternative one. Besides not being subject to the occasional problems with the removal of packages, as explained before, there is the problem that in the alternative scheme the accidental removal by a node, or loss by any other mechanism, of any file in the `usr/` filesystem will

affect all the nodes at once. In the standard scheme only the one node where it occurred is affected by the accident or error. Another advantage is that in the standard scheme one has the ability to have a certain number of different packages installed in different nodes, a possibility which is lost in the alternative scheme. As an example of the usefulness of this possibility, we may note that sometimes a package is needed in only one of a set of remote-boot terminals, for example due to some peculiarity of the hardware, which requires some special driver and support software. Sometimes software which is not part of the Debian distribution will require the substitution of only a few files in the `usr/` filesystem rather than the removal of a package. This cannot be accomplished in the alternative scheme without breaking the system for other terminals.

In short, the standard scheme is more robust and flexible than the alternative one, but takes a bit more disk space and involves fairly long run times of the filesystem maintenance script in the `usr/` filesystem. We recommend that you start by using the standard scheme, and only consider changing to the alternative one if you are dissatisfied with it for some reason. It may turn out that you will need to do a bit of experimentation in order to decide which alternative is the right one for the `usr/` filesystem in your case. It should be mentioned that, if you decide to use the possibility of having different packages installed in different nodes, provided by the standard scheme, then you should pay close attention to the comments in the next section about the `strict` variable which exists in the `hard-link-common-files` filesystem maintenance tools, since in this case you can only use these tools if you keep the variable `strict` set to the appropriate value.

4.2.3 Server System Adjustments

Once you have cloned all the nodes you need, you will be almost ready to have your complete cluster of compute nodes in operation. However, you must still make some adjustments in the server in order to accommodate this complete collection of nodes. Before you do anything else, you must add entries for all your nodes in the file `/etc/hosts` of the server, so it looks like the example below:

```
# JLdL 23Aug04.
#
127.0.0.1       localhost
172.16.129.30   fit.free.univ.com       fit
192.168.0.1     pmcs00.free.univ.com    pmcs00
#
192.168.0.10    n0000.free.univ.com     n0000
#
192.168.0.11    n0001.free.univ.com     n0001
192.168.0.12    n0002.free.univ.com     n0002
192.168.0.13    n0003.free.univ.com     n0003
192.168.0.14    n0004.free.univ.com     n0004
192.168.0.15    n0005.free.univ.com     n0005
192.168.0.16    n0006.free.univ.com     n0006
```

A more complete version of this file can be found in the resources area, in the subdirectory `etc/`, with the name `hosts.3`. You should adjust it, of course, for your own set of nodes. You should also remember to add appropriate entries for each one of the nodes to the `dhcpd.conf` file of the server, and to restart the `dhcpd` daemon after you do that. You will need to record in that file the Ethernet address of the network cards of all the nodes, which you should have looked for and written down while you were configuring and/or programming the cards. These addresses, also known as MAC addresses, are also reported by the boot programs when they come in during the boot sequence.

Next we need to export all the node filesystems to the complete set of nodes. So far we have exported them only for the first node n0000. This can be done by means of a simple list of hosts in the `/etc/exports` file of the server but, if the list is long due to the existence of a large number of nodes, this is not a convenient mode of operation. It is a better solution to use NIS netgroups in order to do the exports. This is a structure by means of which we can attribute symbolic names to groups of hosts, then use these symbolic names instead of the corresponding list of hosts in order to control several types of authorization mechanisms, including NFS exports. It is also possible to create groups of users rather than hosts, which may be used, for example, to limit the access to certain hosts to a certain group of users.

The definition of the netgroups resides in the file `/etc/netgroup` of the server, but its contents will be distributed to the NIS clients, so that the symbolic names of the netgroups may be used in any of the systems of your cluster. One can see therefore that this structure, by requiring the maintenance of a single central table defining all the necessary netgroups, can greatly simplify the administration of the whole cluster. As we will see, NIS netgroups can be used for several other purposes as well, and are in fact quite an essential tool for cluster management. A netgroup is defined by a list of triplets of the form `(hostname,username,domain)`, where the `hostname` refers to some machine in your cluster, `username` to one of your users and `domain` is the NIS domain you chose for your cluster. If you are building a netgroup of hosts, the `username` entry should be a dash, so you will have triplets of the type `(hostname,-,domain)`. For netgroups of users you should have triplets of the type `(-,username,domain)`. The hostname should be fully qualified, so that an entry for the node n0001 should look like

```
(n0001.free.univ.com,-,fityp)
```

You should create a file `/etc/netgroup` in your server in order to define e netgroup called `pmcnodes` for your set of nodes. The file should look like this:

```
# JLdL 23Aug04.
#
# NIS netgroups.
#
# A group for the real PMC nodes.
```

```
pmcnodes \
(n0001.free.univ.com,-,fityp) (n0002.free.univ.com,-,fityp) \
(n0003.free.univ.com,-,fityp) (n0004.free.univ.com,-,fityp) \
(n0005.free.univ.com,-,fityp) (n0006.free.univ.com,-,fityp)
```

Note the uses of backslashes at the end of some lines, turning the next lines into continuation lines. Note also that the first entry in the definition is the name of the netgroup, while all the others are triplets defining the members. There is a more complete example of this file in the resources area, with the name `etc/netgroup`. After you do this, do not forget to go to the directory `/var/yp` of your server and execute `make` in order to refresh the NIS databases. You may now add a new exports entry for the set of nodes, modifying the `/etc/exports` file of the server, which should become something like this:

```
# JLdL 30Sep04.
#
# PMC system to the server itself.
/pmc        localhost(rw,async,no_root_squash) pmcs00(rw,async,no_root_squash)
/pmc/tmp    localhost(rw,async,no_root_squash) pmcs00(rw,async,no_root_squash)
/pmc/var    localhost(rw,async,no_root_squash) pmcs00(rw,async,no_root_squash)
/pmc/usr    localhost(rw,async,no_root_squash) pmcs00(rw,async,no_root_squash)
#
# PMC system to the complete group of compute nodes.
/pmc        n0000(rw,async,no_root_squash) @pmcnodes(rw,async,no_root_squash)
/pmc/tmp    n0000(rw,async,no_root_squash) @pmcnodes(rw,async,no_root_squash)
/pmc/var    n0000(rw,async,no_root_squash) @pmcnodes(rw,async,no_root_squash)
/pmc/usr    n0000(rw,async,no_root_squash) @pmcnodes(rw,async,no_root_squash)
```

Note the syntax `@pmcnodes` to refer to the netgroup. A copy of this file can be found in the resources area with the name `etc/exports.3`. After you finish modifying the `exports` file, do not forget to reload the exports table of the NFS server with

```
/etc/init.d/nfs-kernel-server reload
```

You are now ready to boot up and explore all your nodes, so do so and check them all out, logging into each one via the network or running a remote shell in each one. You will be able to check the system status of all the nodes using the `ruptime` command, which is part of the `rwho` package. You can use it like this:

```
ruptime | grep ^n0
```

You will get an output reporting the status of the cluster of nodes. You will be able to see, for example, that the nodes are idle, that no users are using them interactively and that they have been up for a little more that day and two hours. In this example the output would look like this:

```
n0001         up    1+02:07,     0 users,  load 0.00, 0.00, 0.00
n0002         up    1+02:07,     0 users,  load 0.00, 0.00, 0.00
n0003         up    1+02:07,     0 users,  load 0.00, 0.00, 0.00
n0004         up    1+02:07,     0 users,  load 0.00, 0.00, 0.00
n0005         up    1+02:07,     0 users,  load 0.00, 0.00, 0.00
n0006         up    1+02:07,     0 users,  load 0.00, 0.00, 0.00
```

At this point it is a good idea to install in the server the package `dsh`, containing the so-called distributed shell. It will enable you to run the same command on sets of machines and is, therefore, very useful in clusters. However, since by default it will use the remote shell `rsh` to access the machines, it will be necessary to implement the required `rsh` authorization structures in order for `dsh` to work. We will discuss this kind of access control structure in more detail in Chapter 7, for the time being all you have to do is to put in the `/root/` directory of all your machines, including the server and all the compute nodes or remote-boot terminals, a file named `.rhosts`, with ownership and access authorizations given by

```
-rw-r--r--    1 root     root           79 Feb 14  2004 .rhosts
```

and with content such as

```
# JLdL 26Nov04.
#
# All machines in the cluster.
fit.free.univ.com       root
+@pmcnodes              root
```

This file allows access to the root account from the hosts listed within it. Note the use of the netgroup `pmcnodes` in this authorization structure, allowing us to configure a whole set of hosts with a single entry. With this in place you will be able to use `dsh` to run a command in all the nodes of the cluster. You can do this in any set of nodes you choose by listing their hostnames in the command line with the option `-m`, but here we find another interesting example of the usefulness of the NIS netgroups, since you can submit a whole group of machines with the `-g` option,

```
dsh -g @pmcnodes hostname
```

In this trivial example the command `hostname` will be executed in each of the nodes. You can make it run a little faster with the `-c` option, but the results will not be in any particular order. Have a look at the manual page of the command for the details.

In order to really start using the nodes in normal production mode we still have to create the user accounts in the cluster, distribute the home filesystem by NFS and install a few useful tools and monitoring utilities. Most of this will be discussed later, when we talk about the user environment and the centralized user configurations. Right now we must go on to discuss the maintenance of the cluster of nodes you just got up and running.

4.3 Upgrade and Maintenance

Customarily one uses the Debian package manipulations tools in order to manage software packages in a Debian system. The most common and useful tolls are the `dpkg` and `apt-get` tools, both of which we have already used before. The first of these is a lower-level tool that can install and remove packages, as well as list the status of all packages present in the system, among other things. The second is a higher-level tool, which is network-aware, and which is extremely useful for the installation of new packages and for the upgrading or removal of existing packages. The introduction of the APT (Advanced Package Tool) system by Debian revolutionized the way in which we manage network-centered systems such as clusters. It permits a very transparent use of software mirrors scattered throughout the whole Internet and represents a leap forward in terms of support structures through the network. It would be much harder to install and maintain clusters without the APT system and free access to software mirrors. After a while being able to rely on `apt-get` one wonders how we ever managed to get by without it in the past.

These tools are, however, meant to be used in a single system, obviously they know nothing about the cluster structure we describe here. We need, therefore, an even higher-level set of tools in order to perform these functions in a whole cluster is a coordinated fashion. Fortunately, the Debian tools were very wisely designed as command-line interface tools, so it is a simple matter to build scripts which use them. We will discuss here a few scripts that may be used for package administration and filesystem maintenance in a cluster. These scripts can be understood as simple extensions of the usual Debian tools, they are sufficient for the usual administration shores, but there is no claim that they are a complete new layer of tools in any way. However, if new needs arise which are not predicted in the current version of these tools, there should be no difficulty in using the ideas and techniques described here to upgrade these scripts or write new ones in order to satisfy these needs. The tasks which are contemplated in these scripts are the installation, upgrade and removal of packages, and the maintenance of the hard-link structure of the cluster filesystems.

Note that the use of these tools is *not* recommended for a full-fledge Debian distribution upgrade, changing from a major release to the next major release. They are meant for including new software in an existing system, making security upgrades of the software, and other such incremental operation. For a complete distribution upgrade other ideas and techniques, which we will discuss later on, are both faster and less error-prone.

4.3.1 Package Administration

As we saw in the previous section, in order to install, remove or upgrade packages in one of the nodes, one can use the usual Debian tools. It follows that, in order to do the same on all nodes, all that we must do is to repeat the operation in each node. One could, for example, do this through the network, logging remotely into each node and

executing the necessary tasks. It would be quite simple to write a script to automate this kind of task. In fact, the first clusters of this type which were built, years ago, used to be managed this way. This scheme has, however, the serious disadvantage of requiring all nodes to be up and running in perfect order, otherwise one cannot perform the operation in all the nodes in immediate sequence, ensuring that their systems are kept in proper synchronism.

The scheme which we will describe here combines this idea of looping over the nodes with the technique of using the `chroot` command in order to simulate each node within the system of the server itself, which we already used before. In this way we will be able to keep all nodes updated in a coordinated way, regardless of the state of the hardware of each one. In fact, one may even clone and start maintaining a set of new nodes *before* their hardware actually arrives from the supplier, or keep maintaining a node which broke down and is under maintenance.

In this subsection we will describe only the package-manipulation scripts, the ones meant for filesystem maintenance will be described in the next subsection. There are two scripts involved in the operations of package administration, one called `apt.chroot` and another one called `multi-apt.chroot`. We will use only the `apt-get` utility in these scripts. The first script is the one which actually performs the operations in each node, the second does the looping over the nodes. Let us start with the second script, which is meant to be run on the server. You will find a copy of it in the resources area, in the subdirectory `pmc/`. It is a good idea to edit it with the `emacs` editor in order to have it fully colorized on the screen, so you can peruse it while you read this.

The `multi-apt.chroot` script is meant to be run in the `/pmc/` directory of your server, the root directory of the cluster of nodes. It starts with a block of code to check for the existence of spurious loopback NFS mounts of the cluster filesystems, possibly from a previous run of the program which was interrupted by some serious error. The first two lines of this block define a target for the `egrep` search command, and record the basename of the script in a variable, for posterior use in the error messages,

```
set etarg = "^pmcs00:/pmc[^ ]* /pmc/"
set bname = `basename $0`
```

The target contained in the variable `etarg` will match any NFS mounts from the host pmcs00 for a filesystem within the `/pmc/` directory, mounted in loopback fashion within the `/pmc/` directory itself. The code that does the checking is this:

```
cat /proc/mounts | egrep -q "$etarg"
if ( "$status" == 0 ) then
    echo ${bname}: ERROR: there are spurious chroot mounts:
    cat /proc/mounts | egrep "$etarg"
    exit 1
endif
```

Note that the mount is searched for in the `/proc/mounts` file, since it may not be recorded in the `/etc/mtab` file of the server. If a match is found the command `egrep` exist with status equal to zero, which is passed to the shell variable `status`. In this case an error message is written out, the spurious mounts are shown in the terminal, and the program exits in error. If this ever happens you should analyze the situation and undo the spurious mounts manually before trying again. Next comes a block of code that loops over the nodes and executes a few commands,

```
foreach node ( [0-9][0-9][0-9][0-9] )
    echo "======== n$node ========"
    cp -pf /pmc/apt.chroot /pmc/$node/root/apt.chroot
    chroot /pmc/$node /bin/tcsh -f /root/apt.chroot
    find /pmc/fake-var-run/ -name \*.pid -exec \
        echo ${bname}: WARNING: stray PID file found: \{\} \;
end
```

The structure `[0-9][0-9][0-9][0-9]` is a wildcard which matches any file within the current directory with a name consisting of four digits, which should correctly select the node directories. The variable `node` will traverse the list of these directories. The first line of code within this block simply prints out a progress-report separator, indicating the node which is currently being worked on. The next line copies the other script, called `apt.chroot`, into the `/root/` directory of the node. This is the script which contains the commands which will perform the actual package manipulation actions within the node. The following line executes a `chroot` command with two arguments. When executed this way, the `chroot` command changes the root to the directory given as the first argument and executes the program given as the second argument, which must be within the new root, of course. In our case here this is a shell, which is invoked to read and execute the file `/root/apt.chroot`. The option `-f` to the `tcsh` command tells the shell not to execute the user customization files, so that it will run a bit faster.

The last line within this block looks for any stray files with names ending with `.pid`, that may be in the directory `/pmc/fake-var-run/` after the `apt.chroot` script completes and the `chroot` command returns, and writes out a warning message if it finds any. The directory `/var/run/` of the node contains files with the PID numbers of daemons which are running on the node. Sometimes the Debian package manipulation tools will look in there in order to find the PID number of a certain daemon, so that they may shut the daemon down during an upgrade. If this number happens to be equal to the PID number of some other process running on the server, the process on the server will be killed instead, with unpredictable consequences which, however, cannot be good. In order to avoid this the `apt.chroot` script hides this directory by mounting over it the directory `fake-var-run/`, so that the tools will not find a PID file and will think that the daemon is not running. When this happens the Debian tools usually do the right thing: they install the package but do not try to start the daemon.

If it ever happens that some package misbehaves during installation and this precautionary search returns some result, you should be aware that some daemons may have been erroneously started in your server instead of in the nodes. You may be able to sanitize the situation manually, identifying and killing the rogue processes, or you might want to try to use a piece of code like the following one in order to kill them as the `multi-apt.chroot` script works along from node to node,

```
foreach file ( `find /pmc/fake-var-run/ -name \*.pid` )
    set pid = `cat $file`
    echo ${bname}: WARNING: killing process $pid
    kill $pid
    if ( $status != 0 ) kill -9 $pid
    echo ${bname}: WARNING: removing file $file
    rm -f $file
end
```

You will find this code suggestion included but commented out in the copy of the `multi-apt.chroot` in the resources area. Since we have not seen such an event happen so far, it has *not* been thoroughly tested, so use it only with the greatest caution. The last block of code in this script is very similar to the first one, it checks once more for the existence of spurious loopback NFS mounts of the cluster filesystems, this time after all the work is done on the nodes, and writes out a warning message if it finds any. There should not be any, since the `apt-chroot` script unmounts all the node filesystems before it exits.

Let us now examine what the `apt-chroot` script does when it runs within each node. Remember to edit it with `emacs` so you can peruse its text. The first few lines of code perform a safety check. Since this script will make loopback NFS mounts, it should in fact be executed only on the cluster server, never on the nodes themselves, which already have their normal system mounts in place. Hence, the first line of code stores the hostname in a variable and the following block checks whether or not the script is being executed in the server, and exits in error if not. Next one finds the line `onintr cleanexit`, which instructs the shell to go to a line labeled as `cleanexit:` if it receives a keyboard interrupt (usually `[Ctrl]-[C]`). This label is just before the last block of code in the file, which does a clean exit, unmounting all the filesystems of the node before exiting. In this way, keyboard interrupts will not cause the script to exit leaving stray mounts. Note that this does not work for errors in the commands contained within the script, but only for keyboard interrupts. The possibility of all such errors should be either predicted or found out by experimentation, and must be handled individually along the script. The next block of code starts the preparation for the package administration operations, by mounting all the filesystems of the node:

```
foreach fs ( /tmp /var /usr )
    @ ec = 0
    again1:
```

```
    mount -n $fs
    if ( $status != 0 ) then
        echo -n "WARNING: cannot mount filesystem ${fs}: "
        @ ec = $ec + 1
        if ( $ec < 6 ) then
            echo "trying again if 5 seconds..."
            sleep 5
            goto again1
        else
            echo "ERROR: failed 6 times, quitting..."
            goto cleanexit
        endif
    endif
end
```

It loops over all the filesystems except the root, which does not have to be mounted. Within the loop it starts an error counter, which will be used to handle possible errors in the mount operations. Note that there is a label `again1` marking the re-entry position for a mount re-trial. It then tries the mount and checks for an error status after the command. Note the use of the option `-n` of the `mount` command. This will prevent the command from recording the mount in the `/etc/mtab` file of the node, which could interfere with the operation of a running node. In case there is an error the `if` block which follows writes out an error message and waits for a few seconds before trying again. Note the shell arithmetic to increase the error counter. The mount will be tried up to 6 times at 5-second intervals, before the script gives up and does a clean exit. This code is in place because there have been events of mount errors due to NFS port conflicts in servers with many mounts and many nodes. We do not know the exact cause of these errors, which may or may not be related to the fact that these clusters were using NFS by TCP when it was still experimental; or they may be caused by timing problems due to the fact that in these circumstances many mounts are made and unmade in quick succession. In any case, a wait of a few seconds is usually enough for the ports to be freed and for a new mount attempt to succeed.

The next block of code is very similar to the previous one, it makes an additional mount in order to mask the `/var/run/` directory, for the reasons explained before. It takes exactly the same precautions discussed above regarding the possibility of mount errors. The only differences are the absence of the outer loop, the fact that the `mount` command is used with two arguments instead of one, because the mount of the `/pmc/fake-var-run/` directory of the server on the `/var/run/` directory of the node is of course not contemplated in the `/etc/fstab` file of the node, and the use of a different label `again2` instead of `again1`, because labels must of course be unique within a script. After the mounts are completed the script does a `rehash` in order to refresh the executable search path. This is necessary because when the chroot shell was started not all the filesystems where in place and hence the mounts just completed made available additional executable files which are not yet in the hash table that the

shell keeps for them.

With all the system mounts in place, we are ready to start doing package manipulations within the node. The next line of code defines the list of packages which will be dealt with, using a vector variable to store it in,

```
set packs = ( lapack-dev lapack-doc )
```

This is where you should list the packages to be worked on. If there is any need, you may use backslashes at the end of lines in order to create continuation lines in this statement. Next the script stores in the variable `aopts` the options which are to be used when running the `apt-get` utility. The option `-y` is used for actually doing the operations in batch (non-interactive) mode, since it instructs the utility to answer with "yes" to all questions asked by the package scripts. The options `-sy` will make the utility simulate the operations, without actually changing anything in the filesystems of the node. This alternative is to be used only for testing purposes and is, therefore, commented out in the script. Then there is the set of lines which does the actual package-manipulation work,

```
apt-get update
#apt-get $aopts install $packs
#apt-get $aopts --reinstall install $packs
#apt-get $aopts -f install
#apt-get $aopts remove $packs
#apt-get $aopts --purge remove $packs
apt-get $aopts upgrade
apt-get clean
```

You may need to comment out, uncomment and/or adjust some of the lines in order to choose the type of action you require. The three lines which are *not* commented out in this example are the ones usually needed for security upgrades in nodes configured to use FTP or HTTP in order to access a Debian mirror. You may use here any command that you would use in the command line of a terminal to manage Debian packages. The last block of code finalizes the task by unmounting all the filesystems of the node,

```
cleanexit:
foreach fs ( /tmp /var/run /var /usr )
    @ ec = 0
    again3:
    set error = `umount -n $fs |& cat`
    if ( "$error" != "" && "$error" != "umount: ${fs}: not mounted" ) then
        echo -n "WARNING: cannot unmount filesystem ${fs}: "
        @ ec = $ec + 1
        if ( $ec < 6 ) then
            echo "trying again in 10 seconds..."
            echo "         (error message was: $error)"
            sleep 10
```

```
            goto again3
        else
    echo "failed 6 times, giving up."
    echo "ERROR: doing a lazy unmount for $fs..."
            umount -nl $fs
        endif
    endif
end
```

Once more we see a loop through the filesystems, which are unmounted in an appropriate order, and there is error handling here too, this time using the label `again3`. First a straight `umount` is tried, and the resulting error message, if any, is stored in the variable `error`. If there is an error message which is not due to the fact that the filesystem is simply not mounted, the `if` block that follows starts re-trying as before, but this time in 10-second intervals. If the unmount trials fail six times then a lazy `umount` is tried. This lazy unmount will make the kernel detach that particular NFS filesystem from the global filesystem structure immediately, and then wait to clean up all references to the filesystem as soon as it is not busy anymore.

This should seldom be necessary unless a package which is being installed leaves something running on the background. The typical case is a package of some X11-related software which will run the `update-menus` Debian script, in order to include the software in the menu systems of various window managers. If the cluster server is not very fast and/or there are many menu items to do, for example in a very full system, the 60-second waiting period may not be enough and may force a lazy unmount. If the situation gets to this point, this lazy unmount may cause stray temporary files to be left within the mount points `tmp/`, `var/` or `usr/` of the root of the node, or in the directory `fake-var-run/`. Besides, due to the 60-second waiting period, X11-related packages and other packages which leave something running on the background may take significantly longer to install than other packages. This will seldom happen in compute nodes but may often happen in remote-boot terminals. If some such backgrounded operation is not completed you may need to fix things, usually by running the script or configuration program by hand. In the typical case of the `update-menus`, you may run it by hand at any time, try for example executing it in verbose mode, `update-menus -v`, to see what it does.

Once you have these scripts in place, dully adapted for your particular cluster, you should try them out for a standard security update/upgrade. Just go to the `/pmc/` directory of the server and execute the `multi-apt.chroot` script. No packages should actually be upgraded unless you are using a mirror of the Debian security site or you are using the testing distribution before its release. As a rule you should try to avoid interrupting the script, but if you feel you must, try to do so immediately after the progress-report line of a new node shows up. In this way you will probably get the process while the system mounts are being executed and before the APT utility starts,

so that there is less chance that the `apt-get` command will get interrupted and leave the system of a node in some funny state. Of course, having the system in such a funny state is not such a big problem either, since the Debian tools themselves can fix it and will even tell you how to do it, next time you run one of them.

4.3.2 Filesystem Maintenance

After you run the package manipulation script the filesystems of your cluster will have acquired at least some new files which will not be hard-linked to each other although they are identical. In the `/var/` filesystem this will happen even if no packages actually got installed or upgraded, because the `apt-get update` command will have downloaded from the Debian mirror a new set of package database files. This is not a big problem in itself, but it does imply a slight increase of disk occupation above the minimum possible. After some more packages are installed it may be time to do some filesystem maintenance, in order to decrease the disk occupation back to its minimum value. The set of scripts with the name `hard-link-common-files` found in the resources area, in the subdirectories `pmc/`, `pmc/var/` and `pmc/usr/` are meant for this task. In their basic mode of operation they will search the filesystems, looking for identical files in the corresponding subdirectories within two different node trees, and will hard-link together the ones which are found.

Besides hard-linking identical files, these scripts may also be set to create hard links which are missing in one of the two node trees that they compare. The scripts contain a shell variable named `strict` which is set to `yes` at the beginning of the code. With this variable set in this way the scripts will not create missing hard links, and will keep strictly to hard-linking to each other identical files found on the two node trees being examined. In order to enable the creation of missing hard links, you must set this variable to `no`. In this case the scripts will create the missing hard link whenever a file found in one of the node trees is missing in the other node tree. This will effectively tend to homogenize the node trees, making them as similar as possible. It will also make the scripts take a bit longer to run.

Although this may help fix a broken node tree, care must be taken with some files which are generated by the system itself, in an automatic fashion. The presence of some of these files may be related to status information pertaining to only a certain node, and therefore one must avoid cloning them in other nodes. An example of this is the file `/etc/nologin` which is supposed to block logins in the system. In other cases such files should be cloned, for example in the case of the TEX font files which are generated by the system as the need arises and then stored within the directory `/var/cache/`. In this case cloning the files to the other nodes will avoid the necessity of generating them again in each node when they are needed. This example is, of course, more appropriate for remote-boot terminals, but is enough to illustrate the idea. The scripts are already set to deal with all the currently known exceptions and problems of this kind but, as the software develops and more software gets installed into the nodes, it may be necessary

to make a few simple adjustments to them from time to time. Note that stray files such as editor backup files that are left in one node will also be distributed to all nodes, so if you decide to use this capability it is a good idea to always clean up after some work is done in a node.

There is one case in which one certainly does *not* want to create missing links, namely in a cluster in which there are different sets of packages installed in various nodes. Such a scheme is feasible in terms of disk occupation so long as the differences among the sets of packages in each node are not too large. Since usually there is a fairly large set of packages common to all nodes, this possibility can be put to good use without an excessively large increase in disk occupation. However, in this case allowing the scripts to create missing hard links will distribute to some nodes hard links to files which are not installed in them, possibly breaking the system. In order for the scripts to be able to function in this kind of cluster the variable `strict` *must* be set to `yes`. You might want to use strict linking also if you just do not want to bother about possible problems and exceptions. The only thing that you do loose if you do this is the ability to distribute some cached files among the nodes, saving the effort of generating them again in each one. It is not really such a big loss.

The `hard-link-common-files` scripts for the root and `usr` filesystems are very similar and somewhat simpler than the one for the `var` filesystem. We will describe here the root script and then comment on the differences found in the others. Make sure you have the script open in the `emacs` editor as you follow this. In the first line of code we see once more the structure `[0-9][0-9][0-9][0-9]` being used to select the node directories, which in this case are stored in the variable `nds`. Then comes the definitions of the variable `strict` and of a variable `sep` containing a separator line. Next we see the start of a large loop over all the node directories, which encompasses all the rest of the code and uses a counter `jnd`, which starts at 1. This is a loop over what we will call the *template* directories. The template directory is the node directory currently being used as the basis for comparison with the others. The directory the template is being compared to is called the *target* directory. Within this outer loop there are several blocks of code we must comment on. The code starts by storing the name of the current template directory in the variable `tpl` and writing a separator line and a message. Then a block of code checks the mount points of the other filesystems within the root,

```
foreach dir ( tmp var usr )
    if ( `\ls -1a $tpl/$dir | wc -l` > 2 ) then
        echo `basename $0`": ERROR:"
        echo "  spurious content detected in mount point $tpl/$dir;"
        echo "  you must clean it up before running this again."
        echo $sep
        exit 1
    endif
end
```

It looks for any spurious content within the mount points, which should be empty directories. If it finds any, it issues an error message, prints a final separator and exits in error. This code is in place because there have been cases of files or directories being erroneously created under the mount points by a package installed with the chroot shell. This may happen if the `apt.chroot` script has any difficulty unmounting the filesystems and uses a lazy unmount, for example. Packages belonging to the KDE and Gnome desktops are known to cause this kind of thing. If this ever happens you will need to examine the situation in order to clean up the mount points by hand. Usually the files and directories involved are either temporary or things which are regularly and automatically created within the system, so that they can be simply deleted without any serious loss.

Next comes a block of code to define the collection of target directories for the given template directory. This has to be done in different ways depending on whether or not you are using strict linking. In non-strict mode each pair of directories must be tested both ways, because there may be missing links in either one of them. This means that, if you have N nodes, there will be $N(N-1)$ comparisons to make. In strict mode, however, each pair may be tested only once, because there will be action to take only if the same file exists in both nodes. In this case we need only half as many comparisons as in the non-strict case, so the script will take less time to run in strict mode. This block of code starts with an `if` structure,

```
if ( $strict == no ) then
    set ind = 0
    set trg = ""
    while ( $ind < $#nds )
        @ ind = $ind + 1
        if ( $nds[$ind] != $tpl ) set trg = ( $trg $nds[$ind] )
    end
else
    set ind = $jnd
    set trg = ""
    while ( $ind < $#nds )
        @ ind = $ind + 1
        set trg = ( $trg $nds[$ind] )
    end
endif
```

The collection of target directories is stored in the variable `trg`. In non-strict mode it consists of all the node directories except the template directory. In strict mode it consists of all the directories which are after the template in the ordered list of node directories. Note that in strict mode the list will be empty in the case in which the template directory is the last one. The next block of code uses the `find` command to find all the common files within the directory tree of the template node which have less than the number of hard links it would be expected to have if it were fully linked, that is, linked on all the nodes. The `-links -<number>` option of the `find` command does

this test. The "`-`" before the number has the meaning of "less than" and the number is obtained from the value of the variable `#nds`, which returns the number of elements in the vector variable `nds`. The command `find` is instructed to produce on its output the full path of the files it finds, and this output is then passed through a pipeline,

```
echo -n "finding hard-link candidate files in the template... "
find $tpl -type f -and -links -$#nds -exec \ls -l \{\} \; | \
    tr -s " " | \
    cut -d " " -f 9- | \
    cut -d / -f 2- | \
    egrep -v '^$|^etc/nologin|^etc/upstatus' | \
    sort >! hard-link-candidates.$tpl
set ncand = `cat hard-link-candidates.$tpl | wc -l`
echo "( $ncand )... done."
```

The pipeline acts as a filter, manipulating the string resulting from the `find` command. It first squeezes all multiple spaces into single spaces, then cuts out the name of the file, then cuts off the first directory name in the path, which is the name of the node directory. Then it does a reversed search with the command `egrep` in order to avoid empty lines and the files `etc/nologin` and `etc/upstatus`, both of which relate to node status information. Last, the resulting list of relative paths to files if saved to a temporary file named `hard-link-candidates.*`, with an extension given by the name of the node directory. This file contains therefore the set of candidate files which the last part of the code will verify and hard-link if possible. Note that this block of code also reports on the terminal the number of candidates it found in the current template.

The code which does the actual hard-linking is contained in the `bash` subroutine `hard-link-common-files.sub` which is called next. The use of another shell at this point is convenient because it provides a programming structure which allows us to deal with files with special characters such as spaces and bangs (exclamation marks) within their names. Note that the routine is kept in the directory `/usr/local/lib/scripts/` and that it is only called if the set of target directories is not empty. This single subroutine is used by all the `hard-link-common-files` scripts. Three variables are passed to the subroutine, the variable `strict`, the variable `tpl` containing the template directory and the variable `trg` containing the list of target directories. This last one is contained within quotes, which causes the whole list within this vector variable to be passed as a single variable to the `bash` script, which simplifies the passing of the list to the subroutine. After the subroutine is called, the `jnd` counter is incremented and the outer loop closes. The only thing that the script does after that is to write out a final separator line.

The subroutine `hard-link-common-files.sub` performs the central task of hard-linking identical files which are not already links to one and the same file. In order to do this it checks, for each candidate file in the template directory, whether the corresponding files in the target directories have the same content and whether they already are hard

links to that file. If the files in the target directories are identical to the file in the template directory but are not links to that file, then they are deleted and a link to the file in the template directory is created in their place. The subroutine will also create any missing links if the `strict` variable is set to `no`. As we discuss this script, node the various small differences in syntax between the `tcsh` and `bash` shells. The first few commands recover from the command line parameters the values of the three necessary variables, which are given here the same names that they have in the calling script. The only difference is that `trg` is here a single (scalar) variable containing a string with the names of the target directories separated by spaces. The `while` loop that follows is a very interesting structure, the one which was alluded to before. What it does is, first, to read in a record-by-record fashion the temporary file `hard-link-candidates.*` which contains the list of candidate files and, then, to loop over these records, so that it loops over the candidate files avoiding trouble with special characters within their names,

```
cat hard-link-candidates.$tpl | \
while read file ; do
    [SNIP]
done
```

Here the token `[SNIP]` represents the intervening code which was cut out. One sees that the contents of the temporary file are sent to the standard output channel and then piped to the following line, which contains the `while` token of the loop. The pipe is actually received by the `read` shell command, which reads the input record-by-record and inserts each record in turn into the variable `file`. The loop is then executed with this variable containing one record of the input at each turn. The intervening code contains another loop, a `for` loop which goes over the target directories,

```
    for dir in $trg ; do
        [SNIP]
    done
```

This inner loop contains a three-layered structure of `if` blocks. The first one tests whether or not the candidate file exists in the target,

```
        if [ -e  "$dir/$file" ] ; then
            [SNIP]
        else
            [SNIP]
        fi
```

Note the use of quotes to protect any special characters contained in the variable `file`. If the file exists then the following block is executed,

```
dif=‘diff -q "$tpl/$file" "$dir/$file"‘
if [ "$dif" == "" ] ; then
    if [ ‘\ls -i "$tpl/$file" | cut -c 1-8‘ != \
        ‘\ls -i "$dir/$file" | cut -c 1-8‘ ] ; then
        echo "hard-linking: $dir/$file"
        rm -f "$dir/$file"
        ln "$tpl/$file" "$dir/$file"
    fi
fi
```

In the first line of this block the set of differences between the two files, if any, or the message saying that two binary files differ, if they are binary files, if stored in the variable `dif`. If this variable contains the empty string, indicating that the two files are identical, the code goes on to verify whether or not they are already links to the same file. The structures inside the test area of the `if` statement will return the inode numbers of the two files. If they are the same, then the files are already links to the same file, otherwise the code proceed to remove the file in the target directory and exchange it by a hard link to the file in the template directory. A message reporting what is being done is written to the screen. If the candidate file does not exist then the following block is executed,

```
if [ $strict == no ] ; then
    echo "making MISSING link: $dir/$file"
    mkdir -p ‘dirname "$dir/$file"‘
    ln "$tpl/$file" "$dir/$file"
fi
```

This is only done if the `strict` variable is set to `no`. If it is, then the directory which should contain the file is created, as well as all its parent directories if they do not yet exist. When the `mkdir` command is used with the `-p` option it is not an error if any or all of the directories involved already exist. The hard link to the file in the template directory is created. Here too a message reporting what is being done is written to the screen.

The script `hard-link-common-files` for the `usr/` filesystem is very similar to the one for the root. The only differences are that it lacks the code for checking for spurious content within the mount points, as well as the filtering exceptions for the files `etc/nologin` and `etc/upstatus`, all things that obviously belong only to the root. The script for the `var/` filesystem does contain some more significant differences. It too does not have the code just mentioned, of course, since that code belongs only to the root. But in this case there are some further differences. The first important difference is that only the shared part of the tree is scanned for common files, of course. This includes only two subdirectories of `var/`, namely the `cache/` and the `lib/` subdirectories. Further, some subdirectories of `lib/` should also be avoided, because they either are too active to be worth maintaining or contain files with status information pertaining to

some piece of software. The first block of code we must discuss here has the objective of building the appropriate list of subdirectories of the `var/lib/` directory, to be stored in the variable `libdirs`,

```
set etarg = "^$tpl"'/lib/$'
set etarg = "$etarg|^$tpl/lib/gnome"
set etarg = "$etarg|^$tpl/lib/kde"
set etarg = "$etarg|^$tpl/lib/auctex"
set etarg = "$etarg|^$tpl/lib/afterstep"
set etarg = "$etarg|^$tpl/lib/wnn"
set etarg = "$etarg|^$tpl/lib/nfs"
set etarg = "$etarg|^$tpl/lib/alsa-base"
set etarg = "$etarg|^$tpl/lib/ntp"
set etarg = "$etarg|^$tpl/lib/xdm"
set libdirs = `find $tpl/lib/ -type d -maxdepth 1 | egrep -v "$etarg"`
set tsd = ( $tpl/cache $libdirs )
```

It starts by building a search target for the `egrep` command, in the variable `etarg`. Let us recall here that the variable `tpl` contains the name of the template directory, within which the search will be carried out. This search target includes the `lib/` directory itself, the subdirectories `lib/gnome/` and `lib/kde/` of the Gnome and KDE desktop environments, and several other directories relating to specific pieces of software. The `find` command which is executed within the `lib/` subdirectory of the template looks for immediate subdirectories of it and has its output stored in the variable `libdirs`, after a reverse search with `egrep`, which eliminates from this output the directories which are included in the `etarg` variable. The result of all this is that the `libdirs` variable contains a list of the subdirectories of `lib/` which should be searched for common files to be hard-linked together. Next the variable `tsd` is defined, with a complete list of the target subdirectories of the `var/` filesystem to be searched for common files, including the `cache/` subdirectory and the list of subdirectories of `lib/` just constructed. The next block of code executes the search, using again the command `find`, in a fashion similar to the scripts of the root and `usr/` filesystems, but with the search limited to the collection of selected subdirectories,

```
set etarg = "/lib/emacs/lock/"
set etarg = "$etarg|/cache/man/"
echo -n "finding hard-link candidate files in the template... "
find $tsd -type f -and -links -$#nds -exec \ls -l \{\} \; | \
    tr -s " " | \
    cut -d " " -f 9- | \
    egrep -v "$etarg" | \
    cut -d / -f 2- | \
    egrep -v '^$' | \
    sort >! hard-link-candidates.$tpl
set ncand = `cat hard-link-candidates.$tpl | wc -l`
echo "( $ncand )... done."
```

Note that some further exclusions are made at this point, namely the exclusions of the third-level subdirectory `lib/emacs/lock/` of the `lib/` directory and of the `cache/man/` subdirectory of the `cache/` directory. For this purpose there is an extra `egrep` command in the pipeline after the `find` command. In this way the script of the `var/` directory does its work on the largest possible subset of the filesystem where its action should cause no trouble.

This completes our exploration of the code of the filesystem maintenance scripts. Note that one is in no way obliged to run these scripts every time that packages are installed or upgraded in the nodes. The nodes will run quite happily with packages in which some or all files are unlinked. The reason to run these scripts is to keep the disk occupation on the server down. Note also that the files `hard-link-candidates.*`, which are left after the scripts run, may serve as a record of what is different in each node. They will list all files which are either not linked to all nodes or absent from some other node. Here is an example of the output that the script will produce in the root filesystem of a small PMC:

```
fit_ROOT:/pmc# ./hard-link-common-files
------------------------------------------------------------------------
current template is: 0001
finding hard-link candidate files in the template... ( 6 )... done.
------------------------------------------------------------------------
current template is: 0002
finding hard-link candidate files in the template... ( 6 )... done.
------------------------------------------------------------------------
current template is: 0003
finding hard-link candidate files in the template... ( 6 )... done.
------------------------------------------------------------------------
current template is: 0004
finding hard-link candidate files in the template... ( 6 )... done.
------------------------------------------------------------------------
current template is: 0005
finding hard-link candidate files in the template... ( 6 )... done.
------------------------------------------------------------------------
current template is: 0006
finding hard-link candidate files in the template... ( 6 )... done.
------------------------------------------------------------------------
```

The number of files not linked reported by the script, which is 6 in this case, is the minimum that one can have in the root of a PMC, assuming that all nodes have exactly the same set of installed packages. The expected minimum number in the `var/` filesystem is 1 or 2, and for the `usr/` filesystem it is 0. Here is the list of entries contained in these files for the root filesystem:

```
fit:/pmc> cat hard-link-candidates.0001
etc/exim/exim.conf
etc/fstab
```

```
etc/hostname
etc/mtab
etc/network/interfaces
root/.Xauthority
```

These are mostly the files which define the node and files which are created automatically by the system. The output of the script in the case of a simple cluster of terminals can be seen below:

```
fit_ROOT:/trm# ./hard-link-common-files
-------------------------------------------------------------------------
current template is: 00
finding hard-link candidate files in the template... ( 12 )... done.
-------------------------------------------------------------------------
current template is: 01
finding hard-link candidate files in the template... ( 12 )... done.
-------------------------------------------------------------------------
current template is: 02
finding hard-link candidate files in the template... ( 12 )... done.
-------------------------------------------------------------------------
current template is: 03
finding hard-link candidate files in the template... ( 12 )... done.
-------------------------------------------------------------------------
current template is: 04
finding hard-link candidate files in the template... ( 12 )... done.
-------------------------------------------------------------------------
current template is: 05
finding hard-link candidate files in the template... ( 12 )... done.
-------------------------------------------------------------------------
current template is: 06
finding hard-link candidate files in the template... ( 12 )... done.
-------------------------------------------------------------------------
```

In this case we have 12 files which are specific to each node. This is close to the minimum possible for a cluster of terminals, usually one has more than 12 such files on larger clusters. The expected minimum number in the `var/` filesystem is 1 or 2 in this case also, and it is 0 for the `usr/` filesystem, just as in the case of compute nodes. All this is true assuming that all the terminals have exactly the same set of installed packages. If there are different sets of packages installed in various terminals, the numbers of node-specific files will be much larger than these. Here is a typical list of node-specific files in such a small cluster of terminals with homogeneous package sets:

```
fit:/trm> cat hard-link-candidates.01
etc/X11/XF86Config
etc/X11/XF86Config-4
etc/exim/exim.conf
etc/fstab
etc/gpm.conf
```

```
etc/hostname
etc/modules.conf
etc/modutils/ide-JLdL
etc/modutils/scsi-JLdL
etc/mtab
etc/network/interfaces
root/.Xauthority
```

We see here that, in addition to the files usually found in clusters of compute nodes, there are files relating to the X11 windowing system and to kernel modules needed to drive various types of I/O devices. As the hardware becomes more heterogeneous the number of these files will increase. On a larger cluster of terminals, still with homogeneous package sets, the typical number is close to 30 or so.

With the tools described here and the willingness to adapt them to your particular situation, you should have no difficulty in doing the basic package administration of your clusters. Of course, this is only part of all that is involved in the complete administration of computer systems, but it is the only part which is very specific to clusters. In later chapters we will discuss several other issues which are also important for cluster management, such as disk quotas, resource usage limitations, the default user environment, backup strategies, access control, etc.

Chapter 5

Remote-Boot Terminals

Both the hardware and the software of remote-boot terminals are supersets of those of compute nodes, except perhaps for a few adjustments in emphasis and priority. However, quite differently from the case of compute nodes, the main objective here is not processing performance, but instead to provide the users with a sufficient number of access windows into the overall cluster system. These access windows may be in the form of simple ASCII terminals or of X11-capable graphical terminals running desktop software within a windowing system. Usually both will be available in each terminal. The relevant performance characteristics here are those leading to the comfortable use of the systems by the users, which include a reasonably large monitor, good graphics and sound performance and possibly a few handy input/output units.

Almost all that was said in previous chapters about compute nodes is also valid for remote-boot terminals. Therefore, in this chapter we will frequently refer the reader back to those chapters, and will include here only a discussion of the changes and adjustments that should be considered in the parts common to both types of machine. We will also discuss in detail the components of the hardware which are specific to the terminals, such as the video and sound subsystems, as well as the corresponding software elements. The software installation sequence of remote-boot terminals is also very similar to that of PMC nodes, and will also be treated here by comparison to that case, and by the required completion when necessary.

An interesting difference between remote-boot terminals and compute nodes is that terminals may be combined with one or more systems installed in local disks, in multiple-boot machines. One very important difference is that, unlike a cluster of compute nodes, which are typically all assembled together in a closed room, at least some terminals are usually scattered around the facilities of the workgroup, wherever the users need them. Terminals are therefore more exposed to environmental hazards and hence in this case assembling the hardware in standard computer cabinets is an absolute necessity, not an option.

Specially if you are going to re-use existing machines, the hardware of remote-boot terminals is likely to be much more heterogeneous than that of compute nodes. The

structure presented here is meant to support such an heterogeneity, including the reutilization of older machines. However, it must be said that a large amount of heterogeneity will also imply an increase in the necessary amount of installation and configuration work, as well as an increase in the overall complexity of the cluster. If the main objective in building a cluster of remote-boot terminals is to simplify installation and maintenance, it is advisable that the set of terminals be as homogeneous as possible.

If all the terminals have exactly the same hardware, then you will be able to make available new access windows by means of a completely automatic cloning process, which can be done simply and easily. If each terminal has different hardware, then each one may present new installation and configuration problems during the installation. Of course, during a revolutionary period such as the one we are living now, with the hardware changing all the time, it may be difficult to ensure that you have a set of absolutely identical terminals unless you purchase the whole set in a single lot. If you must purchase more than one lot, at different times, and hence introduce a limited amount of heterogeneity into the cluster, this is still much better than having a set of completely different machines.

5.1 Completion of the Hardware

Conceptually, a remote-boot terminal is basically a compute node with some extra hardware meant to allow the interactive use of the machine by the user in a graphical console. Besides having all the same basic parts, it will have also video, sound and input/output subsystems, including at least a monitor, a keyboard, a mouse, a headset and possibly a few I/O units. A remote-boot terminal does not need to have a particularly fast processor for its most typical functions, but it is important that it have a sufficient amount of RAM. A fast CPU is only justifiable if the routine use of the terminal will involve the frequent execution of some computationally intensive task, such as complex image manipulation, 3-dimensional rendering, movie viewing or heavy computer games.

However, if a fast processor is used and a sufficient amount of RAM is installed on the machine, then a remote-boot terminal may double as a compute node too, so long as their users will not shut them down or reboot them into other systems with any appreciable frequency. Usually an intensive job or two running in the background with low priority will not disturb the interactive use of the machine as a terminal, so long as these jobs do not use up too much of the memory of the machine. Later on we will discuss how to establish memory usage limitations for user processes, which it is a good idea to implement in the servers as well.

5.1.1 Adding a Few Good Parts

Most of the criteria for the choice of the basic parts common to compute nodes and remote-boot terminals are the same in either case, so the reader is referred to Section 4.1 for a complete description of the criteria for each one of these basic parts. Let us only review them quickly here, taking the opportunity to comment on a few adjustments that should be considered. Following this review we will discuss in detail the additional parts which are specific to remote-boot terminals.

Choice of motherboard: the motherboard should be a single-processor one, and can be exactly the same used on a compute node; an on-board sound chip may be actually welcome in this case, if it is of good quality.

The motherboard should, of course, match the CPU and RAM chosen for the terminal. For older machines that you may want to re-use as terminals, this means that you should have the complete set of older parts, including the motherboard, the CPU and the RAM modules. It may be difficult and expensive to upgrade the memory of such older motherboards to the standards needed for a remote-boot terminal, because the appropriate type of memory modules may not be in production anymore. Sometimes you can get around this problem by canibalization, making one useful machine out of two or more older ones. If you cannot get sufficient memory to run without swap space, configuring a swap partition on a local IDE disk may be a reasonable way out.

Choice of processor: the CPU can also be the same, but may be a much slower one, allowing for the possibility of the re-utilization of older machines which would have no other useful purpose.

The minimum CPU that can be recommended for use on a terminal is a 200 MHz Pentium MMX, which will do just fine for most common uses, such as editing text and programs, dealing with electronic mail and surfing the web. Absolutely *any* CPU of the i386 architecture which is for sale today can be used and will in fact have many CPU cycles to spare, opening up the possibility of a double role for the machine, as a terminal and as a compute node. Therefore, if you are going to buy new machines for the terminals, then you can safely go for the cheaper and slower CPU's available, unless you have very special needs or want to have terminals doubling as compute nodes.

One issue which is relevant for terminals is silent operation, since they are going to be very close to the users. The major source of noise on a diskless machine are the cooling fans on the power supply, the cabinet and the CPU. A slower CPU may have the advantage in this respect, since it will probably need only a smaller, slower and therefore less noisy fan. In some cases it may be possible to install in such CPU's a larger heat sink, thus avoiding the need for any CPU fan at all.

Choice of memory: the type and amount of memory on a remote-boot terminal may also be the same as in a compute node, but the minimum necessary amount is actually a little smaller.

You can have a terminal running quite nicely with only 128 MB or RAM, even with a high-resolution X11 monitor. It is even possible to run one with 64 MB, but then you may be on the borderline and some performance degradation may occasionally set in. Since memory in such amounts is not very expensive these days, it is a good idea to adopt 256 MB as the standard amount, specially if the terminals are to double as compute nodes.

If you have 128 MB or more on the terminals, it should not be necessary to configure swap for them. A swap partition can be configured on a local disk, of course, with little impact on the hardware and software maintenance issues. This may be a good way to re-use old machines which have only 64 MB or possibly only 32 MB of RAM. Although it is possible to do swap over the network, as will be discussed later on, you should not make use of this possibility unless you have some special reason to do so. It is always better to get more RAM, if it is at all possible.

Choice of network card: the network card for a compute node should be exactly the same as for a compute node; a remote-boot terminal will work very well with a good-quality 100 Mbps card; a 1 Gbps card will probably render the network completely transparent to the user, almost invisible, in fact.

Choice of power supply: this can also be the same as for a compute node; if you are going to have many disks and I/O unit on a terminal, a slightly larger power supply may be in order, but most of the time a standard one will do just fine.

This is another part for which the noise issue is relevant. In order to build a silent terminal you will need to install a special power supply with noiseless fans, possibly with controlled rotational speed. However, this may be quite expensive, for such a high-end power supply may cost as much as 5 times more than a run-of-the-mill model.

Choice of cabinet: as was mentioned before, a standard computer cabinet is a necessity rather than an option for a terminal, but it also can be the same midi-tower which was described in Section 4.1 for a compute node; in the case of a terminal you may use the front bays for the floppy drive, as well as for CD writers, DVD readers and other I/O devices, of course.

The concerns with the size of the cabinet and with ventilation issues are the same, including the installation of extra 3-inch fans if necessary. However, here me must recall that these extra fans are also a source of noise. If you do not have any hard disks installed in the terminal and the processor is not a particularly hot one, it may be possible to leave these extra fans out in order to obtain a more silent terminal.

This completes the discussion of the parts common to compute nodes and remote-boot terminals. Before we go on to describe the requirements for the monitor and the video card, we must establish some basic concepts relating to the video hardware. Unlike the case of the server, in which the video monitor will be used only in text mode and then only for a few special system administration tasks, the monitor of a terminal will be used constantly and mostly in graphics mode. Therefore, in this case we must consider more thoroughly the characteristics of the hardware. The purpose of a graphical video system is to deliver an image to the user. In this day and age color is an absolute necessity, so we will discuss only color monitors. There are three basic characteristics of the image which are directly relevant to the user and hence to our efforts at establishing the required characteristics of the hardware.

The first is the *logical size* of the image, which is usually a rectangle with a certain number of dots or *pixels* in the horizontal and vertical directions. This is called a graphics *mode* and is sometimes referred to, in a rather loose way, as the resolution, as we shall do here, but it is not really a resolution in the strict sense of the word. A typical example of such a graphics mode is 1024×768, meaning that the image consists of 1024 pixels in the horizontal direction and of 768 pixels in the vertical direction. About a thousand pixels in either direction, leading to about a million pixels overall, are the typical orders of magnitude for most modern monitors. You will note that the numbers above are in a 4×3 ratio, called the *aspect ratio*. This four by three aspect ratio is also typical of most modern monitors, so all the modes we will discuss here have this same aspect ratio. It is important to use only modes with the correct aspect ratio. The use of modes with

other aspect ratios, such as the 5×4 ratio common in older large-size monitors used in Unix workstations, will result in distorted images, in which a circle will appear to be an ellipse on the screen, unless one calibrates the monitor to use only a portion of the available screen area. These are various standard and popular 4×3 modes in common use these days: 1024×768, 1280×960, and 1600×1200. Older PC hardware used to support modes with lower resolutions than these. If you divide the numbers by 2 you may get some of the familiar older modes: 512×384, 640×480 and 800×600.

The second concept we must examine is the *color depth*, which is expressed as a certain number of bits and relates to the number of different colors that may be represented and hence that may be shown at the same time on the monitor. For example, a strictly black and white video system has a color depth of 1 bit, for with a single bit one can represent just two numbers in base 2, the numbers 0 and 1, which can be used to represent the two possible colors, black and white. The smallest common color depth for color monitors is 8-bit color (or 8 bpp, for bits per pixel), in which one can represent up to 256 colors, since a 8-bit binary number can have 256 different values, from 0 to 255. Other common color depths are 16-bit, 24-bit and 32-bit. With 16 bpp you can have up to about 65 thousand colors, and with 24 bpp up to about 16 million colors. Usually the 32-bit color depth is just another implementation of the 24-bit color depth and does not really represent a higher color capacity. A 16-bit color depth already gives you a lot of color, but note that, since a monitor has around a million or a few million pixels, with the 24-bit color depth you may have a different color in every single pixel of even the largest available monitor, which is what one may qualify as an infinitely rich image for all practical purposes.

The 24 bpp color depth is an ideal one because it matches nicely the way in which color is represented within the operating system. The usual representation of color in Linux is by the RGB system, in which the color is represented by three components, (red, green and blue), each one with an intensity given by an 8-bit integer, varying therefore from 0 (no color) to 255 (fully saturated color). The 24-bit color depth has just the three bytes that it takes to represent the intensities of the three basic colors of the RGB system. The RGB system is also the one used by the hardware, since all modern cathode ray tubes used in monitors do have in fact three electron beams lighting up small phosphorescent pixels with exactly these three colors. By varying the intensity of these three basic colors all other colors can be generated. The RGB representation of a color consists therefore of three numbers, giving the intensity of the red, green and blue beams. For example, (0,0,0) represents black, or zero intensity for all three basic colors, while (255,255,255) represents white, with maximum intensity for all three basic colors. Usually these numbers are represented in hexadecimal format, in which 255 becomes `FF`, using two digits per number and writing them all together in a single block, so that black is represented by `000000` and white by `FFFFFF`. Pure and intense red, green and blue are represented by `FF0000`, `00FF00` and `0000FF`, respectively.

The third concept we must discuss is the *refresh rate* of the image, or the number

of times it will be redrawn per second. Every monitor has a certain degree of image persistence in its phosphorescent layer, meaning that there is a certain amount of afterglow even after the electronic beam is turned off. This is necessary because the beam must scan all pixels in succession, and hence each one must have some persistence of glow because there will be some time until the beam can return to it and make it glow again. Therefore, you may think of the image on the screen as one which is being reinforced a certain number of times each second, and which may change from scan to scan in order to produce the illusion of movement on the screen. Since one expects the screen to be able to represent movement well, the amount of image persistence cannot be too large or it will cause a slurring of the movement in images which change fast. Given a limited amount of image persistence, if the rate in which the image is refreshed is too slow, the phenomenon of *flickering* may set in. If this happens the user feels that the image is flashing quite fast in his eyes, specially the brighter light-colored regions of it. The amount of sensitivity to flicker varies from person to person, but it is hard on the eyes and very tiresome, so it must be carefully avoided.

The concepts of logical size, color depth and refresh rate are all the characteristics of the image which matter directly to the user. We must, however, go a little deeper into this discussion in order to be able to discriminate the hardware. There are in fact three frequencies which are involved in producing the image. The first is the aforementioned refresh rate, which is typically of the order of several tens of Hertz. Since the image is updated line by line along the horizontal lines, you may think of an image update as a horizontal line which scans the screen vertically with this frequency, which for this reason is called the *vertical refresh rate*. Now, during a refresh of the complete image each one of its lines is scanned horizontally by the electron beam, and the number of pixels in the vertical direction is also the number of horizontal lines which must be scanned during a refresh. We see therefore that the electron beam must scan the horizontal lines at a much higher rate, given by the product of the vertical refresh rate by the number of pixels in the vertical direction. This is the *horizontal scanning rate*, also called the *horizontal synchronism rate*, which is of the order of several tens of kHz, and which gives how many horizontal lines are drawn per second.

There is yet one more frequency to consider: as every horizontal line is scanned, every one of its pixels is scanned, and for every pixel the information about its color intensities must be implemented, by means of the variation of the intensity of the beam. We see therefore that the intensity of the beam must be modulated with a very high frequency, given by the product of the horizontal scanning frequency and the number of pixels along the horizontal direction. This is the *pixel rate* or *pixel clock* and may be of the order of 100 MHz or more. While the first two frequencies are a concern only for the cathode ray tube of the monitor, this last one is also the frequency with which the information contained in the image is passed from the video card to the monitor, so that it is of relevance both for the monitor and for the video card. In order that an image with a given set of characteristics be produced, the hardware must be able to deal with

all three frequencies which are implied by the choice of image.

Choice of monitor: of the three characteristics of the image, only the logical size and the refresh rate affect the choice of monitor; since the monitor is an analogic device when it comes to color intensity modulation, the color depth is of no concern to it and will affect only the choice of video card.

Common physical sizes of monitors vary from 14 inch to 24 inch, but the prices go up very steeply at the high end of the range. The smaller size you should consider using is a 15-inch monitor, but in order to ensure reasonable comfort for the user it is better to adopt a 17-inch monitor as the standard for remote-boot terminals, preferably a good-quality flat-screen monitor. This size of monitor is currently a very common one and can be obtained for very reasonable prices. At this time 19-inch monitors are just starting to get affordable, while 21-inch or large units are still very expensive. We are, of course, talking only about traditional CRT (Cathode-Ray Tube) monitors, all other technologies currently available, such as LCD (Liquid Crystal Display) and plasma monitors, are still much more expensive, probably too expensive to consider except in very special cases.

The graphics mode that you choose to use should match the physical size of the monitor. All modern monitors have an actual physical resolution of about 100 dpi (dots per inch), so the larger monitors can handle larger sets of pixels and you can configure larger logical image sizes for them. The physical resolution of monitors is usually reported as the "dot pitch", or the spacing between dots in the screen, in millimeters. A resolution of 100 dpi corresponds to a dot pitch of 0.254 mm. Common values of this quantity go from 0.28 mm for lower-end monitors to 0.24 mm for higher-end ones. Note that all monitors, even the largest and highest-resolution ones, are in fact low resolution devices if compared to printers and scanners. Even the simplest printers and scanners can print or acquire images at about 300 dpi, and the better ones can easily go as high as some 1400 dpi. Most graphical software under Linux include resolution mapping and zooming capabilities in order to circumvent this limitation, of course. The need to match the graphics mode to the size of the monitor in order not to go much over 100 dpi comes also from a purely visual criterion, given the text fonts which are available: if a high-resolution mode is used on a small monitor, leading to an unreasonably large physical resolution, then all the text fonts will be proportionally smaller in the screen, to the point where it may be difficult to read things even with the largest fonts available. If, in addition to this, the resolution of the mode is considerably larger than the physical resolution of the monitor, there will be image degradation of the fonts, further compounding this readability problem.

In Table 5.1 you will find most common monitor sizes as well as the modes recommended for each one of them, all having approximately 100 dpi of actual physical resolution. These are maximum values for the numbers of pixels in the modes, you will be able to use also all the other lower-resolution modes provided by the X server, right down to a

Nominal size (in inches)	Numbers of pixels horiz.	×	vert.	Resolution (in dpi)
14	1024	×	768	91.4
15	1152	×	864	96.0
17	1280	×	960	94.1
19	1440	×	1080	94.7
21	1600	×	1200	95.2
24	1856	×	1392	96.7
24	1920	×	1440	100.0

Table 5.1: Typical monitor sizes and the corresponding recommended maximum mode sizes. The physical resolution of each combination is also given.

very low 320×240. The built-in 4×3 form factor modes available without manual configuration in version 4.2.1 of the XFree86 server are: 1920×1440, 1856×1392, 1792×1344, 1600×1200, 1400×1050, 1280×960, 1152×864, 1024×768, 960×720, 928×696, 896×672, 800×600, 700×525, 640×480, 512×384, 400×300 and 320×240. Some low-resolution modes are available as double-scan modes, for use in the larger monitors, when supported by the video hardware. This may be necessary because these modes need relatively low horizontal scanning frequencies and the monitors also have, besides a maximum, also a minimum possible for such frequencies. The double-scan modes improve the quality of the image and require doubled scanning frequencies, which makes them usable in the larger monitors. Other modes or variations of these modes may be included by hand in the "Monitor" section of the configuration file of the server, which is located at `/etc/X11/XF86Config-4`. This is the case for the 1440×1080 mode which we recommend for 19-inch monitors. You will find appropriate mode lines for these modes in the subdirectory `trm/00/etc/X11/` of the resources area, in the file `XFree86-1440x1080-mode-lines`.

Note that all entries in Table 5.1 except the first (for 14-inch monitors) and the last (a very large mode for huge 24-inch monitors) have physical resolutions very close to 95 dpi. Since not all the nominal area of the physical screen is actually used for the image, the image is in fact a bit smaller and hence the actual resolution is a bit larger, very close to 100 dpi. For the calculation of these physical resolutions you must remember that the nominal size of monitors is measured along the *diagonal* of the screen, and that in a 4×3 form-factor screen the diagonal, the horizontal and the vertical sizes are in 5×4×3 ratios. If you use the fact that the horizontal size of the screen is therefore 4/5 of the nominal size and consider the number of pixels along the horizontal, you will arrive at the numbers shown for the resolution. The same calculation can be done using the fact that the vertical size is 3/5 of the nominal size and the number of pixels along the vertical direction, of course.

Although Table 5.1 lists the largest modes you should use in each monitor, you may

want to use smaller modes even on a large monitor, of course. The appropriate mode will depend also on the kind of use intended for the terminal, and there are cases in which a smaller mode may be best. Users with a significant amount of vision impairment may be one such case, since they will probably be better off if the text fonts are significantly larger in the screen. Some types of use require lower resolution. For example, many computer games are characterized by intense action, leading to high-speed video output in frames per second (fps). They work better with a large vertical refresh rate, but usually require low resolutions. Even a 800×600 mode represents high resolution for a computer game. On the other side of the range we find CAD (computer-aided design) programs, which are characterized by low-speed video output, but require very large modes in a large monitor. A CAD program is best used in a huge 24-inch monitor with a large mode close to 2000×1500. Other common uses which typically require lower resolutions are DVD viewing, which can be done best with a 800×600 mode, and compressed AVI movie file viewing, which may require as little as 640×480, which is close to the resolution of traditional broadcast television. For most of the more routine types of utilization of the system one can use quite comfortably the modes listed in Table 5.1 for each size of monitor.

Having chosen the size of the monitor and the largest mode to be used, one must now consider the electrical characteristics that the monitor should have in order to support this mode. However, the first one that we must discuss is the vertical refresh rate, which is to be chosen based purely on visual comfort criteria. The minimum vertical refresh rate acceptable in order to ensure good visual comfort is 70 Hz. Many people will be bothered by flickering on a monitor with 60 Hz refresh rate, which should be avoided. If possible, use a 75 Hz refresh rate to be well on the safe side, but 70 Hz is in fact a safe minimum. For activities requiring fast action on a low-resolution mode a higher refresh rate such as 85 Hz or even 100 Hz may be better, but these will probably be available automatically, so long you make sure that you have at least 70 Hz for the largest mode recommended for your chosen monitor. Virtually any monitor, new or old, will support a vertical refresh rate of 70 Hz. The typical range for a 15-inch monitor is 50–100 Hz, and for 17-inch and 19-inch monitors one can expect 50–160 Hz, so having support for a good vertical refresh rate on the monitor is not usually a problem.

The main electrical characteristic to watch, in order to be able to run on the monitor the X11 modes recommended here, is the maximum horizontal scanning frequency supported by the monitor. This is usually the most demanding characteristic for a monitor, and the one you should pay most attention to. Most modern monitors will detect when an input horizontal scanning frequency is out of their range and will cut off the input, showing an out-of-frequency error message instead, but *be warned* that older ones may actually burn out if subject to too high a horizontal frequency for any sizable amount of time. As we discussed before, the horizontal frequency should be the product of the vertical frequency and the number of horizontal lines of the mode, but in practice it must be a little higher than that due to the way in which the scanning circuitry of

the monitor works. This frequency overhead is caused by the fact that the beam must over-scan the size of the image a little bit, actually spending a bit of time outside of it. If we represent the horizontal scanning frequency by f_H, then you can calculate the minimum value that it must have by means of the empirical formula

$$f_H = C\, n_V f_V,$$

where f_V is the vertical refresh frequency and n_V the number of pixels in the vertical direction. The constant C represents the frequency overhead and varies from 1.04 to 1.06 for all modes commonly used, $C = 1.05$ being a good average value for most of them. You may want to use $C = 1.1$ to play it safe, but this may make you miss a mode that your monitor actually does support. Therefore, if you are using the formula in order to orient the acquisition of a new monitor, use 1.06 or 1.1 to be on the safe side, but if you are trying to determine what an existing monitor can do, use 1.04 and then just test the resulting modes on it. You can get more detailed information about the default modes supported by the X server, including the sizes, the refresh frequency in Hz and the necessary scanning frequency in kHz, in the file

`/usr/share/doc/xserver-xfree86/examples/vesamodes.gz`

which is part of the documentation of the X server and lists the standard VESA modes which are supported by the server. Note, however, that not all modes mentioned here are included in this list. For the extra modes mentioned here this information can be found within the files in the resources area which contain the definitions of these extra modes for the X server. Using the formula above or the information contained in these various files you should be able to determine what is the horizontal scanning frequency that you need for your monitor. Many old 14-inch monitors may support only a certain fixed discrete set of horizontal synchronism frequencies, but most modern monitors have what is called "multi-sync" capability, meaning that they can work with any horizontal frequency in a certain range. The typical range of values for a 15-inch monitor is 30–65 kHz, for a 17-inch monitor it is 30–70 kHz and for a 19-inch monitor one can expect something like 30–90 kHz or even better.

As a very typical example, a 17-inch monitor must support a horizontal synchronism frequency of 70 kHz in order to be able to operate with a 1280×960 mode under a 70 Hz vertical refresh rate, which are our recommendations here for monitors of this size. This mode is not, however, part of the standard set of modes which is built in the X server, so you will need to add some mode lines to the end of the "Monitor" section of the X server configuration file `/etc/X11/XF86Config-4`. You will find appropriate mode lines for 1280×960 modes in the subdirectory `trm/00/etc/X11/` of the resources area, in the file `XFree86-1280x960-mode-lines`, including one with 70 Hz refresh which has been fine-tuned and tested to work in monitors with a maximum horizontal scanning frequency of 70 kHz, such as the ones which will be mentioned below in our example of hardware listing for a minimal remote-boot terminal.

In the case of a 19-inch monitor, the situation is typically quite a bit more comfortable, since you will be just fine with a machine supporting a 80 kHz horizontal scanning frequency, which should be easy to get in a monitor of this size. With this frequency you will be able to run the recommended mode of 1440×1080 with a 70 Hz refresh. With a 85 kHz horizontal scanning rate, which is also not difficult to get, you will be able to go to a 76 Hz refresh rate. Many 19-inch monitors support horizontal scanning rates of up to 96 kHz, which will allow you to use a vertical refresh of 85 Hz. As was said before, the 1440×1080 mode lines are available in the resources area, in the file `XFree86-1440x1080-mode-lines`. Usually it is not difficult to make a 15-inch monitor work with a 1152×864 mode, but if the 75 Hz standard VESA mode does not work for you, a few extra 1152×864 modes are available in the resources area, in the file `XFree86-1152x864-mode-lines`, including a mode to work at 70 Hz vertical refresh, with a horizontal scanning rate under 65 kHz.

The last electrical characteristic which is relevant for monitors is the pixel clock frequency. As we discussed before, it should be the product of the horizontal scanning frequency and the number of pixels along the horizontal direction. Since the horizontal scanning frequency is itself the product of the refresh frequency and the number of pixels along the vertical direction, the pixel clock frequency can be reduced to the product of the refresh frequency and the total number of pixels in the mode. Here too there is a frequency overhead due to the technicalities of the operation of the monitor and of the video card, and this time it is considerably larger. If f_P is the pixel clock frequency, then we can calculate the minimum value it must have using the empirical formula

$$f_P = 1.5\, n_H n_V f_V,$$

where f_V is the vertical refresh frequency, n_V the number of pixels in the vertical direction and n_H the number of pixels in the horizontal direction. The multiplicative constant 1.5 represents the frequency overhead. The value of f_P for each mode is also available in the documentation of the X server for the standard VESA modes, as well as in the files with the extra modes in the resources area. It is the third entry from the left to the right in the mode line which defines the mode, the first number after the name of the mode, and is given in MHz. For the largest resolutions recommended here for each size of monitor, and using a 70 Hz refresh rate, the pixel clock rate for a 15-inch monitor is about 92 MHz, for a 17-inch monitor it is about 120 MHz and for a 19-inch monitor about 150 MHz. Since typically a 15-inch monitor should be able to handle a pixel clock frequency of 100 MHz, and 17-inch or 19-inch monitors should accept frequencies of up to 150 or 200 MHz, the pixel clock is not usually a problem for the choice of monitors, hence not a crucial criterion of choice.

In point of fact, while the allowed ranges for the vertical refresh rate and the horizontal scanning rate are typically readily available in the hardware documentation of the monitors, this is not so for the pixel clock rate, which is not usually available. You may, however, orient yourself by the supported modes reported in the documentation of the

monitor, which will usually contain the supported mode sizes and refresh rates. Using that information and the formula above, you will be able to get lower bounds on the maximum supported pixel clock frequency. On very recent monitor models the X server may be able to query the monitor directly for this information, among several others. It is interesting to note that when it happens that the pixel rate in use is somewhat above the nominal value reported by the monitor in this way, the X server reports this as a warning and not as an error, and the monitor seems to work perfectly, notwithstanding the mismatch. Monitors have been observed to work flawlessly for long period of time with pixel clock frequencies up to about 10 % above the nominal value reported by the monitor.

Choice of video card: all the three main characteristics of the image will influence the choice of video card; while the combination of the logical size with the color depth chosen will determine the amount of video memory needed, the combination of the logical size with the refresh rate will determine the pixel clock frequency, which must be supported by the card as well as by the monitor; however, there is an even more important issue here, besides these technical characteristics, the issue of compatibility with free software.

In order for the operation of the X11 system to be possible on the video card you choose, it must be supported by the X server, of course. There is also the issue of support in the Linux kernel, which may be needed in some circumstances. In order to discuss this issue we must first separate it in two parts, because video cards can be used in either text mode or graphical mode. While the text mode operates according to a very well known and widely used standard, the graphical mode usually operated is a way which is very specific to each card, depending strongly on the video processing chip it uses. Due to this, virtually any video card will work in Linux in text mode, without any need to worry about software support. While this is quite sufficient for servers, terminals need, of course, to operate the card in graphics mode, and that is a completely different story. Traditionally the graphics mode support has not been in the Linux kernel but rather in the X server, which is something like the kernel of the X11 subsystem. However, now there is the possibility of having the graphics support in the Linux kernel as well, at least for some cards.

The tradition of having the graphics mode supported by the X server rather than by the kernel is a consequence of the history of the development of the pieces of free software which are involved, and is an exception to one of the fundamental roles of the Linux kernel, which is to control directly and completely all the hardware existing on the machine, presenting to the upper layers of software a set of structures which are abstractions of the actual hardware, by means of which these other layers of software can make use of the hardware through the intermediation of the kernel. This exception means that a user-space program, the X server, will have direct control over a certain part of the hardware, the video card, and that fact can introduce an additional source

of instability into the system, for it is possible that a bug in the X server will make the machine hang without the kernel having any control over or relation to the event. This is, in fact, one reason why we do not run the X11 subsystem on central servers.

The current shift of introducing support for the graphics mode of video cards in the kernel can be seen as a return to the original role of the kernel in these matters, and is a very good thing, although it is still not completely developed. The abstract structure representing the video card which is presented by the kernel to the upper layers of software is called the *frame buffer*. There is already a driver for the frame buffer in the X server, called `fbdev`, and a fair number of cards is supported by the kernel, but most drivers are still in an experimental stage in the 2.4 kernel and using the frame buffer device for the X server may not perform as well as the direct driver in the X server. The situation seems to have improved greatly in the 2.6 version of the kernel, in which most drivers are not marked as experimental anymore. Since the 2.6 kernel has been released not so long ago and is not yet in very wide use, it currently seems too soon to recommend the use of the frame buffer as the default alternative for the X11 subsystem, although this will probably be the best alternative on the long term.

Some accelerated drivers for OpenGL-capable video cards may require an additional kernel module in order to operate, even if you use a specific driver for them in the X server, which shows that the kernel is the natural place to have the hardware drivers for video cards. The possible installation complications caused by this kind of thing should vanish once all the video cards are fully supported by the kernel, but there may be some serious legal difficulties involved in this, because some of these drivers are proprietary, non-open-source software, and the detailed technical specifications of the corresponding video processors, needed in order that open-source drivers can be written, may not be publicly provided by card manufacturers. Such video processors and cards will, of course, have to stay out of the kernel until their manufacturers agree to release publicly and completely their detailed technical specifications. Sometimes an open-source graphical driver exists, but does not support the full amount of OpenGL acceleration that the card is capable of.

One must also keep in mind that the market of video cards is very active, new cards and video processors seem to be released all the time, with ever-increasing capabilities. Since many manufacturers will not release publicly available drivers for their new hardware, either for the Linux kernel or for the XFree86 server, even if they do release sufficient technical information, the development of open-software drivers may lag behind the release of new cards and there may be a waiting time before an appropriate driver becomes available. Also, the released open-source drivers may not support all the capabilities of the new cards and may not explore fully their acceleration capabilities, specially if there is not much cooperation of the manufacturers with the release of the relevant technical information, a situation which unfortunately is all too common.

The upshot of all this is that you should make sure that the video card you get is well supported in the XFree86 project. There are two versions of the X server currently

in use (in the "woody" version of the Debian distribution), the older one is version 3.3.6, which is no longer in development, and the newer one is major version 4, which is in active development, and which includes a major re-design of the server. Its development is currently in version 4.4.0. The reason why both versions are still in use is that not all cards supported by the older version of the server are already supported by the new version, because the drivers of the old architecture of the server must be ported to the new one, if they are to be used in the new, re-designed version of the software. Since the older version of the X server seems not to be available in the "sarge" distribution, if you find that you need it in order to drive some older video card that you might want to use, you may have to use the packages from the "woody" distribution. There should be no problem in installing packages of the older distribution in the new one. Information about the supported cards can be obtained in the documentation of the version 4 of the X server, which is in the directory

```
/usr/share/doc/xserver-xfree86/
```

Look for example at the file `Status.gz`, using the `zless` command. The documentation of the version 3 of the server is in the directory

```
/usr/share/doc/xserver-common-v3/
```

This is so because, unlike the single package for the new X server, which is in fact a single server for all the supported cards, there are several packages with several different versions of the older server, for video cards from various different manufacturers. You will have to read the documentation in order to determine which of them is the appropriate one for your card. You may also want to browse the pages of the XFree86 project, which is located at the URL

```
http://www.xfree86.org
```

If you decide to try using version 2.6 of the kernel, you may want to check out the "Graphics support" menu in the window of the `make xconfig` configuration utility, under which you will find the frame buffer support for about 20 types of card. If your card is included, you may want to play around with using the `fbdev` driver of the version 4 of the X server, to see if it works well enough for you.

Let us now go back to the discussion of the graphics characteristics. While in ancient times we seemed to be happy to have a simple black and white monitor, so long as it was fairly large and had a resolution of 75 dpi, which was considered quite reasonable at the time, and later dreamed of getting a grayscale monitor with a 4-bit color depth, capable of displaying 16 shades of gray from black to white, with the advent of the World Wide Web color became an imperious necessity. At first one was happy to have an 8-bit color monitor, but that quickly showed its severe limitations. Many older Unix workstations with large and expensive 21-inch color monitors are useless today as video terminals

because they have video hardware limited to 8 bpp and no easy (and affordable) way to change that. Hail to the world of open-standards PC hardware technology, in which upgrades such as this are always possible, one way or another.

An 8 bpp video subsystem just has too few colors for a contemporary informatics environment, where the Internet is a constant presence, and will lead to problems with the exhaustion of space in color maps. Some applications such as browsers may feel compelled to load their own color maps every time the focus of a desktop session is shifted to them, causing a general upheaval in the image on the monitor. Parts of the windows of other applications may become invisible altogether, due to the way the server will round off the color values in order to fit them into a limited color space. The minimum acceptable color depth is 16 bpp, which will avoid all such problems and should be quite sufficient in all but the most demanding environments. Since these days the difference in investment between a 16 bpp video setup and a 24 bpp video setup is not very large, one may consider adopting 24 bpp as the standard for the workgroup, and forget for ever after about any colorspace limitations.

The amount of video RAM needed depends on the size of the mode and on the color depth due to the necessity of storing the corresponding video information in the video memory on-board the video card. Every pixel will correspond to a memory location, with a size given by the number of bytes needed for the color depth. Therefore, it would seem that in order to store the image one needs a number of bytes equal to the total number of pixels in the mode times the number of bytes per pixel needed for the given color depth. In fact, there is an overhead here too. This memory overhead is, in percentile terms, of the same order of magnitude as the frequency overhead for the pixel clock rate. If the minimum amount of video memory needed for a mode is represented by M_V, then it can be calculated in MB by means of the empirical formula

$$M_V = C\ n_H n_V B_{pp}/(1024)^2,$$

where, just as before, n_V the number of pixels in the vertical direction and n_H the number of pixels in the horizontal direction, while B_{pp} is the number of bytes per pixel needed for the color depth, that is, 2 for 16 bpp and 3 for 24 bpp. The constant C is the memory overhead factor, and has values from 1.3 to 1.5 depending on the mode, $C = 1.4$ being a mean value which is valid for many usual modes. If you use this formula to decide on the purchase of a new card, use $C = 1.5$ to be on the safe side. If you are trying to figure out what older existing cards are capable off, try out modes even if they do not quite satisfy this condition, because you may get lucky with the combination of the driver for the card and the mode you chose.

There is another reason, besides the memory overhead, why one may need more video RAM than indicated by the calculation relating to the video information in the image. Older video cards used to do just the simple low-level operation of painting pixels, so that the space for storing the image in the memory was all that was needed. More recent cards, however, can do much more than simply to light up pixels individually one by

one. They are able to receive and execute without further instructions from the CPU higher-level operations such as to draw a line or some other geometrical object from a point to another point, or to fill a whole region with a color, by means of routines which are hard-coded into the video chip, which is really just a dedicated processor. Since this is much faster to execute than the detailed low-level instructions of painting each pixel with a certain color, this kind of high-level instruction can cause a considerable acceleration of the graphic operations. The most recent cards can even to 3-dimensional operations relating to the representation of objects by means of perspective, involving operations of an even higher level. Cards of this type are said to support the OpenGL (Open Graphics Language) system in hardware, or to have either 2D of 3D acceleration capabilities.

Cards which are capable of doing these kinds of things may want to use video RAM for things other than raw image information. They might need memory space for other data or for instructions relating to 2D or 3D graphics acceleration. For such cards it might be a good idea to double the result of the memory calculation with the formula above, or even more than double it. Usually this is not a problem, since most modern cards supporting OpenGL acceleration in the hardware will only be available with large quantities of RAM anyway. In fact, it is almost impossible to get a video card with less than 32 MB of video RAM these days, and cards with 64 MB are already quite common. This is more memory than one used to be able to have on the machine itself, not so long ago. Add to this the fact that virtually all AGP cards available today do support hardware acceleration, and you will see that video memory should not be a problem unless you are trying to re-use some old video cards.

Let us give a few examples of possible uses for various typical amounts of video memory. If you have a card with 2 MB, the best you can hope for is a 15-inch monitor with the 1152×864 mode and 16 bpp color, but you might actually find yourself limited to the 1024×768 mode. You may be able to use 24 bpp with a 800×600 mode, but it is also borderline. So you can see you will not be very happy with only 2 MB of video RAM. The first common amount with some chances of happiness is 4 MB. With this you may be able to have 24 bpp with 1152×864 and will certainly be able to have a 17-inch monitor with 1280×960 and 16 bpp. You see, therefore, that 4 MB is the absolute minimum you should consider using. A card with 8 MB represents a great improvement and will take you quite far up the scale of modes and monitors. Not only you will be able to have 24 bpp and 1280×960 on a 17-inch monitor, but also the 19-inch monitor with the 1440×1080 and 24 bpp will be within reach. It will even be enough for a 21-inch monitor with 1600×1200 and 16 bpp. This makes 8 MB quite sufficient for most people. To go above this you need 16 MB, which will give you the 21-inch monitor with 1600×1200 and 24 bpp, as well as either 1856×1392 or 1920×1440 with 24 bpp on a huge 24-inch monitor. We see, therefore, that in terms of the size and color depth of the image 16 MB is as much as one may need, for any size of monitor. If you add OpenGL support and double these figures, we conclude that a 16 MB card

will satisfy most people and that 32 MB will satisfy even the more demanding users in their graphics needs. Since 32-MB cards are common these days and 64-MB cards not difficult to get, we see that indeed video RAM is not going to be an important limitation on new machines.

Possibly the weakest spot on many cards is the maximum pixel clock rate that they support. Given the large amounts of video memory currently available in almost all cards, it may be the characteristic that most severely restricts what the card can do in terms of image size and refresh rate. The maximum pixel clock rate of a card may depend on the color depth used, since cards must be able to read very fast one or more bytes per pixel in order to produce the video modulation signal for each one of the three basic colors. This information is seldom easily available when you purchase a card, and if it is available it may omit the dependency on the color depth, so that the nominal value you get may be larger than the actual value for a 16 bpp of 24 bpp color depth. The driver for the card within the X server does know all about this, of course, and it will write this information, among many others, to its log file `/var/log/XFree86.0.log`.

Just as in the case of the monitors, in this case too the easier way to determine minimum bounds on what a card can do may be to work back from the supported video modes, vertical refresh rates and color depths, which are usually reported in the hardware documentation of the cards. Using the formula given before relating the pixel frequency to the size of the video mode and the vertical refresh rate, one may get approximate but useful information on the pixel rates supported by the card. If one relates this information to the color depth reportedly supported by that mode in the hardware documentation, then one establishes that the pixel rate obtained in this way is supported at least for that color depth.

Sometimes this maximum pixel frequency is reported as the "RAMDAC" frequency, after the piece of hardware involved in the modulation of the video signal. The first part of the name of this device tells us that it has something to do with video memory (RAM), and the second (DAC) is an acronym for digital to analogic converter, which describes its function: to read the digital image information from video RAM and to convert it into modulated analogic signals to be sent to the monitor. Sometimes this RAMDAC is a separate chip in the video card, mostly in some older cards, but most commonly it is integrated into the video processor chip. The pixel frequency should not be mixed up with a certain "video bandwidth", usually reported as a certain number of bits, which may show up in the hardware documentation of the card. This video bandwidth relates to the video bus, the communication channel between the video processor and video memory within the card and is only indirectly related to the maximum available pixel clock rate.

As examples of commonly available pixel rates, we may mention that common values of the maximum pixel rate for typical 4 MB cards of brands such as S3, SiS, Rendition and Verite are in the 170–175 MHz range. Typical 8 MB cards of brands such as ATI, SiS and Matrox have maximum pixel rate in the 230–250 MHz range. Cards with 16 MB

such as those of the 3Dfx brand may support up to 300 MHz, and cards with 32 MB to 64 MB such as those of the nVidia brand may go up to some 350 MHz. All these maximum pixel rates are quite sufficient for the modes recommended here, considering the corresponding amounts of memory needed by them.

One important point is that one should pay attention to the type of AGP video card that one gets, in relation to the motherboard one intends to install it on. As the performance of video cards increases, involving ever higher frequencies, the issue of heat dissipation on the video processor comes to the foreground. You will note that all modern high-performance video cards have a heat sink on their processor chips, some of them with small cooling fans as well. In order to allow this increase in frequency without excessive heat dissipation, the AGP standard has been changing and some motherboards will support only the most recent standard. Currently the high-performance cards use a 1.5 V voltage standard for the AGP bus, while older ones used to work with 3.3 V. Some recent motherboards will support only this new 1.5 V standard on their AGP slot, but fortunately most recent video cards are of the correct type. You may not be able, however, to use an older AGP card on such recent motherboards.

It should be noted that, due to the high frequencies traveling along the video cable, from the video card to the monitor, which exist in typical high-performance video hardware, one should *not* try to use a video switch in this case, either to have several monitors on the same machine or to share a single monitor among several machines. This is possible in the case of the servers because there are no high-resolution video modes involved in that case, but if it is tried with high-performance graphics terminals, it will in all probability ruin completely the quality of the image, or even not work at all, unless a very expensive high-performance electronic video switch is employed.

Choice of sound card: the sound subsystem is another thing we used to do without in the past and is an absolute necessity today; the typical sound card will cost somewhat less than a video card at the same general quality and performance level, so this is not a very expensive investment and it is one which is certainly worth making.

Unless you are re-using older machines as remote-boot terminals, you should use only PCI sound cards. Unless the machine is meant for a public terminal room, you should also get a pair of good-quality amplified speaker enclosures. In a public room with several machines one usually cannot use loudspeakers, and therefore users should be encouraged to have and use their own sets of headphones. A headset consisting of headphones and a microphone integrated into a single piece is a very good idea, whether or not the machine is to be installed in a public room, specially if one intends to use the system for telecommunications purposes such as voice-over-IP telephony, conference calls and video-conferencing. If possible, get a headset which has a volume control on its cable. In case one wants to have both a set of speakers and a headset, a small audio adapter box to switch from one to the other is a handy thing to have, on top or in front of the CPU cabinet, since it avoids the need to mess around with the audio connections

in the back of the cabinet.

A sound card is basically a device to convert signals from digital format into analogic format and vice-versa. When used for playing back music stored in a digital format such as WAV or MP3 files, for listening to an audio CD, or for hearing recorded conferences over the Internet, the card operates as a DAC, a digital to analogic converter, in a way which is similar to what was discussed before regarding the role of video cards. The main difference is that the frequencies involved in this case are several orders of magnitude smaller than those involved in handling video images. While typical video pixel frequencies are of the order of 100 MHz, audio frequencies are always below 50 kHz, even for very high-performance hardware. This implies also that the file sizes involved in storing digital audio are much smaller than those involved in storing digital video.

When used to record sound from some analogic source to a digital format, the card acts in exactly the opposite way, as an ADC or analogic to digital converter. This includes the digitalization of sound from the microphone in order to send voice-over-IP traffic through the network, for example. For some applications it is important that the card be able to operate in both ways at the same time, in what is called a *full-duplex* mode. Telecommunications software will usually either work better if full-duplex operation is available or actually require it in order to operate. Most modern cards do support full-duplex operation, but the older sound drivers in the Linux kernel, the so-called OSS (Open Sound System) drivers, unfortunately do not, this feature seems to be present only on the commercial version of these drivers.

Fortunately there already exists in Linux a much better and much more truly open set of drivers for sound cards, which goes by the name of ALSA (advanced Linux sound architecture). These drivers automatically support full-duplex operation on all cards, and have been integrated into the kernel source tree in version 2.6 of the kernel. Although many sound-related open-source application still use the OSS software interface, the new ALSA system includes a compatibility interface, so that these application can use the new drivers. This compatibility interface is implemented through the support of the set of OSS device nodes in `/dev/`. Hence, if your cluster of terminals is to be used for sound-oriented telecommunications through the Internet, you may consider using version 2.6 of the kernel, so that it becomes a simple matter to use the ALSA drivers.

The remaining requirements are also satisfied by most sound cards in the market today. Sound digitalization works by samplings of sound pressure level (SPL), which are in fact measurements of relative air pressure executed periodically at a certain rate. Every measurement of air pressure is transformed by the card into an integer, represented in binary form by a certain number of bits. The number of bits used for storing the integer representing the relative air pressure is important because it determines the largest dynamic range which is possible, that is, the ratio of the most intense to the least intense sounds which it is possible to represent. While for voice-over-IP applications an 8-bit representation may be quite adequate, music reproduction requires a 16-bit representation. Virtually all cards support these sound data sizes, possibly even larger

ones.

The last characteristic to be considered is the highest sampling rate supported by the card. While an 8 kHz rate may be adequate for voice applications, the high-quality reproduction of music requires much more. Sampling rates of 16 kHz and 32 kHz are common ones, but the best sound quality, such as one finds in audio CD's, requires the 44.1 kHz sampling rate which is the standard used in that type of media. You should make sure that your card does supports this 44.1 kHz sampling rate. Some cards support only a certain set of fixed discrete sampling frequencies, but mostly the supported set is large enough to satisfy the needs of most common applications. The sound software within the system may also be capable of transparently translating the data in a digital audio stream from one sampling frequency to another, in order to fit the capabilities of the sound card.

It also goes without saying that the card should have at least two independent channels, so that it may reproduce and record stereophonic sound. It should have, of course, input connections for microphone and line input, as well as a line output connection, and possibly a speaker output, all located in the metal face plate of the card. In the case of on-board sound chips these connections will be on the motherboard, of course. The card should also have a few internal connections, at least one for the analogic audio output of a CD unit, although some more recent cards do not require that anymore and may be able to capture the digital sound output of an internal IDE-interface audio CD player through its data connection to the system bus.

The remaining parts: you will also need all the usual trivialities such as power cables, a network patch cord, a PS/2 type keyboard, a 3-button mouse and a floppy drive; the keyboard may have a keycap layout appropriate for your language, but a standard US-QWERTY layout can be used for most western languages by means of appropriate software configurations; the mouse may be a serial device or a PS/2 device, which is quickly becoming the most commonly found type; the floppy drive will be used only for occasional booting purposes and may even be avoided in a well-structured environment with many machines, in which eventual booting problems can be faced in other ways. Leaving the floppy drives out may, in fact, constitute a significant security enhancement in public terminal rooms; if included at all, they should be standard 1.44 MB, 3.5-inch devices.

You may have any number of input and output units on a terminal, of many different types. These units can also be used for user data content backups. Typical such units are CD and DVD players and recorders, which may be used to listen to music or to watch movies, as well as for backup and data storage purposes. Any of these units may use either an IDE-ATA or a SCSI interface, both are well-supported. Tape devices can also be used, but this type of I/O unit is falling into disuse and is seldom seen anymore. The recommendation for I/O and backup purposes is now either a CDWR unit or a DVD player/recorder, which is better so long as it is affordable. The prices of these

units have been coming down quite fast in recent times. Other types of floppy drives such as ZIP drives of various sizes may also be used, but are also falling out of favor. One may also have scanners, printers and many other devices connected to various types of ports in a terminal, such as serial, parallel and USB ports.

Motherboard:	Soyo model SY-KT333 Dragon Lite single-processor, with 3 DDR memory slots, with a 32-bit, 33 MHz PCI bus.
Processor:	Athlon XP 1900+, with ≈1.6 GHz, with appropriate heat sink and cooling fan.
Memory:	DDR with 256 MB in a single DIMM, for a 256 MHz memory bus.
Network card:	3COM model 3C905CX or Intel model EtherExpress Pro/100, 100 Mbps, for a 32-bit, 33 MHz PCI slot.
Power supply:	run-of-the-mill ATX unit with 300 W.
Cabinet:	a midi-tower model with two independently removable side covers, with space for one or two 3-inch fans.
Monitor:	a 19 inch flat-screen unit such as the Philips model 109B, capable of 96 kHz of horizontal synchronism frequency.
Video card:	an AGP card such as the nVidia model RIVA TNT2, with 32 MB of video RAM and 350 MHz of maximum pixel clock frequency.
Sound card:	a modern PCI sound card such as the Creative Sound Blaster model "Live!", with a EMU10K1 chipset.
Keyboard:	a good-quality PS/2 model, with a standard US-QWERTY keycap layout.
Mouse:	a good-quality 3-button PS/2 mouse.
Floppy drive:	a standard 3.5-inch, 1.44 MB floppy unit; note that this unit is not strictly necessary.

Table 5.2: Discrimination of a standard high-performance remote-boot terminal.

Finally, one may also have hard disks on a terminal, which can be used for local scratch areas and possibly for a swap partition. These disks will usually be IDE-ATA devices, which are cheaper and more tolerant to imperfect environmental conditions. It is important to remember, however, that hard disks can be quite noisy, which might make it inconvenient to have them installed on a terminal. Also, intense I/O activity to a large disk on a terminal may render the terminal sluggish for other purposes, unless you use version 2.6 of the Linux kernel with the "preemptible kernel" capability turned on. If you do decide to use IDE hard drives on a terminal, make sure to compile, install and configure the IDE-PCI chipset support module of the kernel for the particular IDE chipset used in your motherboard, so that the disks may be used in the fastest Ultra-

ATA mode that they support. Otherwise, the access to the disks will default to a generic ATA mode which can be much slower than the accelerated Ultra-DMA modes used by Ultra-ATA chipsets.

This completes the discussion of the criteria which should guide the choice of the main elements of the hardware for the terminals. You will find in Table 5.2 the discrimination of a standard high-performance terminal, made with brand-new parts. Although it is also minimal, in the sense that it has the slowest CPU that you can currently purchase, it is a much more capable machine than a terminal made out of older machines, and even has the capability of doubling as a compute node. In Table 5.3 we see a typical discrimination of a truly minimalistic remote-boot terminal, built from some older machine which is to be re-used as a terminal, with reasonable improvements to the video and sound subsystems. It is important to keep in mind that these descriptions are being put together at around the beginning of the year 2005, and should be scaled or changed appropriately to fit the state of the technology at later times.

5.1.2 Assembling a Terminal

By and large, assembling the hardware of a remote-boot terminal is not all that different from assembling the server, or from assembling a compute node in a standard computer cabinet. The same guidelines, tips and warnings already given and the same cares already discussed in Section 3.2 apply equally here, so the reader is referred back to that section for the details. Let us just comment here on some of the differences one finds when assembling a terminal.

The assembly of the motherboard and of the add-on cards in the cabinet is done in the usual way, but there may be some additional cards to install, by comparison to a compute node, such as a sound card, or some other more specific things such as a video capture card, etc. None of these cards present any special difficulties. The installation of additional cooling fans on the cabinet should only be done if there is an absolute necessity of additional ventilation, since they may contribute significantly to increase the overall operating noise level of the machine.

One of the main differences is that there are no SCSI disks in this case, possibly no disks at all, which contributes to make the ventilation problems typically less serious. If there are any IDE devices they will most likely be connected to the appropriate connectors on the motherboard rather than to a separate add-on card such as the SCSI card of the server. A few short comments about the IDE controllers are in order here. Virtually all motherboards have two IDE-ATA controllers, a *primary* one and a *secondary* one, each supporting up to two IDE devices on a single cable. This gives a maximum number of 4 IDE devices in the system. A few special motherboards may have 4 IDE-ATA controllers rather than 2, and hence may have up to 8 IDE devices connected to them. Typically the second pair of IDE controllers are in fact a IDE-RAID controller, but you

Motherboard:	any single-processor motherboard with at least two or three 33 MHz PCI slots, respectively with or without an AGP slot, with memory slots for SIMM's or DIMM's, capable of handling 128 MB of RAM.
Processor:	anything from a Pentium MMX with 200 MHz to a Pentium II or Pentium III with anywhere from 300 MHz to 800 MHz, with an appropriate heat sink and cooling fan.
Memory:	either EDO RAM in SIMM's or SDRAM in DIMM's, with a total of 128 MB, for whatever memory bus exists on the motherboard.
Network card:	3COM model 3C905CX or Intel model EtherExpress Pro/100, 100 Mbps, for a 32-bit, 33 MHz PCI slot.
Power supply:	run-of-the-mill AT or ATX unit, as needed to fit the motherboard, with from 200 W to 300 W.
Cabinet:	whatever cabinet the motherboard is already assembled in, or a midi-tower model with two independently removable side covers and space for one or two 3-inch fans.
Monitor:	a 17 inch unit such as the LG models 77i, 775N or 700S, capable of at least 70 kHz of horizontal frequency.
Video card:	any PCI or AGP card, as required by the motherboard, with at least 4 MB of video RAM and at least 170 MHz of maximum pixel clock frequency, of any good-quality brand such as S3, Matrox, ATI, Rendition, Verite, etc.
Sound card:	any PCI sound card capable of full-duplex operation, of two-channel stereo operation and of 16-bit sound encoding at 44.1 kHz sampling rate, such as the Creative Sound Blaster model AWE 128 with an Ensoniq ES1370 or ES1371 chipset.
Keyboard:	a good-quality AT or PS/2 model, as required by the motherboard, with a standard US-QWERTY layout.
Mouse:	a good-quality 3-button PS/2 or serial mouse, as required by the motherboard.
Floppy drive:	a standard 3.5-inch, 1.44 MB floppy unit; note that this unit is not strictly necessary.

Table 5.3: Discrimination of a minimalistic remote-boot terminal, meant to be built out of an older machine, with some improvements to the video and sound subsystems.

may use them also as normal IDE controllers.

Each IDE-ATA controller corresponds to a 40-pin IDE connector on the motherboard. The primary controller is the one the motherboard will usually be able to boot from and its connector should be clearly identified on the motherboard. Traditionally the IDE devices are connected to the motherboard by means of a simple 40-conductor flatcable with 3 connectors, one for the motherboard and one for each IDE device. However, Ultra-ATA operation of recent devices will require the use of a 80-conductor flatcable, in order to handle the higher frequencies involved. Unlike the case of the traditional 40-conductor cables, in these new 80-conductor cables the 3 connectors are *not* interchangeable, each one has a fixed role and must be connected to the motherboard, to the *master* device of to the *slave* device in that IDE bus. The connectors should have their roles clearly marked on them.

This master/slave duality is the simple form that bus addressing assumes in the case of IDE devices. Each controller can have one master device and one slave device connected to it by means of the flatcable. All IDE devices, including disks and any I/O units such as ZIP or CD drives, must be strapped to define them as master or as slave devices. There are, however, no bus termination preoccupations in the case of IDE devices. Since the two devices connected to a single controller must share the IRQ line of that controller, the data traffic between two IDE devices will always be a bit more efficient if the two devices are connected to *different* IDE controllers. It is useful to keep this in mind when connecting several IDE devices to a system. If you will have just two devices on the machine, connect one to the primary controller and the other to the secondary controller, possibly but not necessarily with both configured as master devices.

Finally, when installing a sound card and a CD player or recorder on a machine, do not forget to install the small stereo coaxial cable which should connect the analogic audio output of CD unit to the internal connector marked as `CD-IN`, or something like that, on the sound card. Although some more recent cards do not need this, in other cases not having this connection will prevent the use of the CD unit for playing audio CD's. With the experiences of assembling the server and the compute nodes behind you, there is very little more in the way of a challenge in assembling remote-boot terminals, and you should have no real difficulty in putting the hardware together.

5.1.3 Configuring the Hardware

The configuration of the hardware of a remote-boot terminal is very similar to the configuration of the hardware of a compute node. For example, all the operations involved in the configuration and eventual programming of the network card are exactly the same, except perhaps for the fact that in the case of a terminal, which has its own keyboard, video card and monitor, there is no need of using the workbench for this purpose. We refer the reader, therefore, to Section 4.1 for most of the information on this subject, and comment here only on the differences found in the case remote-boot

terminals.

In the case of the setup of the BIOS of the motherboard there are a few differences between compute nodes and remote-boot terminals, mainly due to the existence of several additional devices on the latter. Many of the configuration options should be set just as in Section 4.1, except for the ones discussed below.

- You can start with a "load factory defaults", and also with a "load failsafe defaults" if it is available, but it is not a good idea to play around with "load optimized defaults" unless you want the terminal to double as a compute node.
- In the main menu there should be an option determining what the BIOS should do if there are errors during POST. You should leave it on the default setting of "halt on all errors".
- If the machine is to have one or more local boots besides the network boot, include the additional boot devices *after* the network, on the boot-devices menu.
- If you decided to leave out the floppy drive, disable both the configuration of the floppy unit in the first menu and the floppy controller in the peripheral device menu.
- If the terminal has any IDE disks or I/O devices, leave the IDE controllers on in the peripheral devices menu, and configure the IDE devices in the first menu.
- If you are going to use them, leave activated the following on-board devices: sound chip, PS/2 mouse, USB ports, parallel port, serial ports, network chip, etc.
- Set a password for access to the BIOS setup program; this is particularly important if the machine is to be installed in a public terminal room.
- Check the speed of the fans connected to the motherboard, as well as the temperatures of the motherboard and of the CPU.

As one can see, there are no special complications with the hardware setup of remote-boot terminals. All the changes pointed out above are quite intuitive and rather obvious, given the known differences in the hardware setup of a terminal as compared to a compute node.

In the case of the setup of the network card, it all goes on in exactly the same way as in Section 4.1 if you are using the PXELinux alternative. In the case of the Etherboot alternative there is a point where you must make a boot floppy in order to boot the machine with its filesystems in the server, so that you can use the `cromutil` utility in order to program the network card. Here there will be some small differences, since the kernel for the terminals is a bit different from that for the compute nodes. The floppy

image `lilo-boot.dd` for the case of the terminals is in the subdirectory `trm/` of the resources area, instead of `pmc/`.

When you make the necessary adjustments in order to use it, as explained in Section 4.1, there is a change to be done in the IP numbers, from those of the private PMC network `192.168.0` to numbers valid in your external LAN. We will continue to use here the network address `172.16.129` as a stand-in for the external LAN, although it too corresponds to a private network. The line within the `lilo.conf` file containing the network parameters now reads, in our example,

```
ip=172.16.129.40:172.16.129.30:172.16.129.1:255.255.255.0:::
```

where we see that the numbers corresponding to the server and to the gateway are now different. Note also that the parameter setting the console to use a serial port, `console=ttyS0,38400`, is missing in this case. Using the information contained in Section 4.1 and the additional discussion presented here, you should have no difficulty in getting your terminals to boot from your cluster server and into regular operation.

5.2 Adjustments in the Software

The general structure of the software aspects of clusters of remote-boot terminals is the same as that of clusters of compute nodes. We will use here the word "node" to denote also the filesystem structure for the system software of a single remote-booted machine, which may be either a compute node or a remote-boot terminal. The differences reside mainly in details and in the quantity of software which is typically installed in each case. Although the structure of the filesystems is essentially the same, some aspects of it may be a bit different, such as the location of the kernels, for reasons which will be discussed below. Also, usually a much larger set of packages is installed on terminals, and some of this additional software relates directly to the extra hardware which exists on the terminals, usually requiring some amount of manual configuration.

If one decides to install from scratch a cluster of terminals rather than compute nodes, then one may use most of the same installation sequence that was developed in Charters 3 and 4, involving the installation of the server, the cloning of the first node from it and the necessary changes in some files. The most basic set of software packages is still the same, but there is a fairly large list of additional packages to install. There are also some small adjustments to be made in the sequence of steps presented in those chapters. First, there is a general translation of some directories and hostnames: the directory `/pmc/` where the nodes reside should be changed to `/trm/`; the 4-digit naming scheme for the subdirectories for each node, starting with `0000/`, should be changed to a 2-digit scheme, starting at `00/`. In Table 5.4 you will find a complete list of all the necessary translations.

Description	Compute Cluster	Terminal Cluster
cluster directory	`/pmc/`	`/trm/`
node subdirectory	`0000/`	`00/`
node hostname	`n0000`	`fit00`
cluster server	`pmcs00`	`fit`
network address	`192.168.0`	`172.16.129`
IP of gateway	`192.168.0.1`	`172.16.129.1`
IP of cluster server	`192.168.0.1`	`172.16.129.30`
IP of first node	`192.168.0.10`	`172.16.129.40`
NIS netgroup	`pmcnodes`	`fitterms`

Table 5.4: Translations needed for the use of the installation sequence presented in Chapters 3 and 4 for the installation of a cluster of remote-boot terminals from scratch, or for the cloning of the first terminal out of a compute node.

If there will be both compute nodes and remote-boot terminals in the cluster, then one can also clone the first terminal from the template compute node. The translations in Table 5.4 apply to the adjustments needed in this case as well. Some translations are

related to the fact that there is no private network for the terminals in our example here. Although this is not strictly necessary, we assume that the terminals will be installed in the external LAN, directly connected to the Internet. Therefore, in this case all the action happens in the external LAN, the server has only one network interface and there is no need of any routing action on its part. This means that the IP masquerading structure will not be necessary and that the part dealing with it can be skipped in the installation sequence.

The addresses of the compute nodes are in a private network `192.168.0`, and start at `10`, while the terminals will be in the external LAN, which we will represent here by the network `172.16.129`, starting at the address `40`. Note that in fact this network address also belongs to a private network range, so that it is only a placeholder here, and should be changed to the address of the actual LAN the cluster will be built in. Note also that in this case the cluster server is not using the address `1` of the external LAN, indicating that it is not the gateway of that network. We assume that some other machine of the external LAN has the address `172.16.129.1` and acts as its gateway.

The host naming scheme for the nodes, such as `n0000` for the first node, should be changed to a name using the nickname chosen for the cluster of terminals. In our example here we use the same name used as a hostname for the cluster server, so that the hostnames of the terminals start with `fit00`. The hostname of the cluster server is the same as before, `fit`, but there is no longer a second hostname such as `pmcs00` associated to a second, private network. Since the hostnames are different, so will be the NIS netgroup for the terminals, which we choose to call `fitterms`.

All these changes must be implemented in every script and configuration file that uses these names and addresses, of course. In the resources area you will find a second set of such files in the directory `trm/`, in which all the changes have already been made, using our placeholder network as an example. Other files, which are outside both the `pmc/` and `trm/` subdirectories, but for which it is also necessary to have two versions, are marked with terminations `.PMC` and `.TRM`, with a symbolic link establishing the PMC version as the default.

5.2.1 Review of the Installation Sequence

Let us review quickly here the whole installation sequence, pointing out the important differences. It starts in Section 3.3, which deals with the installation of the basic system. Except for the translations listed in Table 5.4 and possibly some changes in disk partitioning, nothing changes in what is described in the initial part of this section. This includes Subsection 3.3.1, which deals with the installation of the system of the server itself. The first changes we must mention here come near the end of Subsection 3.3.2, which deals with the cloning of the first node. Here you will note that in the file `/etc/network/interfaces` of the terminal node the IP address of the gateway is no longer the IP address of the cluster server, which is this case is `172.16.129.30`:

```
# JLdL 26Sep04.
#
# /etc/network/interfaces -- configuration file for ifup(8), ifdown(8)
#
# The loopback interface
auto lo
iface lo inet loopback
#
# The first network card (network, broadcast and gateway are optional)
auto eth0
iface eth0 inet static
        address 172.16.129.40
        netmask 255.255.255.0
        network 172.16.129.0
        broadcast 172.16.129.255
        gateway 172.16.129.1
```

Instead, it must be address of the gateway of the external LAN, just as in the case of the corresponding file in the server. Another difference in this section, and a very important one, is that you should *not* disable the `getty` program for the `tty1`–`tty6` terminals in the `inittab` file of the terminals, as was done in the case of the compute nodes. Also, there is no need to enable the `getty` program for the serial ports `ttyS0` and `ttyS1`, so the `inittab` file should be simply left alone.

Next we get to Section 3.4, which deals with setting up the basic services for network booting. No changes are needed until we get to Subsection 3.4.1, which is about testing the filesystems of the first node. Here we should record the absence of the second hostname `pmcs00` for the server, in its file `/etc/hosts`. A fairly complete version of this file should now look like this:

```
# JLdL 26Sep04.
#
127.0.0.1       localhost
172.16.129.30   fit.free.univ.com       fit
#
172.16.129.40   fit00.free.univ.com     fit00
#
172.16.129.41   fit01.free.univ.com     fit01
172.16.129.42   fit02.free.univ.com     fit02
172.16.129.43   fit03.free.univ.com     fit03
172.16.129.44   fit04.free.univ.com     fit04
172.16.129.45   fit05.free.univ.com     fit05
172.16.129.46   fit06.free.univ.com     fit06
```

Another important difference due to the absence of the private network is the absence of the configuration for a second network card in the file `/etc/network/interfaces` of the server. Near the end of this subsection we come to the installation on the node of the second set of common packages. Since this may or may not have already been

done in the first terminal node, you should make sure that this set of packages is in fact installed on it, executing within the chroot shell, in the `/root/` directory of the node, where there should be a copy of the file `common-package-list-2`, the command

```
apt-get install ‘common-package-list-2‘
```

One should also install a larger collection of packages in the first terminal node, before one starts cloning other terminals from it, and this is a good spot to do that. A fairly complete list of the extra packages to be installed on a terminal can be found in the resources area, in the subdirectory `root/`, in the file `terminal-package-list`. This file should be copied into the `/root/` directory of the node, just like the file `common-package-list-2`. The installation of this extra set of packages can be done within the chroot shell as usual, with the command

```
apt-get install ‘terminal-package-list‘
```

executed in the `/root/` directory of the node. There will be some configuration questions to answer, but most of the necessary answers are reasonably obvious and, in case you do not know any better, it is safe to choose the defaults given by the packages. A few important additional packages, which do require a significant amount of configuration, will be discussed in more detail and installed a little later, including the X11 server. The installation of this additional set of packages will result in a total of about 850 packages installed in your first remote-boot terminal, so this process may take quite a while to complete. Also, do not forget to execute once more the command `apt-get clean` afterwards, in order to lower the occupation level of the `/var/` filesystem of the remote-boot terminals.

It is important to note here that this set of packages should *not* be installed on the server, for it contains several packages which are inappropriate for that case. A corresponding list of additional packages to be installed on the server can be found in the file `server-package-list` in the subdirectory `root/` of the resources area. If you wish, you may install that list in the cluster server. This list is a subset of the one for the terminals and should also not pose any complex configuration questions. We come next to Subsection 3.4.2, dealing with the preparation of the server for network booting. One important difference here is that the DHCP server is configured to use only the interface to the external LAN, rather than the interface to a private LAN. The relevant file is the `/etc/default/dhcp` file of server, which in this case should look like

```
# JLdL 26Sep04.
#
# Defaults for dhcp initscript
# sourced by /etc/init.d/dhcp
#
# On what interfaces should the DHCP server (dhcpd) serve DHCP requests?
#       Separate multiple interfaces with spaces, e.g. "eth0 eth1".
INTERFACES="eth0"
```

This change is reflected also in the file `/etc/dhcpd.conf` of the server, which is the case of the PXELinux alternative should now look like

```
# -*-shell-script-*-
# JLdL 30Sep04.
#
# dhcpd.conf for the PXELinux alternative.
#
# The subnet for eth0.
subnet 172.16.129.0 netmask 255.255.255.0 {
    option routers 172.16.129.1;
    filename "pxelinux.0";
    next-server fit.free.univ.com;
    server-name "fit.free.univ.com";
}
#
# The remote-boot terminals.
host fit00 {
    hardware ethernet AA:AA:AA:AA:AA:AA;
    fixed-address fit00.free.univ.com;
}
```

In the case of the Etherboot alternative there is a further important difference, for you will note that the discrimination of the filename of the kernel image to be booted in no longer within the global subnet declaration, but has been transfered to the host declaration of the node,

```
# -*-shell-script-*-
# JLdL 22Oct04.
#
# dhcpd.conf for the Etherboot alternative.
#
# The subnet for eth0.
subnet 172.16.129.0 netmask 255.255.255.0 {
    option routers 172.16.129.1;
    next-server fit.free.univ.com;
    server-name "fit.free.univ.com";
}
#
# The remote-boot terminals.
host fit00 {
    hardware ethernet AA:AA:AA:AA:AA:AA;
    fixed-address fit00.free.univ.com;
    filename "vmlinuz-00";
    option root-path "/trm/00,rw,nolock";
}
```

Also, this filename is now different for each node, marked with the node number. This is so because in the case of the remote-boot terminals, which tend to have a more

heterogeneous hardware, we allow for the possibility that each terminal will have a different kernel, possibly compiled with optimizations for a particular type of processor, or with support for some specialized hardware existing only in that terminal. The same is true in the PXELinux alternative, but in that case the difference does not show up here, but rather in the PXELinux configuration files. These configuration files, which are located now in the `/trm/tftpboot/pxelinux.cnf/` directory, should have their names changed in order to reflect the change from the private network `192.168.0` to the external LAN 172.16.129. Using once more the command `gethostip`, one finds that the terminal `fit00`, with address `172.16.129.40`, corresponds to the filename `AC108128`. Due to the change in the names, the `Makefile` within this directory must also change, of course. The `AC108128` configuration file looks like this:

```
# JLdL 02Oct04.

# General settings.
DEFAULT linux
DISPLAY pxelinux-banner
F0 pxelinux-banner.f0
F1 pxelinux-help.f1
F9 pxelinux.cfg/AC108128.f9
TIMEOUT 30
PROMPT 1

# Bootable images.
LABEL linux
  KERNEL vmlinuz-00
  APPEND root=/dev/nfs nfsroot=/trm/00,rw,nolock
  IPAPPEND 1
LABEL memtest
  KERNEL memtest86
LABEL local
  LOCALBOOT 0
```

where one can see the node-specific kernel name. Also, the activation of the serial console for PXELinux is absent, as well as the corresponding kernel boot parameter. If you look up the parameter `TIMEOUT` in the three example configuration files `AC108128`, `AC108129` and `AC10812A` in the resources area, you will see that in this case there is an global delay of 10 minutes, plus a stagger interval of 6 seconds per node. The long overall delay is convenient to give the boot server time to finish its own boot before the terminals start trying to boot through the network, for example when returning from a power failure. The kernel parameter activating the serial console is also absent from the `Makefile` in the `/trm/tftpboot/` directory, used for making the Etherboot NBI kernel image. In the current case that file should look like

```
# JLdL 22Oct04.
#
```

```
# Rule to make the NBI kernel image.
vmlinuz-nbi: vmlinuz Makefile
        mkelf-linux --rootdir=rom --ip=rom \
                --append="root=/dev/nfs" --output=$@ $<
```

We come next to Subsection 3.4.3, dealing with preparing the node for network booting. The only topic on which we must comment here is the compilation of the kernel for the node. First of all, the kernel configuration is different. You will find new kernel configuration files in the resources area, in the directory `trm/usr/00/src/`, for example with the name `Config-2.4.27-JLdL`. These new kernel configurations differ from those of the PMC nodes by not including support for the serial console, which is not necessary on terminals, and by having many more modules included, with drivers for a large number of devices one may want to use on a terminal. This includes, for example, modules for sound cards, USB devices and IDE devices. About 400 modules are included in the case of the 2.4 kernels.

Since the 2.6 kernel has several new features which can be very useful in remote-boot terminals, you will find in that directory configuration files for the newer kernels as well. You will also find there appropriate `disable-init-param` patches for these kernels. These configurations have basically the same characteristics of the configurations for the 2.4 kernels, but in this case more than 540 modules are included. Two noteworthy differences in these modules are the use of the ALSA sound card modules and the inclusion of "frame buffer" modules for the graphical control of video cards by the kernel. Also, these configuration have the "preemptible kernel" option turned on. Since older machines with ISA sound cards on a ISA bus may be re-used as remote-boot terminals, the support for this older type of bus is also included, as well as the ALSA modules for ISA cards.

In order to use the 2.6 kernels with modules, one needs the `module-init-tools` package, which is already included in our `common-package-list` file. The compilation of the 2.6 kernels goes in the same way as that of the 2.4 kernels, the only important differences being that the `make dep` step is no longer necessary and that the `make bzImage` and `make modules` steps can be executed at once with a simple `make`. The installation of the modules also goes in the same way, as well as the installation of the system map file, with only the necessary translations of directory names,

```
cp -p System.map /trm/00/boot/System.map-2.4.27
```

For the installation of the kernel image itself, however, we will use a different scheme here. Instead of installing it directly in the `/pmc/tftpboot/` directory, we will start by copying it into the `boot/` directory of the node,

```
cp -p arch/i386/boot/bzImage /trm/00/boot/vmlinuz-2.4.27
```

This makes the kernel image accessible from within the filesystem structure of the terminal node. We will also create in the same directory a hard link `vmlinuz` pointing to the same file, with the command

```
ln /trm/00/boot/vmlinuz-2.4.27 /trm/00/boot/vmlinuz
```

Since in order to enable the boot via TFTP it is necessary that the kernel file be accessible within the `/trm/tftpboot/` directory, we will create another hard link to the same file, within that directory,

```
ln /trm/00/boot/vmlinuz /trm/tftpboot/vmlinuz-00
```

Note that this last link has the name expected by the boot loaders, with the tag identifying the node. Note also that in all cases we are using here *hard links* rather than symbolic links, which is necessary due to the fact that the TFTP server in the `tftpd-hpa` package does not follow symbolic links pointing to files which are outside the `tftpboot/` directory tree. Since hard links can only be made for files within the same physical filesystem, we see that in this case it is *essential* that the `tftpboot` directory be located within the filesystem `/trm/`, which also contains the directories `boot/` within the root of each node. The scheme just described is appropriate for the PXELinux alternative. In the case of the Etherboot alternative the two last commands creating links must be modified. First, the `/trm/00/boot/vmlinuz` file may be a symbolic link rather than a hard link, created within the `/trm/00/boot/` directory with the command

```
ln -s vmlinuz-2.4.27 vmlinuz
```

One must then run `make` within the directory `/trm/00/boot/` in order to create the NBI kernel image `vmlinuz-nbi`. Second, the hard link within the `/trm/tftpboot/` directory must point to the NBI kernel image rather than to the normal kernel image, so it should be created with the command

```
ln /trm/00/boot/vmlinuz-nbi /trm/tftpboot/vmlinuz-00
```

In either case, this new installation scheme for the kernels allows the kernel of a terminal to be compiled and installed from within the system of the terminal, by an operator siting at its console, so long as certain cares are taken: the operator must be careful not to delete and loose the hard link `/boot/vmlinuz` (in the PXELinux alternative) or the hard link `/boot/vmlinuz-nbi` (in the Etherboot alternative), because this is the hard link to the file which is going to be sent over the network by the server during the next boot of the terminal. In order to install a new kernel the operator should use this link to *overwrite* that kernel file with the new image. In the Etherboot alternative this is done automatically by the `make` command, but in the PXELinux alternative one has to do it by hand with a command such as

```
cat vmlinuz-2.4.27 >!  vmlinuz
```

executed within the `/root/` directory of the terminal. This scheme opens up the possibility of having remote-boot terminals with autonomous managers, which will be able to compile and install a new kernel within the terminal, and to reboot the terminal with the new kernel, even if they do not have root access to the cluster server. If one uses the strict linking strategy for the terminal node filesystems, this establishes an interesting extension of the concept of a cluster of remote-boot terminals, in which management responsibilities could be shared between the managers of the central cluster server and the users of these terminals, giving to these users, for example, the capability of installing and maintaining the software packages of their terminals, as well of as activating and configuring the use of elements of hardware existing in their terminals.

The last part of Chapter 3 which we come to in this review is Subsection 3.4.4, which deals with the last preparations needed for full diskless operation of the cluster. Since there is no private network in our example of a cluster of remote-boot terminals, no routing or masquerading is required and all the initial part dealing with these issues can be simply skipped. The installation of the NTP system of hardware clock synchronization over the network requires only the obvious translations of the relevant names and addresses, as listed in Table 5.4. The same is true for the installation and configuration of the NIS system in both the server and the terminal, since the NIS server should now be `fit.free.univ.com` rather than `pmcs00.free.univ.com` in all the `/etc/yp.conf` files, and the NIS server should be configured in the `/etc/ypserv.securenets` file of the server to use only the external LAN instead of the private network.

At the end of this subsection we have the installation on the server of the second set of common software packages, to which may now be added the installation of the set of packages meant for the server, which can be found in the file `server-package-list` in the subdirectory `root/` of the resources area. In order to do this, copy the file into the `/root/` directory of the server and run there the command

```
apt-get install ‘cat server-package-list‘
```

Do not forget to run the command `apt-get clean` after this installation process is finished.

This ends the review of Chapter 3. In Chapter 4 we must restart the review only when we get to Section 4.2, which deals with the cloning of other nodes from the first one. All the basic ideas are the same here, the only difference being that one needs to adapt slightly the cloning scripts `make-new-netboot-node` which are described in Subsection 4.2.1. New versions of these scripts, adapted to out paradigmatic cluster of terminals, can be found in the resources area, in the subdirectories `trm/`, `trm/tmp/`, `trm/var/` and `trm/usr/`. These scripts operate and should be used, in our case here, exactly in the way described there for the case of compute nodes. Remember, however,

that you should complete the installation of the first terminal node before you start cloning other terminals from it. Therefore, do not actually start cloning additional terminals before you have finished reading this Chapter.

All that is said about the shared `/usr/` alternative in Subsection 4.2.2 applies equally in the present case. The adjustments in the system software of the server that are described in Subsection 4.2.3 are again subject to the set of translations given in Table 5.4. The first place were we find a significant difference is in the definition of the NIS netgroup `fitterms` for the remote-boot terminals, in the file `/etc/netgroup` of the server. This file should now look like

```
# JLdL 26Sep04.
#
# NIS netgroups.
#
# A group for the real remote-boot terminals.
fitterms \
(fit01.free.univ.com,-,fityp) (fit02.free.univ.com,-,fityp) \
(fit03.free.univ.com,-,fityp) (fit04.free.univ.com,-,fityp) \
(fit05.free.univ.com,-,fityp) (fit06.free.univ.com,-,fityp)
```

There is, of course, a corresponding difference in the NFS exports file `/etc/exports` of server, which should now look like

```
# JLdL 02Oct04.
#
# Terminal system to the server itself.
/trm        localhost(rw,async,no_root_squash) fit(rw,async,no_root_squash)
/trm/tmp    localhost(rw,async,no_root_squash) fit(rw,async,no_root_squash)
/trm/var    localhost(rw,async,no_root_squash) fit(rw,async,no_root_squash)
/trm/usr    localhost(rw,async,no_root_squash) fit(rw,async,no_root_squash)
#
# Terminal system to the complete group of remote-boot terminals.
/trm        fit00(rw,async,no_root_squash) @fitterms(rw,async,no_root_squash)
/trm/tmp    fit00(rw,async,no_root_squash) @fitterms(rw,async,no_root_squash)
/trm/var    fit00(rw,async,no_root_squash) @fitterms(rw,async,no_root_squash)
/trm/usr    fit00(rw,async,no_root_squash) @fitterms(rw,async,no_root_squash)
```

We come last to Section 4.3, which describes the package upgrade and filesystem maintenance scripts. All proceeds here for the remote-boot terminals exactly as in the case of compute nodes, up to the translation in Table 5.4. One needs, of course, new versions of the scripts `apt.chroot`, `multi-apt.chroot` and `hard-link-common-files`. The appropriate versions of these scripts can all be found in the resources area, the first two in the subdirectory `trm/` and the third one in the subdirectories `trm/`, `trm/var/` and `trm/usr/`.

One sees, therefore, that indeed there are few functional differences between clusters of compute nodes and remote-boot terminals, both types of cluster work in essentially

the same way, despite the rather marked differences in the role of each type. As usual, versions of all the files mentioned or shown here, possibly more complete and with more explanatory comments than the versions included in the text, can be found in the resources area.

5.2.2 Terminal-Specific Software

Many of the packages included in the list `terminal-package-list` are specific to terminals, relating to things such as recording CD's, playing sound from several different types of sources, editing sound files, editing images and various other types of X11 graphical applications. Of special importance, however, are two packages which relate directly to the operation of the monitor in graphical mode, and which may require some amount of manual configuration. These are the packages `gpm` and `xserver-xfree86`, and we will discuss here, in some detail, their installation and configuration.

Let us start with the package `gpm`, which implements the use of the mouse in the console, when it is used in text (non-graphical) mode. This is very useful both in the terminals and in the server itself, and we do recommend that you install it in both cases. Using it you will be able to mark, cut and paste pieces of text within a virtual terminal with your 3-button mouse. The program `gpm` is a daemon which reads the signal from the mouse device, and which also repeats this signal, presenting the repeated signal as another device which may be used by the X server, so that the mouse may be made available to the user both in test mode and in graphical mode. The daemon is also capable of translating the repeated signal from one mouse protocol to another, which sometimes helps to make the whole thing work better. Although in recent Linux kernels, which allow a double-open of the mouse device by two different applications, this mode of operation is not strictly required, it is usually a convenient one. Also, it is always convenient to have more than one way to do a thing, because if you encounter some serious difficulty with one, the other may save the day.

In order to make the `gpm` daemon operate, you will need to know not only which device in the machine corresponds to the mouse, but also the type of the mouse installed on it. The Linux kernel provides support for several different types of mouse device, but the two most common ones are PS/2 mice and serial mice. A PS/2 mouse uses the PS/2 connector of the motherboard, a small round mini-DIN connector which is identical to the one used by the keyboard, and located right next to it in all modern motherboards. A serial mouse uses a DB9 connector which is to be inserted into one of the two standard serial ports of the motherboard. Other less common devices are also supported, such as touchpads, digital tablets and some types of bus mouse, which are mice which connect to a special card on the system bus. While PS/2 mice are quite homogeneous in terms of the software protocol used, there is a large number of alternative protocols in use by serial mice. For PS/2 devices there are only two or three alternative protocols that you may have to try, but serial mice come with more than a dozen different types of

protocol.

The PS/2 mouse uses the auxiliary PS/2 device `/dev/psaux`, while a serial mouse uses either the device `/dev/ttyS0` or the device `/dev/ttyS1`, depending on whether it is plugged into the first or the second serial port. It is a good idea, however, to create a link called `mouse` in the `/dev/` directory of each machine, pointing to and so recording the device which corresponds to the mouse in that particular machine. This can be done with a command such as

```
ln -s psaux mouse
```

executed within the `/dev/` directory. Having done this and having determined the type of your mouse, you may now install the package with the usual command,

```
apt-get install gpm
```

This will be the most convenient approach either for the server or for the first remote-boot terminal *before* you clone others from it. In this last case you may do it on the actual running terminal or on the server, by means of a `chroot` shell. If you are doing the installation in remote-boot terminals *after* you have cloned them all from the first one, then it is more convenient to use the tools `apt.chroot` and `multi-apt.chroot`. The package installation tool will ask you a few questions, which of course you will try to answer as best you can. However, when you use these tools it may be more convenient to just accept all the defaults and leave the actual configuration to be done later by editing the configuration files manually. Here are a few pointers on how to answer the questions. When asked about the device, you can enter the full name of the link you just created. If you are in doubt about the type of mouse, try `imps2` for a PS/2 mouse or `ms3` for a serial mouse, which are very common types in wide use these days. Choose `msc` when asked about the repeat protocol. When asked about additional arguments to append to the command line of the daemon, answer with `-3 -R`. The installation tool will create a configuration file `/etc/gpm.conf` and then will start the daemon for you. If the `gpm` daemon is not started or if the mouse fails to work properly by the end of the installation, you will need to determine the mouse type and to edit the configuration file manually. You can start by making sure that the daemon is not running, executing a couple of times the command

```
/etc/init.d/gpm stop
```

You may then obtain a list of the supported mouse types with the command

```
gpm -t help
```

In order to try one of the types to check whether or not it works, you can use the `gpm` command in non-daemon mode, with the `-D` option. Assuming that you have made the soft link `/dev/mouse` and that the mouse has three buttons, you should execute the command line

```
gpm -D -m /dev/mouse -3 -t <your-pick-of-protocol>
```

and then try to move the mouse around to see what happens. The option `-3` tells the program that the mouse has three buttons. You may have to try several types before you see some reasonable results on the monitor. Make sure that the link is pointing to the correct device before you try this. If the command does not work at all or the mouse cursor jumps around the monitor like crazy, you can safely interrupt the program with `[Ctrl]-[C]` and try again with another type of protocol. In order to check the correct operation of all the three buttons, try selecting a piece of text in the screen by pressing and holding the left button, dragging the mouse, and then releasing the button. Pressing once the middle button of the mouse should then paste the selection at the current position of the keyboard cursor. The third button should extend the selection made, including everything up to the current position of the mouse cursor.

In this way you should be able to find the correct type for your mouse. After this, all you have to do is to record the correct results in the configuration file `/etc/gpm.conf`, editing it with your favorite editor. With this file in place and correctly configured the `gpm` daemon should be started automatically during the system boot. There is also the issue of the repetition of the mouse signal for the use of the X server. The option `-R` will tell the program to enable repetition of the signal on the FIFO `/dev/gpmdata`, which you will use as the mouse device for the X server. You may also choose the protocol (that is, the mouse type) into which the signal will be translated for the repetition. The configuration file should look like this:

```
# JLdL 27Sep04.
#
#  /etc/gpm.conf - configuration file for gpm(1)
#
device=/dev/mouse
responsiveness=
repeat_type=msc
type=imps2
append="-3 -R"
sample_rate=
```

A more fully commented version of this file can be found in the resources area, in the subdirectory `trm/00/etc/`. A version appropriate for a server, without signal repetition, can be found in the subdirectory `etc/`. The `msc` key in the line determining the type of protocol to repeat the signal in corresponds to the Mouse Systems protocol, which usually works very well with the X server. Having fixed this file, you may start the daemon with the command

```
/etc/init.d/gpm start
```

and you should have the mouse working correctly in text mode. This will make it a bit simpler to install the X server, because now you do not have any doubt as to the type of protocol the mouse signal read by the server will be in.

We can pass now to the installation of the X server. First of all you should determine the characteristics of your monitor, which you need to have handy during the installation. The relevant information consists of the vertical and horizontal ranges of frequencies supported by the monitor. The installation tool should be able to detect automatically the type of video card installed in the machine, so you should not have to worry about that. The installation is to be done with the command

```
apt-get install xserver-xfree86
```

Note that if you are doing this *after* having cloned all the terminals from the first one, the same remarks as in the case of the `gpm` software apply, regarding the use of the tools `apt.chroot` and `multi-apt.chroot`, as well as regarding the strategy of configuring the software by manually editing its configuration files later. Once more quite a few questions will be presented to you, and the configuration file will be written based on the information you give. Many questions can be answered with the defaults presented to you. The ones that you need to worry about are those that follow. You should let the installation tool probe your system for the video card. If you have enough video memory in your card, choose to have a color depth of 24 bits. You should enter carefully the exact frequency ranges of your monitor. You should enter `/dev/gpmdata` as the mouse device, and `MouseSystems` as its type of protocol. Since you have a 3-button mouse, you should reject the emulation of the third button by the simultaneous use of both buttons on a 2-button mouse. You will also you need to choose the graphics modes, which you should do based on the recommendations given in the previous section about their sizes, in relation to the size of the monitor. You should list them in descending order, such as

```
1440x1080 1280x960 1152x864 1024x768 800x600 640x480
```

which is the recommendation for a 19-inch monitor. After all questions are answered, the configuration file will be written and the installation process will end. However, do not expect the X11 system to start just now. In order to complete the information needed for the monitor section of the file, before we try to test the graphics subsystem, you may need to enter manually a few additional modes in the section about the monitor, as was discussed before. The files containing the extra modes are in the subdirectory `trm/00/etc/X11/` of the resources area. You will also find there examples of completed configuration files, including the additional modes appropriate for 17-inch and 19-inch monitors, with the names `XF86Config-4-17inch` and `XF86Config-4-19inch`. Note

that these files are subdivided into sections, one for each element of hardware or software involved in the configuration of the server. By looking at these files and comparing them to the one generated for you by the installation tool, you should have no difficulty in deciding how to make any corrections or additions which may be needed in your case.

With all this done, we are in a position to test the configuration of the X server. Remember that the `gpm` daemon should be running before you try to run the server. You can test the X server with the command

```
startx >& /tmp/startx.log
```

This should start the server and open an X11 session with a window manager. The mouse should be active, so move it around a bit and try out its buttons, both on the background and within windows. Try changing from one graphics mode to the next smaller one with the key combination `[Ctrl]-[Alt]-[Keypad(+)]`, as well as coming back large one with `[Ctrl]-[Alt]-[Keypad(-)]`. If you have any difficulty getting out of the graphics session, you can zap the server with the key combination `[Ctrl]-[Alt]-[BackSpace]`. Use this last one carefully, for it will suddenly kill all applications running in the graphics subsystem and then kill the server. What you have just done in this test is to start an X session manually. Usually the X session will come up during the boot of the system, managed by the `xdm` daemon, the X display manager. You can start it now using the command

```
/etc/init.d/xdm start
```

This will start a XDM banner which will allow you to log in, directly into the X11 subsystem. The X server will be running associated to the virtual terminal number 7 (if your system is running in multi-user mode), but remember that your original text session is still active in the first virtual terminal. You can go back to it with the key combination `[Ctrl]-[Alt]-[F1]`, and back to the X11 session with `[Alt]-[F7]`.

This ends the installation and configuration of the X server, including all the aspects relating to the hardware. You may need to adjust the geometry of the monitor for each one of the graphics modes, as a final touch in your graphics configuration. This is done with controls existing on the monitor itself, and can be done just once, since any fairly modern monitor will be able to remember the adjustments that you make. Later on you may want to further customize the appearance and functionality of the X session, which can be done by each user or in a centralized way, but all the basics for the operation of the graphics subsystem should already be in place. In later chapters we will discuss some of these customization issues, when we discuss the user working environment.

Another important subsystem which you must still configure is the sound subsystem, but in this case most of the relevant packages are already in place and there are no further packages to install relating directly to the control of the hardware. Usually the configuration of the sound card is only a matter of identifying and loading into the

kernel the correct module or modules, containing the appropriate driver. The loading can be done manually with the `modprobe` command, during the system boot by the inclusion of the module or modules in the `/etc/modules` file, or automatically with the on-demand module loader function of the kernel, called `kmod`, which involves creating an extra configuration file either in the `/etc/modutils/` directory, for the 2.4 kernels, or in the `/etc/modprobe.d/` directory, for the 2.6 kernels.

There are some differences in the procedure depending on whether you are using the 2.4 or the 2.6 kernels, and on whether your sound card is a PCI card or an ISA card. In the case of the 2.4 kernel you will be using the older OSS modules, while in the case of the 2.6 kernel you should use the newer ALSA modules. These two families of kernels use different sets of tools for dealing with modules, but both sets are already installed, since they are included in the basic system or in our lists of packages. The PCI sound cards are always much easier to configure than the ISA cards, specially if these are of the PNP (plug and play, or maybe it should be plug and *pray*) variety. While the 2.6 kernels with the ALSA modules come with a module to deal with this internally within the kernel, in the case of the 2.4 kernels with the OSS modules you will need a userspace utility to configure the ISA-PNP cards. The utility that does the hardware configuration of the card is called `isapnp` and is in the package `isapnptools`. There are also two auxiliary tools to help you to write the configuration file `isapnp.conf` of the `isapnp` utility, one called `pnpdump`, which come in the same package, and one called `sndconfig`, which does the same thing and also configures the modules as explained below. The `isapnptools` and `sndconfig` packages are also included in our lists of packages.

Manual loading of the modules can be used as a means of ascertaining which one is the correct module for the card. This will always work in the case of a PCI card, but for an ISA card you may have to first configure the card first with an ISA-PNP-capable tool, and then to configure the module-handling tools with the hardware characteristics of the card, such as its IRQ and I/O addresses. In order to learn how to handle these ISA cards you should read the documentation of the `isapnptools` and `sndconfig` packages. The sound card modules for either the 2.4 or the 2.6 kernels are located within the directory

```
/lib/modules/<kernel-version-number>/kernel/drivers/sound/
```

If you do not know which module will work for your card, you can look up within this directory the available modules and them simply try to load each one in turn until one of them succeeds. Each module will probe the hardware for the corresponding card, and will only load if it in fact finds the card it is looking for. In order to try to load a module into the kernel you can use the command

```
modprobe <module>
```

where you should use the name of the module, which is the name of the file holding it, without the `.o` or `.ko` termination. Since there may be dependencies among the modules, other modules may be loaded automatically by this command in addition to the one you list in the command line. Once the correct module has been found, the simplest way to automate its load is to include the name of the module in the file `/etc/modules`, so that it will be loaded during the boot of the system, and then remain loaded permanently. However, a better solution is to use `kmod`, the automatic module loading capability of the kernel. In order to configure this a file defining a few module aliases must be created in the directory `/etc/modutils` or in the directory `/etc/modprobe.d`, depending on which kernel is being used. A typical file for a PCI card using the OSS modules under a 2.4 kernel would look like this:

```
# JLdL 06Oct04.
#
# The sound card is a Creative PCI Sound Blaster AWE128.
alias char-major-14 es1371
alias sound-slot-0 char-major-14
```

A more complete version of this file can be found in the resources area, in the subdirectory `trm/00/etc/modutils/`, with the name `sound-PCI`. A file for an ISA-PNP card would be somewhat more complex, such as the following one:

```
# JLdL 06Oct04.
#
# The sound card is a Creative ISA-PNP Sound Blaster AWE64.
alias char-major-14 sb
alias sound-slot-0 char-major-14
#
# Load opl3 after sb.
post-install sb /sbin/modprobe "-k" "opl3"
#
# Options for everything.
options sb io=0x220 irq=10 dma=1 dma16=5 mpu_io=0x330
options opl3 io=0x388
```

A more complete version of this file can also be found in the resources area, with the name `sound-ISA`. Note that after any files are modified or created within the `/etc/modutils/` directory of the terminal, you must run the utility `update-modules`. This command will use the files within the directory to regenerate the actual configuration file for the modules, which is `/etc/modules.conf`. This is needed only for the 2.4 kernel, not for the 2.6 kernel, which uses a different configuration scheme.

It is important to emphasize, however, that the configuration of the modules is not enough to make an ISA-PNP card work under the 2.4 kernels with the OSS modules. One must also use the `isapnp` program to configure the addresses of the card, and these addresses must be the same that are encoded in the configuration of the modules. The

`isapnp` program runs during the system boot in order to configure all the ISA-PNP cards existing in the machine. It does this based on a configuration file `isapnp.conf` in the directory `/etc/`, which you must provide. The `pnpdump` tool will probe all the ISA-PNP cards and write a template configuration file that you then have to adjust manually in order to choose a set of addresses for each card. The `isapnptools` programs are not specific to sound cards, but can be used to configure any ISA-PNP cards. In the case of the sound card you may want to try the alternative tool `sndconfig`, which will produce both an ISA-PNP configuration and a configuration file for the modules. This is a tool written for the RedHat distribution which is now also available for Debian.

Things become significantly easier in the case of the 2.6 kernels with the ALSA sound modules, both in the case of PCI cards and in the case of ISA-PNP cards. This is due to the fact that this kernel has a module to handle ISA-PNP cards without the need for any userspace tools or utilities, and to the existence of an excellent configuration utility for the sound cards, the `alsaconf` program. This utility will probe for either PCI or ISA-PNP cards, and will write a module configuration file for the card it finds in the directory `/etc/modprobe.d/`. The configuration of the hardware will be handled automatically by the kernel, without any need for us to worry about it. A configuration file written by `alsaconf` for a PCI card looks like this:

```
# JLdL 06Oct04.
#
# The sound card is a Cirrus Logic PCI CrystalClear SoundFusion.
alias snd-card-0 snd-cs46xx
alias sound-slot-0 snd-cs46xx
install snd-cs46xx /sbin/modprobe --ignore-install snd-cs46xx && \
                /usr/lib/alsa/modprobe-post-install
```

The comments were included manually after the file was written by `alsaconf`, of course, and the line break in the last line was included only for formatting purposes, so the line would fit within margins. This file is available in the resources area, in the subdirectory `trm/00/etc/modprobe.d/`, with the name `sound-PCI`. A similar file written for an ISA-PNP card can also be found there, with the name `sound-ISA`, and looks like this:

```
# JLdL 06Oct04.
#
# The sound card is a Creative ISA-PNP Sound Blaster 16.
alias snd-card-0 snd-sb16
alias sound-slot-0 snd-sb16
options snd-sb16 isapnp=0
install snd-sb16 /sbin/modprobe --ignore-install snd-sb16 && \
                /usr/lib/alsa/modprobe-post-install
```

We can see, therefore, that the configuration of the sound card becomes significantly simpler with the ALSA modules of the 2.6 kernels. It is strongly recommended that

you use the `alsaconf` program to do the configuration of the card in this case. You will note that the program will run the `update-modules` utility, although this is not necessary in the case of the 2.6 kernels. This is so because the ALSA modules can also be used with the 2.4 kernels, but in this case it is a significantly more complex task to use them, because they are not within the 2.4 kernel source tree and must be compiled separately. The handling of the ISA-PNP cards also becomes more complex and you may face some unexpected compilation and installation problems. Therefore, if you want to use the ALSA modules it is advisable to migrate to the 2.6 kernel.

Having gone through the installation and configuration of a complex piece of software such as the X server, and having now the sound hardware in operating order, you might want to end this chapter trying your hand at the untutored installation of a couple of packages. It is suggested that you try to install in the terminal the packages `gom-x` and `xmcd`. They will ask you a few questions in order to configure the software. See what you can do about them, but there is no need to panic, neither of them is a crucial piece of software. Both are related to the sound subsystem, the first one is a mixer and the second an audio CD player. If all goes wrong with any of them, you already have alternatives in the system, since there are other mixers installed and the `xmms` (X multi-media system) player is able to play audio CD's.

As a final touch on the installation of your remote-boot terminal, we suggest that you install the `mplayer` application. It will enable you to play animations and movies in your terminal, from a variety of sources, such as movie files or a DVD player. This software is not yet included in the Debian distribution due to issues relating to software patents, but is available in the Debian package format. It can be found on the site `ftp.nerim.net`, in the directory `debian-marillat`. You can configure the `apt-get` program to get the software directly from it, including in the file `/etc/apt/sources.list` the line

```
deb ftp://ftp.nerim.net/debian-marillat testing main
```

while the "sarge" distribution is not yet released, or the line

```
deb ftp://ftp.nerim.net/debian-marillat stable main
```

after the release. With this done, you can refresh the databases of the APT structure with the command `apt-get update`, then install the program with the usual command,

```
apt-get install mplayer
```

As usual, several other packages will also be installed, to satisfy the dependencies among the packages. This site has also many other packages you may be interested in, you can have a look at its contents using a browser in FTP mode, that is, opening in it the URL `ftp://ftp.nerim.net/`.

With the X11 graphical subsystem up and running and the sound subsystem in operation, you will have a very complete terminal available. The installation of the packages described here will provide it with a solid software foundation, in fact with a fairly large set of packages. There, are however, many more packages available in the Debian distribution or elsewhere in Debian format. The final set of software packages you will install on your remote-boot terminals will depend, of course on the role you want them to perform and on your own choices among so many packages, sometimes with many alternatives for the same type of task. So, try to have your first terminal installed as completely and fully as possible, then clone the other from it. If later you find that you need some more software packages, there should be no difficulty in installing them, just as in the case of the compute nodes.

Chapter 6

Further System Structures

In this chapter we will discuss some additional structures which you should have in your systems. These are mostly structures that should exist regardless of whether or not the system is part of a cluster. In the next chapter we will discuss other additional structures, which are more closely related to clusters.

6.1 Automatic Backup Strategy

Backup of disk content is an essential ingredient for any reliable and stable computer system. But it can also become a heavy operational burden if it requires the routine manual intervention of the system operators. The traditional medium for data backup is the magnetic tape, but we will not use that medium here. Reliable magnetic tape backup systems are just too expensive, and the less expensive ones have some fairly serious reliability problems. Besides, all magnetic tape systems require manual intervention unless you get a *really* expensive tape robot. The same is true for backups made in optical media such as CD's and DVD's. We will, therefore, implement a backup strategy on disk. Each disk will have associated to it a second disk which will act as a backup. In addition, all backup operations will be completely automatic.

We should recall here that the disk sharing scheme among nodes described in Section 4.2 had the objective of avoiding the huge amount of disk occupation redundancy involved in the installation of a certain number of nodes in a cluster. What we will do here goes in exactly the opposite direction, for what a backup system does is to introduce in the system a certain amount of redundancy. It is, however, a limited and controlled amount of redundancy. Instead of having some disk content copied as many times as there are nodes, while other content may not have any backup at all, we will have a homogeneous factor-of-two redundancy backup for the whole system disk content, including the server and the nodes, as well as redundancy by at least a factor of two for the user data content, including the backup of older versions of files.

Let us recall also that, when we installed the system on the server in Section 3.3, only one of the two system disks installed in the machine was in fact used, and only the home disk was formatted, not the backup home disk. We will now make use of these extra disks. The system disk `/dev/sda` will have a backup disk `/dev/sdb` which will be kept as an exact mirror of the system disk, so that in the event of a complete failure of the system disk, the backup disk can simply be installed in its place. These two system disks will have their contents re-synchronized daily by means of a CRON job, a job which is run regularly and automatically by the system. The home disk `/dev/sdc` will have a backup disk `/dev/sdd` which is twice its size and with the content maintained by another CRON job which will use the `tar` command. This user data backup is more complex than the system backup and is intended to prevent loss of data not only due to hardware failures, but also due to common mistakes by the users.

6.1.1 Formatting The Backup Disks

The backup system disk should be partitioned in exactly the same way as the system disk. Although this is not strictly necessary, it is better if they are identical, with all partitions exactly of the same size. The partitioning schemes for the backup system disks, and the corresponding mount points, can be seen in Table 6.1 for the larger disk and in Table 6.2 for the smaller disk. Note that you should be *very careful* not to make

Partition	Device	Size	Mount point
1 (P)	/dev/sdb1	512	/bckp/
2 (P)	/dev/sdb2	2048	none (swap)
3 (P)	/dev/sdb3	512	/bckp/tmp/
5 (L)	/dev/sdb5	6144	/bckp/var/
6 (L)	/dev/sdb6	12128	/bckp/usr/
7 (L)	/dev/sdb7	512	/bckp/trm/
8 (L)	/dev/sdb8	512	/bckp/trm/tmp/
9 (L)	/dev/sdb9	6144	/bckp/trm/var/
10 (L)	/dev/sdb10	8192	/bckp/trm/usr/

Table 6.1: The partitioning scheme for the 36 GB backup system disk of the cluster of remote-boot terminals. The disk should have SCSI address 1, corresponding to the Linux device `/dev/sdb`. Partition sizes are in MB. Primary partitions are labeled with (P), logical ones with (L).

any mistakes while performing the operation in this subsection, because they could lead to the destruction of all your work so far. You can partition the backup system disk using the command

```
cfdisk /dev/sdb
```

just as you did before during the installation of the system on the server. You should also partition the backup home disk with the command

```
cfdisk /dev/sdd
```

This disk should also have a single partition, just like the home disk, but this partition will be about twice as large as the one in the home disk. The formatting of the filesystems should be done exactly in the same way as the corresponding filesystems in the system disk, including the special options used for some of the filesystems for the nodes. For example, the backup root filesystem of the server should be formatted with the command

```
mke2fs -j /dev/sdb1
```

and the backup `var/` and `usr/` filesystems with similar commands for the `/dev/sdb5` and `/dev/sdb6` partitions. The `/dev/sdb3` partition should not be formatted right now, because there will be no backup for the `/tmp/` filesystem, given that it is a completely volatile filesystem, with only temporary data. The backup of the root of the nodes, `trm/` or `pmc/`, should be formatted with the command

```
mke2fs -j -i 1024 /dev/sdb7
```

Partition	Device	Size	Mount point
1 (P)	/dev/sdb1	256	/bckp/
2 (P)	/dev/sdb2	1024	none (swap)
3 (P)	/dev/sdb3	256	/bckp/tmp/
5 (L)	/dev/sdb5	1024	/bckp/var/
6 (L)	/dev/sdb6	4608	/bckp/usr/
7 (L)	/dev/sdb7	512	/bckp/pmc/
8 (L)	/dev/sdb8	512	/bckp/pmc/tmp/
9 (L)	/dev/sdb9	3072	/bckp/pmc/var/
10 (L)	/dev/sdb10	3072	/bckp/pmc/usr/
11 (L)	/dev/sdb11	4037	/bckp/temp/
10 (L)	/dev/sdb10	7109	/bckp/trm/usr/

Table 6.2: The partitioning scheme for the 18 GB backup system disk of the cluster of compute nodes. The disk should have SCSI address 1, corresponding to the Linux device `/dev/sdb`. Partition sizes are in MB. Primary partitions are labeled with (P), logical ones with (L). Two alternatives for the filesystems at the end of the disk are shown, one of them for use with remote-boot terminals rather than compute nodes.

The `/trm/tmp/` or `/pmc/tmp/` filesystem of the nodes will also have a backup, since they include data which is not temporary, such as cluster maintenance scripts and their log files. This filesystem can be formatted in the usual way, with the command

```
mke2fs -j /dev/sdb8
```

The backup `trm/var/` or `pmc/var/` filesystem of the nodes should be formatted with the command

```
mke2fs -j -i 2048 /dev/sdb9
```

and the `trm/usr/` or `pmc/usr/` filesystem with a similar command for the `/dev/sdb10` partition. The `/dev/sdb11` partition which exists in the case of the smaller disk used for the PMC should not be formatted at this stage, because the temporary filesystem `/temp/` will also not have a backup. Finally, the backup home filesystem should be formatted with the command

```
mke2fs -j /dev/sdd1
```

Next you should label all the filesystems just created, using the mount point as the labels, just as we did before in the case of the system disks. All the backup filesystems will be mounted on the system under a mount point `/bckp/`, so you can label the backup root filesystem with the command

```
e2label /dev/sdb1 /bckp
```

the backup `/bckp/var/` of the `/var/` filesystem with the command

```
e2label /dev/sdb5 /bckp/var
```

and similarly for the filesystems `/bckp/usr/` and `/bckp/home/`, on partitions `/dev/sdb6` and `/dev/sdc1`, which should use `/bckp/usr` and `/bckp/home` as labels, respectively. The same should also be done for the filesystems of the nodes, under `/trm/` or `/pmc/` as the case may be. In Figures 6.1 through 6.8 you can see what the `cfdisk` command should display in each case, for all the disks in the system, after they are all partitioned, and all the filesystems are formatted and labeled.

All that is left to do now is to create the appropriate mount points and to mount the backup filesystems in their places. You should also update the `/etc/fstab` file of the server, of course, which should now look like this in the case of the server for remote-boot terminals:

```
# JLdL 09Oct04.
#
# <file system> <mount point>   <type>  <options>                       <dump>  <pass>
/dev/sda1       /               ext3    defaults,errors=remount-ro      0       1
#
/dev/sda2       none            swap    sw                              0       0
#
proc            /proc           proc    defaults                        0       0
#
/dev/sda3       /tmp            ext3    rw                              0       2
/dev/sda5       /var            ext3    rw                              0       2
/dev/sda6       /usr            ext3    rw                              0       2
#
/dev/sda7       /trm            ext3    rw                              0       2
/dev/sda8       /trm/tmp        ext3    rw                              0       2
/dev/sda9       /trm/var        ext3    rw                              0       2
/dev/sda10      /trm/usr        ext3    rw                              0       2
#
/dev/sdc1       /home           ext3    rw                              0       2
#
/dev/sdb1       /bckp           ext3    rw                              0       2
/dev/sdb5       /bckp/var       ext3    rw                              0       2
/dev/sdb6       /bckp/usr       ext3    rw                              0       2
#
/dev/sdb7       /bckp/trm       ext3    rw                              0       2
/dev/sdb8       /bckp/trm/tmp   ext3    rw                              0       2
/dev/sdb9       /bckp/trm/var   ext3    rw                              0       2
/dev/sdb10      /bckp/trm/usr   ext3    rw                              0       2
#
/dev/sdd1       /bckp/home      ext3    rw                              0       2
```

or like this, in the case of compute nodes:

```
# JLdL 09Oct04.
#
# <file system> <mount point>   <type>  <options>                       <dump>  <pass>
/dev/sda1       /               ext3    defaults,errors=remount-ro      0       1
#
/dev/sda2       none            swap    sw                              0       0
#
proc            /proc           proc    defaults                        0       0
#
/dev/sda3       /tmp            ext3    rw                              0       2
/dev/sda5       /var            ext3    rw                              0       2
/dev/sda6       /usr            ext3    rw                              0       2
#
/dev/sda7       /pmc            ext3    rw                              0       2
/dev/sda8       /pmc/tmp        ext3    rw                              0       2
/dev/sda9       /pmc/var        ext3    rw                              0       2
/dev/sda10      /pmc/usr        ext3    rw                              0       2
#
/dev/sdc1       /home           ext3    rw                              0       2
#
/dev/sdb1       /bckp           ext3    rw                              0       2
/dev/sdb5       /bckp/var       ext3    rw                              0       2
/dev/sdb6       /bckp/usr       ext3    rw                              0       2
#
/dev/sdb7       /bckp/pmc       ext3    rw                              0       2
/dev/sdb8       /bckp/pmc/tmp   ext3    rw                              0       2
/dev/sdb9       /bckp/pmc/var   ext3    rw                              0       2
/dev/sdb10      /bckp/pmc/usr   ext3    rw                              0       2
#
/dev/sdd1       /bckp/home      ext3    rw                              0       2
```

These files can be found in the subdirectory `etc/` of the resources area, with the names `fstab.1.TRM` and `fstab.1.PMC`. In order to mount the new filesystems you must start by creating the `/bckp/` directory with the command

```
mkdir /bckp
```

and then, so long as the changes in the `/etc/fstab` file are made, by mounting the backup root filesystem with the command

```
mount /bckp
```

You may then create the mount points `var/`, `usr/`, `trm/` or `pmc/` and `home/` within the `/bckp/` filesystem, and then make the corresponding mounts. After the `/bckp/trm/` or `/bckp/pmc/` filesystem is mounted you can repeat the procedure within

```
                        Disk Drive: /dev/sda
                  Size: 36703934464 bytes, 36.7 GB
          Heads: 64   Sectors per Track: 32   Cylinders: 35003

   Name        Flags      Part Type  FS Type          [Label]        Size (MB)
------------------------------------------------------------------------------
   sda1        Boot        Primary   Linux ext3       [/]              511.71
   sda2                    Primary   Linux swap                       2047.87
   sda3                    Primary   Linux ext3       [/tmp]           511.71
   sda5                    Logical   Linux ext3       [/var]          6143.61
   sda6                    Logical   Linux ext3       [/usr]         12121.54
   sda7                    Logical   Linux ext3       [/trm]           511.71
   sda8                    Logical   Linux ext3       [/trm/tmp]       511.71
   sda9                    Logical   Linux ext3       [/trm/var]      6144.66
   sda10                   Logical   Linux ext3       [/trm/usr]      8192.53



    [Bootable]  [ Delete ]  [  Help  ]  [Maximize]  [ Print  ]
    [  Quit  ]  [  Type  ]  [ Units  ]  [ Write  ]
```

Figure 6.1: The completed partition table for the 36 GB system disk as seen in the `cfdisk` screen, after all filesystems are formatted and labeled. Note the bootable flag on the root partition, and observe that the partitions which participate on a RAID array appear here in the usual way.

```
                        Disk Drive: /dev/sdc
                  Size: 36703934464 bytes, 36.7 GB
          Heads: 64   Sectors per Track: 32   Cylinders: 35003

   Name        Flags      Part Type  FS Type          [Label]        Size (MB)
------------------------------------------------------------------------------
   sdc1                    Primary   Linux ext3       [/home]         36703.31



    [Bootable]  [ Delete ]  [  Help  ]  [Maximize]  [ Print  ]
    [  Quit  ]  [  Type  ]  [ Units  ]  [ Write  ]
```

Figure 6.2: The completed partition table for the 36 GB home disk as seen in the `cfdisk` screen, after all filesystems are formatted and labeled. This disk is to have a single partition containing all the available space.

```
                        Disk Drive: /dev/sda
                  Size: 18373206016 bytes, 18.3 GB
            Heads: 64   Sectors per Track: 32   Cylinders: 17522

   Name        Flags      Part Type  FS Type          [Label]        Size (MB)
------------------------------------------------------------------------------
   sda1        Boot        Primary   Linux ext3       [/]               255.86
   sda2                    Primary   Linux swap                        1024.46
   sda3                    Primary   Linux ext3       [/tmp]            255.86
   sda5                    Logical   Linux ext3       [/var]           1024.46
   sda6                    Logical   Linux ext3       [/usr]           4608.50
   sda7                    Logical   Linux ext3       [/pmc]            511.71
   sda8                    Logical   Linux ext3       [/pmc/tmp]        511.71
   sda9                    Logical   Linux ext3       [/pmc/var]       3072.33
   sda10                   Logical   Linux ext3       [/pmc/usr]       3072.33
   sda11                   Logical   Linux ext3       [/temp]          4013.95


    [Bootable]  [ Delete ]  [  Help  ]  [Maximize]  [ Print  ]
    [  Quit  ]  [  Type  ]  [ Units  ]  [ Write  ]
```

Figure 6.3: The completed partition table for the 18 GB system disk as seen in the `cfdisk` screen, after all filesystems are formatted and labeled. Note the bootable flag on the root partition, and observe that the partitions which participate on a RAID array appear here in the usual way.

```
                        Disk Drive: /dev/sdc
                  Size: 18373206016 bytes, 18.3 GB
            Heads: 64   Sectors per Track: 32   Cylinders: 17522

   Name        Flags      Part Type  FS Type          [Label]        Size (MB)
------------------------------------------------------------------------------
   sdc1                    Primary   Linux ext3       [/home]         18373.15




    [Bootable]  [ Delete ]  [  Help  ]  [Maximize]  [ Print  ]
    [  Quit  ]  [  Type  ]  [ Units  ]  [ Write  ]
```

Figure 6.4: The completed partition table for the 18 GB home disk as seen in the `cfdisk` screen, after all filesystems are formatted and labeled. This disk is to have a single partition containing all the available space.

```
                        Disk Drive: /dev/sdb
                  Size: 36703934464 bytes, 36.7 GB
            Heads: 64   Sectors per Track: 32   Cylinders: 35003

   Name        Flags      Part Type  FS Type          [Label]        Size (MB)
------------------------------------------------------------------------------
   sdb1        Boot        Primary   Linux ext3       [/bckp]          511.71
   sdb2                    Primary   Linux swap                       2047.87
   sdb3                    Primary   Linux                             511.71
   sdb5                    Logical   Linux ext3       [/bckp/var]     6143.61
   sdb6                    Logical   Linux ext3       [/bckp/usr]    12121.54
   sdb7                    Logical   Linux ext3       [/bckp/trm]      511.71
   sdb8                    Logical   Linux ext3     [/bckp/trm/tmp]    511.71
   sdb9                    Logical   Linux ext3     [/bckp/trm/var]   6144.66
   sdb10                   Logical   Linux ext3     [/bckp/trm/usr]   8192.53



    [Bootable]  [ Delete ]  [  Help  ]  [Maximize]  [ Print  ]
    [  Quit  ]  [  Type  ]  [ Units  ]  [ Write  ]
```

Figure 6.5: The completed partition table for the 36 GB backup system disk as seen in the `cfdisk` screen, after all filesystems are formatted and labeled. Note the bootable flag on the root partition, and observe that the partitions which participate on a RAID array appear here as unformatted.

```
                        Disk Drive: /dev/sdd
                  Size: 73407865856 bytes, 73.4 GB
            Heads: 64   Sectors per Track: 32   Cylinders: 70007

   Name        Flags      Part Type  FS Type          [Label]        Size (MB)
------------------------------------------------------------------------------
   sdd1                    Primary   Linux ext3       [/bckp/home]    73407.67



    [Bootable]  [ Delete ]  [  Help  ]  [Maximize]  [ Print  ]
    [  Quit  ]  [  Type  ]  [ Units  ]  [ Write  ]
```

Figure 6.6: The completed partition table for the 72 GB backup home disk as seen in the `cfdisk` screen, after all filesystems are formatted and labeled. This disk is to have a single partition containing all the available space.

```
                        Disk Drive: /dev/sdb
                  Size: 18373206016 bytes, 18.3 GB
            Heads: 64   Sectors per Track: 32   Cylinders: 17522

   Name        Flags      Part Type  FS Type          [Label]        Size (MB)
------------------------------------------------------------------------------
   sdb1        Boot        Primary   Linux ext3       [/bckp]          255.86
   sdb2                    Primary   Linux swap                       1024.46
   sdb3                    Primary   Linux                             255.86
   sdb5                    Logical   Linux ext3       [/bckp/var]     1024.46
   sdb6                    Logical   Linux ext3       [/bckp/usr]     4608.50
   sdb7                    Logical   Linux ext3       [/bckp/pmc]      511.71
   sdb8                    Logical   Linux ext3     [/bckp/pmc/tmp]    511.71
   sdb9                    Logical   Linux ext3     [/bckp/pmc/var]   3072.33
   sdb10                   Logical   Linux ext3     [/bckp/pmc/usr]   3072.33
   sdb11                   Logical   Linux                            4013.95


    [Bootable]  [ Delete ]  [  Help  ]  [Maximize]  [ Print  ]
    [  Quit  ]  [  Type  ]  [ Units  ]  [ Write  ]
```

Figure 6.7: The completed partition table for the 18 GB backup system disk as seen in the `cfdisk` screen, after all filesystems are formatted and labeled. Note the bootable flag on the root partition, and observe that the partitions which participate on a RAID array appear here as unformatted.

```
                        Disk Drive: /dev/sdd
                  Size: 36703934464 bytes, 36.7 GB
            Heads: 64   Sectors per Track: 32   Cylinders: 35003

   Name        Flags      Part Type  FS Type          [Label]        Size (MB)
------------------------------------------------------------------------------
   sdd1                    Primary   Linux ext3       [/bckp/home]    36703.31




    [Bootable]  [ Delete ]  [  Help  ]  [Maximize]  [ Print  ]
    [  Quit  ]  [  Type  ]  [ Units  ]  [ Write  ]
```

Figure 6.8: The completed partition table for the 36 GB backup home disk as seen in the `cfdisk` screen, after all filesystems are formatted and labeled. This disk is to have a single partition containing all the available space.

it for the subdirectories `tmp/`, `var/` and `usr/` which will be the mount points of the node filesystems. With this all your backup filesystems will be in place and ready to be used.

One additional thing you may want to do at this point is to fix the `lost+found/` directories, to make sure that they have a sufficiently large allocation of disk blocks. Every directory which is the root of a physical filesystem must have a correctly configured `lost+found/` subdirectory within it. You can delete the directory and create a fresh one using the commands

```
rmdir lost+found ; mklost+found
```

In fact, it is not a bad idea to do this in *all* the filesystems of the server, to ensure that all the `lost+found/` directories have large block allocations.

6.1.2 Automatic System Backup Script

The mirroring of the system disk on the backup system disk will be made with the `rsync` command. Although this is a network-aware copy program meant to be used remotely, it can also be used locally as a highly efficient recursive copying machine. This command will be used through a script called `myrror-sysdisk` which should be installed in the directory `/root/bin/` of the server. The script simply uses the `rsync` command to copy each filesystem onto the corresponding backup filesystem, in the way appropriate to guarantee that they become identical. The `rsync` program works in a very efficient way because only new files or files that have been changed are copied, and in the case of changed files only the changed blocks are copied. If the two filesystems are already identical, this is simply verified and nothing is actually copied. Here is the script for the case of the PMC, in full:

```
#!/bin/tcsh -f
# JLdL 08Oct04.
#
# Clone the system disk onto a second disk.
#
set log = /root/log/myrror-sysdisk.log
cat /dev/null >! $log
set opts = "--archive --hard-links --one-file-system --delete --delete-after"
echo doing /                                                    >>& $log
/usr/bin/nice -19 rsync $opts / /bckp                           >>& $log
echo doing /var                                                 >>& $log
/usr/bin/nice -19 rsync $opts /var/ /bckp/var                   >>& $log
echo doing /usr                                                 >>& $log
/usr/bin/nice -19 rsync $opts /usr/ /bckp/usr                   >>& $log
echo doing /pmc                                                 >>& $log
/usr/bin/nice -19 rsync $opts /pmc/ /bckp/pmc                   >>& $log
echo doing /pmc/tmp                                             >>& $log
```

```
/usr/bin/nice -19 rsync $opts /pmc/tmp/ /bckp/pmc/tmp          >>& $log
echo doing /pmc/var                                            >>& $log
/usr/bin/nice -19 rsync $opts /pmc/var/ /bckp/pmc/var          >>& $log
echo doing /pmc/usr                                            >>& $log
/usr/bin/nice -19 rsync $opts /pmc/usr/ /bckp/pmc/usr          >>& $log
echo done                                                      >>& $log
```

This script can be found in the subdirectory `root/bin/` of the resources area, with the name `myrror-sysdisk.PMC`, as well as a similar one named myrror-sysdisk.TRM for remote-boot terminals. Note that it will create a log file in the directory `/root/log/` of the server, which you should now create. The log file will be called `myrror-sysdisk.log` and will be overwritten every time the script runs. The option `--archive` will cause the `rsync` program to faithfully reproduce all symbolic links, device nodes and other types of files, preserving their access permissions and ownerships. This is complemented by the `--hard-links` option, which causes the program to preserve the structure of files with several hard links, something which is quite crucial for the node filesystems. The option `--one-file-system` causes the program to remain within one filesystem, not descending into the mount points of other filesystems mounted under it. Finally, the option `--delete` tells the program to delete from the backup filesystem files which have been deleted from the original filesystem, and the option `--delete-after` tells it to do so only `after` all new files and changed files have been copied.

Note that in this script the program `rsync` runs under the command `/usr/bin/nice`, configured to run the program with the lowest possible priority `19`, so as not to disturb other programs which may be running in the system. Since this type of backup operation is bound by the I/O traffic and not by CPU usage, running at low priority is not a serious limitation for the backup operations. Note also that in each line the `rsync` command has two arguments, the first being the mount point of the filesystem to be backed up, and the second being the mount point of the corresponding backup filesystem. All arguments are absolute paths and the first argument always ends with a slash, while the second argument never does. Usually a trailing slash just marks a name as the name of a directory, but has no further effect in the behavior of any commands or programs. In the case of the `rsync` program, however, a trailing slash on the first argument instructs it to make the program copy the *contents* of the directory, while without the trailing slash the program will copy the directory itself. This difference is important and a wrong choice here, combined with the `--delete` option, can lead to disastrous results.

Having installed this script in your server, you should run it a first time by hand, in order to create the content of the backup filesystems. You can do this running the script in the background, with the command

```
/root/bin/myrror-sysdisk &
```

You can monitor the progress of the backup operation using the command `df` at regular intervals, or by running it under the `watch` command, as in

```
watch df
```

In order to have it run regularly, you must include an appropriately designed entry in the configuration file of the `cron` daemon, for the root user, called the *root crontab*. This daemon should already be running in the system, it examines the crontab of each user once a minute and takes action as determined there. In order to examine the current contents of the root crontab you can execute the command `crontab -l`. It is probably empty. In order to edit it with the `emacs` editor, you should first set the environment variable `EDITOR` to this editor, or to whatever editor you prefer, with a command such as

```
setenv EDITOR emacs
```

Having set this variable, you can edit the root crontab executing as root the command `crontab -e`. You should then include in the crontab a line such as the one below:

```
#
# Backup the system filesystems.
0 0 * * * /bin/tcsh /root/bin/myrror-sysdisk
```

The first five entries in the line discriminate the time when the job should be performed, in this case at zero minutes, zero hours, of every day of every week and month. The rest of the line discriminates what is to be run; in this case the shell `/bin/tcsh` is called to execute the script `/root/bin/myrror-sysdisk`. A more complete version of the root crontab can be found in the subdirectory `root/` of the resources area, with the name `crontab`. Note, however, that this is just an example of what you should see as the output of the `crontab -l` command, for you should never edit directly the crontab file itself, which is located in the `/var/` filesystem, and should always change the crontab through the `crontab -e` command.

6.1.3 Automatic Home Backup Script

The backup of user data will be done in a way which is quite different from the backup of system data. It will *not* be done by mirroring, and will be more frequent than the system backup. The most common use of the backup of user data is the recovery of files which the users delete by mistake or accident. Since you must have the backup copy intact if the file is to be recovered in such cases, an automatic backup by mirroring, which deletes from the backup filesystem files which have been deleted from the home filesystem, simply will not do. Another common problem with user data is the accidental overwriting of a file with erroneous content. If this file is subsequently backed up,

typically the backup copy will be overwritten and its content lost. Therefore, if one is to be able to recover the correct content in such cases, one needs to keep also a backup copy of the older version of the every file which is being backed up. This can be accomplished if the backup system, instead of overwriting files in the backup filesystem, moves the existing files to alternative names before writing new ones.

The backup system for user data can in fact have all these properties if it is made with the command `tar`, the same command we used before to clone whole filesystems. The script to do this is, however, somewhat more complex than the script to do the system backup. A copy of the script can be found in the subdirectory `root/bin/` of the resources area, with the name `myrror-user`. It calls and uses a second script called `myrror-user-do-the-backup`, located in the subdirectory `root/lib/`. You should make copies of these script into the corresponding directories of your server and edit them with the `emacs` editor, so it will be easier to follow the explanation.

The script `myrror-user` starts with a few lines defining the directory `/bckp/` to be the root of the backup filesystems and the directory `/root/var/myrror/` to be the place where the account-specific log files and "snap" files of the `tar` command will be kept. The script will create a set of account-specific log files, containing any errors or warnings pertaining to the backup of the home directory of each user. The snap files are also account-specific and contain a record of what has already been backed up in the past in each account, so that an incremental backup can be made every time the `tar` command is executed, in which only new or modified files are backed up. The next block of code handles a lock file in the usual way, to ensure that the script is not run twice at the same time:

```
if ( -f $lockfile ) then
    echo "‘basename $0‘: this script is already running"
    exit 1
endif
touch $lockfile
onintr cleanexit
```

It also defines a clean-exit label, so that the lock file is not left over when the script is interrupted from the keyboard. The next block of code uses the command `find` in the `/home/` directory, in order to define a list of user accounts to be backed up:

```
set excl = '^/home$'
set excl = "$excl|"'^/home/lost\+found$'
set excl = "$excl|"'^/home/ftp$'
set directs = ‘find /home -maxdepth 1 -type d | sort | egrep -v "$excl"‘
```

There are a few exclusions to be made in this list, including the directory `/home/` itself and the subdirectory `lost+found/`, which are implemented by means of a reverse `grep` search. If you put in the `/home/` directory anything of which you do not want to

be a backup copy, this is the place to include new exclusions. The last block of code is a loop which spans the subdirectories to be backed up, calling the other script in order to actually do the backup:

```
foreach direct ( $directs )
    set usernm = ‘basename $direct‘
    source /root/lib/myrror-user-do-the-backup
end
```

The call with the keyword `source` is equivalent to the inclusion of the called script at that point in the code, so all variables defined so far are available to the script called. The last couple of line of the `myrror-user` script deal with the removal of the lock file. The first block of code in the `myrror-user-do-the-backup` script creates the root of the directory structure of the account in the backup filesystem, if it does not already exist:

```
set names = ( ‘echo $direct | sed -e ’s|/| |g’ | cut -d" " -f 2-‘ )
set count = 1
set parts = $names[$count]
while ( $count < $#names )
    @ count = $count + 1
    set parts = $parts/$names[$count]
    if ( ! -d $backup/$parts ) then
        mkdir $backup/$parts
        if ( ‘basename $backup/$parts‘ == $usernm ) then
            chown ${usernm}:$usernm $backup/$parts
        endif
    endif
end
```

This piece of code is more general than is has to be, in practice all this does is to create the directory of the user's account in the backup filesystem, and to fix its ownership. The next line creates the stem of a filename from the name of the directory to be backed up, for use in the account-specific log and snap files. The next block of code defines a list of subdirectories to be excluded from the `tar` backup operation:

```
set exclude = "--exclude tmp"
set exclude = "$exclude --exclude temp"
set exclude = "$exclude --exclude tempo"
set exclude = "$exclude --exclude Maildir"
set exclude = "$exclude --exclude nsmail"
```

These must be subdirectories of the root of the account which is being backed up, they cannot be further down in the directory tree, and the names in the list cannot contain slashes. Note that subdirectories meant to hold temporary data should be excluded. Mail reception directories are also excluded here, because they should hold only transient

content and backing them up in a regular way would quickly fill the backup directory with lots of unwanted files. The next block of code does the actual backup, using a pipeline of `tar` commands with options that cause them to do an incremental backup:

```
( \
( cd $direct ; \
    /usr/bin/nice -19 tar $exclude -cg $myrror/$filenm.snap -f - . ) \
        | \
( cd $backup$direct ; \
    /usr/bin/nice -19 tar --backup=simple --suffix=.~P~ -xpf - ) \
                ) >&! $myrror/$filenm.log
```

Note that the two `tar` processes are also run with a low priority, under the `nice` command. The option `-g` in the first instance of the `tar` program puts it in incremental backup mode. This process also handles the subdirectory exclusions. The second instance implements the double-layered backup strategy. This is being done in a way which is quite different from what we saw before when we used the `tar` command to clone filesystems. The outer set of parenthesis in each one of the two parts of the pipeline cause their contents to be executed in sub-shells, so that they can contain more than one command. The "simple" backup determined by the options on the second instance of the `tar` program means that files will not be overwritten, but that only the previous version will be kept, not any versions older than that one. The previous version of each file will be kept with a modified name, including a suffix `.~P~` at the end. The last block of code is meant to hunt down and delete all possible browser cache directories:

```
cd $backup$direct
find .kde .opera .netscape .mozilla .phoenix .galeon .Arena \
        \( -name Cache -or -name cache -or -name cache4 \) \
        -exec rm -rf \{\} \; >& /dev/null
cd -
```

These directories are also holders of transient information, which can quickly fill up the corresponding backup directory with unwanted data. Since they are further down in the directory tree of the account, they cannot be excluded in the `tar` command, so we take care of them here.

This completes the explanation of the user-data backup scripts. After you install them in your server, you should run the `myrror-user` script manually by the first time, so that the backup is created. Since there are no snap files when the script is run by the first time, a complete backup is made, rather than an incremental one, and the snap files are created. Whenever you want to make a fresh backup of a certain account, all you have to do is to delete or move elsewhere the snap file of that account in the directory `/root/var/myrror/`. Once more, in order to cause the backup to run regularly, you should edit the root crontab with the command `crontab -e`, in order to include in it a line such as the one below:

```
#
# Backup the user and group accounts.
30 */4 * * * /bin/tcsh /root/bin/myrror-user
```

In this case the first five arguments determine that the `myrror-user` script will run at 30 minutes of the hour, every four hours. This is much more frequent than the system backup, running six times a day instead of just once. This is about as often as you can run the script in a system with several hundred users without any system overload. If the backup period is too short and the number of accounts very large, one backup cycle may not end in time, causing it to overlap with the next, but the file locking mechanism will prevent the second backup from starting and hence will prevent any such problems.

6.2 Multiple Swaps and RAID Arrays

Even after we finish installing all the backup filesystems in the cluster server, there still are some unused partitions left over. These are the partitions in the backup system disk that correspond to the partitions in the system disk which are used for swap, for the temporary filesystem `/tmp/` and possibly for the scratch filesystem `/temp/`. Since due to their roles in the system none of these partitions is supposed to hold permanent information, none of them need a backup. Instead of using the remaining available partitions in the backup disk as backups of their counterparts in the system disk, we will just add them to the system in such a way that we will not only double the space available for each one of these functions, but will also double the speed with which the disk content can be accessed.

6.2.1 Interleaved Swap Partitions

It is a very simple thing to install a second swap partition, but we will do it here in a special way, so that the two swap partitions will be used by the kernel in *interleaved* mode. The partition `/dev/sda2` is already in use by the system for swap, because we configured it that way during the initial installation of the system. You should be able to verify this using the command `swapon -s` or the command `cat /proc/swaps`, both should give the same output, which should look something like this:

```
Filename                        Type            Size    Used    Priority
/dev/sda2                       partition       1000440 0       -1
```

Note that you can get from this output information about the size and usage level of the swap partition, given in kB. In this example it is a 1024 MB swap partition. Note also the default priority of -1 given to this swap partition. This partition was formatted as swap space during the initial installation of the system. In order to format the other available swap partition, which is `/dev/sdb2`, you can use the `mkswap` command, giving the partition device as an argument:

```
mkswap /dev/sdb2
```

This is the correct command assuming you are using one of the standard partitioning schemes given before as examples. Be *very careful* not to make any mistakes in this command line, if you use the wrong partition device the contents of that other partition will be *immediately destroyed*. Having formatted the second swap partition, you may now put it in use manually, with the command

```
swapon /dev/sdb2
```

You may then verify that you now have two active swap partitions using again the `swapon -s` or `cat /proc/swaps` commands. This time you should see something like this:

```
Filename                        Type            Size    Used    Priority
/dev/sda2                       partition       1000440 0       -1
/dev/sdb2                       partition       1000440 0       -2
```

where we once more use 1024 MB swap partitions as an example. Note that the priorities -1 and -2 were automatically attributed to the first two partitions which were activated. This means that the two partitions are being used in *linear* mode, which in turn means that the second one will be used only after the first one is full. We are about to change that, but first let us configure the system to activate this second partition automatically during boot, just like the first one. Is you look at the `/etc/fstab` file, you will find there an entry for the first swap partition, although it does not really contain a filesystem,

```
/dev/sda2       none            swap    sw                              0       0
```

This entry causes the first swap partition to be activated during the boot by the system autoconfiguration scripts. In order to have the system activate the second swap partition as well, all you have to do is to add a line for it in this file,

```
/dev/sda2       none            swap    sw                              0       0
/dev/sdb2       none            swap    sw                              0       0
```

This will automatically add the two partitions with priorities -1 and -2 during boot, just as they were when you activated the second partition manually, doubling the available swap space. There is, however, a better way to do things. The two swap partitions can be used, instead of in linear mode, in what is called *round-robin* or *interleaved* mode. In this better mode of operation the system will write memory pages to the two swap partitions in an alternating way, first to one, then to the other, then back to the first one, and so on. This is quite unlike the linear mode, in which it writes all pages to a single partition until it is full. Because the two swap partitions are located in different disks, and because the main bottleneck for access to disk content is the physical speed of the disks, not the SCSI interfaces or the system bus, this will result in a speed for writing and reading swap pages which is about twice as large as the corresponding speed in the case of the linear mode of operation. In this way, adding the second swap partition not only doubles the available swap space, but it also doubles the speed of access to the swap space. In order to instruct the kernel to use the swap partitions in this way, all you have to do is to attribute to them the same priority level, which can be zero or any positive integer. You can do this including an option in the swap line of the `fstab` file, which you should now change to look something like this:

```
/dev/sda2       none            swap    sw,pri=0                        0       0
/dev/sdb2       none            swap    sw,pri=0                        0       0
```

In order to have the swap partitions operating in this new way, after you finished making the changes to the `fstab` file, you must first deactivate and then reactivate all the swap partitions. In order to deactivate all the swap partitions listed in the `fstab` file, you can use the command

```
swapoff -a
```

after which the `cat /proc/swaps` command should show no entries. In order to activate again all the swap partitions listed in the `fstab` file, you can use the command

```
swapon -a
```

after which the `cat /proc/swaps` command should show something like this:

```
Filename                        Type            Size    Used    Priority
/dev/sda2                       partition       1000440 0       0
/dev/sdb2                       partition       1000440 0       0
```

If you wait for a while until the swap space begins to get used, you should see approximately equal usage in both partitions, differently from the case of the linear mode, in which only the first partition will have any usage at first, and the second one will only show any usage after the first one is full, something that, by the way, should very seldom happen. Here is an example of interleaved swap space in use, this time using two 2048 MB swap partitions:

```
Filename                        Type            Size    Used    Priority
/dev/sda2                       partition       2000888 7824    0
/dev/sdb2                       partition       2000888 7808    0
```

The difference between the two usages is always a multiple of the page size, which is 4 kB in the i386 architecture. It may not be exactly equal to either 0 or 1 pages because, although pages are written out to swap space in this round-robin mode, single pages may become needed and may be taken off the swap area and back into main memory. As you may have guessed from this output of the `swapon -s` command, it is possible to have other types of swap areas besides disk partitions. In fact, you may also use swap files, files within a filesystem which are formatted and used as swap space. This will find use when we discuss remote swapping through the network later on, in association with the NBD (Network Block Device). Just by way of illustration of the functioning of non-interleaved swap areas, here is an example of a set of four 4 MB swap files used in normal, non-interleaved mode on a remote-boot terminal, as seen at different times. Some time after activation one sees this:

```
Filename                        Type            Size    Used    Priority
/nbd/swapfile1                  file            4088    760     -1
/nbd/swapfile2                  file            4088    0       -2
/nbd/swapfile3                  file            4088    0       -3
/nbd/swapfile4                  file            4088    0       -4
```

Note that there is only moderate occupation, and only in the first one of the four swap files. Later on, however, after the activity in the system increases the memory requirements, one sees this:

```
Filename                        Type            Size    Used    Priority
/nbd/swapfile1                  file            4088    4088    -1
/nbd/swapfile2                  file            4088    980     -2
/nbd/swapfile3                  file            4088    0       -3
/nbd/swapfile4                  file            4088    0       -4
```

The first swap file has been completely filled up at this point, and there is some occupation in the second one. Still later on, when more system activity has caused unused pages of memory to be swapped out, one sees that the first two swap files are full and the third in partial use:

```
Filename                        Type            Size    Used    Priority
/nbd/swapfile1                  file            4088    4088    -1
/nbd/swapfile2                  file            4088    4088    -2
/nbd/swapfile3                  file            4088    1272    -3
/nbd/swapfile4                  file            4088    0       -4
```

These swap files were made purposefully very small, so one can easily see the distribution of the occupation among them according to their assigned priorities. Since in the case of swap through a 100 Mbps network, such as in this example, the access speed bottleneck is the network itself, there is no sense in having and interleaving several swap areas, unless you can do that through several different network cards feeding non-competing network connections. With a Gbit network, on the other hand, it may make sense to interleave up to four swap files over the network, so long as they are located in different physical disks, possibly in more than one disk server.

6.2.2 Assembling RAID-0 Arrays

The round-robin or interleaved utilization of the swap areas described in the previous subsection is similar to what happens when one uses the MD (Multiple Device) kernel driver to assemble RAID-0 arrays of disk partitions into a single block device. The MD driver of the kernel creates virtual block devices, with device nodes such as `/dev/md0` and `/dev/md1`, which correspond in fact to writing and reading blocks of data to several other block devices such as disk partitions, which we will take as our example here. In this way several smaller block devices, for example several small hard disks, can be united to hold a single large filesystem. This union of the smaller block devices can be made in several different ways, some of them involving a degree of redundancy in order to improve the reliability of the MD device.

The *linear mode* of the MD device operates in a way similar to the way in which swap space usually operates, without interleaving: every disk partition participating in the multiple device is used in turn until it fills up. The *RAID-0* or *stripe* mode operates in round-robin fashion, just like interleaved swap. While the linear mode only adds up the available space of the participating partitions, the RAID-0 mode also improves access speed performance, by up to a factor equal to the number of participating partitions,

and up to the point where some other bottleneck sets in, assuming that the partitions are all in different disks and that all disks involved have the same performance. The remaining modes all provide some form of redundancy for the disk content. The *RAID-1* mode works by mirroring, repeating the content of one disk in a second disk. The *RAID-4* and *RAID-5* modes provide data redundancy by more sophisticated means, involving parity-based algorithms, and are useful mostly when there are several disks on the array, not just two.

Although it may seem that a RAID-1 MD device with two disks is equivalent to the automatic backup strategy for the system disk described in Section 6.1, on closer examination one can see that this is not quite so. In such a RAID-1 device all updates on the backup device are instantaneous, when something changes in the main device, allowing for no time for recovery from eventual mistakes. Most mistakes and accidents that occur during system maintenance are noticed fairly quickly, mostly immediately, and usually a 24-hour backup cycle allows for plenty of time for recovery before the mistake gets commited to the backup copy as well. On a backup system based on a RAID-1 array with two disks, however, any mistake or accident in the main device gets commited immediately to the backup device, allowing for no chance of recovery. Besides, having to boot from an MD device unnecessarily complicates the operation of the system.

As a consequence of these considerations, of all these possible modes of operation of the MD driver the only one which will be of interest to us here is the RAID-0 mode, and then only for use in disk space meant as temporary space or scratch space. Let us proceed, then, to install a RAID-0 device for the `/tmp/` filesystem. The required support in the kernel is already available in the form of modules in the standard Debian kernels, so the first thing to do is to install the `raidtools2` package, which has the support tools needed for you to create and use MD devices. You can do this in the usual way,

```
apt-get install raidtools2
```

You should answer with yes to the question about starting the raid devices automatically at boot. If you need more information about the various types of RAID array, look at the documentation of this package, in the directory `/usr/share/doc/raidtools2/`. After the installation of the package you should add its executables to the hash table of the shell executable search path, executing the shell command

```
rehash
```

You must now install a configuration file for the raid tools in `/etc/raidtab`, which will determine the characteristics of each MD device existing in the system, including the list of partitions participating in each one. An entry in this file for a RAID-0 array looks like this:

```
# compatible (old) RAID-0 setup for /tmp:
#
raiddev /dev/md0
    raid-level                0
    nr-raid-disks             2
    persistent-superblock     1
    chunk-size                8
#
    device                    /dev/sda3
    raid-disk                 0
    device                    /dev/sdb3
    raid-disk                 1
```

Copies of this entry are available in the files `raidtab.PMC` and `raidtab.TRM` which can be found in the resources area, in the subdirectory `etc/`. It defines a RAID-0 array of two partitions to be used for the `/tmp/` filesystem of the server, and can be used both in the case of the server of compute nodes and in the case the of server of remote-boot terminals. With the `raidtab` file correctly configured and in place in your server, we are ready to create the RAID array. This is a one-time operation, which you must perform just once for every new array you create. In order to do this you must first go to single-user mode, because the current `/tmp/` filesystem will have to be unmounted and destroyed so its partition can be used in this array. Therefore, go to single-user mode with the command `shutdown now` and then unmount the `/tmp/` filesystem. You may then try to create the array with the command

```
mkraid /dev/md0
```

just to see what happens. Since one of the two partitions involved is already formatted, the command should detect this and refuse to run. You have two options here, in order to overcome this difficulty. You may run the command `mkraid` with the option `-f` and then follow the instructions which appear, in order to tell the program to force the creation of the array despite the content detected. Alternatively you may clean up the problematic partition using a command such as

```
dd if=/dev/zero of=/dev/sda3
```

to fill it with binary zeros. Just wait until the program `dd` exits in error because the device is full. Whatever you choose to do, do it *very carefully*, because a mistake here can destroy some partition which is essential for the operation of your server. Note that the example above is just that, an example, and is correct only if you are using the partitioning schemes proposed previously. If the other partition that is part of this array has also been formatted before, you can repeat this cleaning process for it as well,

```
dd if=/dev/zero of=/dev/sdb3
```

Once you get the `mkraid` command to run successfully and it creates the array, you can activate the MD device with the command

```
/etc/init.d/raid2 start
```

This initialization script will be automatically executed every time the system boots, of course, this is the only time you will have to do this manually. You can verify the status of the MD device just created looking at the `/proc/mdstat` file, for example with `cat`. The output should look something like this:

```
Personalities : [raid0]
read_ahead 1024 sectors
md0 : active raid0 sdb3[1] sda3[0]
      1001344 blocks 8k chunks

unused devices: <none>
```

At this point the MD device `/dev/md0` has been made available for use by the system, but so far it has no content. The next thing to do is to create a filesystem structure in this MD device, which you can do just as you would do for any other block device, such as a disk partition,

```
mke2fs -j /dev/md0
```

This will create an ext3 filesystem in the MD device `/dev/md0`. You should also label the new filesystem as usual,

```
e2label /dev/md0 /tmp
```

You may now mount this device as the `/tmp/` filesystem. In order to do this, let us first change the line in the `/etc/fstab` file which corresponds to this filesystem, so that it will be mounted correctly during boot, since this line is currently wrong due to the changes we just made in the system. This line should now read

```
/dev/md0        /tmp            ext3    rw                              0       2
```

The files `fstab.2.PMC` and `fstab.2.TRM` in the subdirectory `etc/` of the resources area have this correction in place. With this change in place you can mount the new `/tmp/` filesystem with the usual command,

```
mount /tmp
```

If you have in your server partitions reserved for a scratch filesystem `/temp/`, you can now repeat the whole process used to create the `/dev/md0` device for the `/tmp/` filesystem in order to create a `/dev/md1` device for this scratch filesystem. The steps are all the same, and you must start by adding a second entry to your `raidtab` file. This new entry should look like this:

```
# compatible (old) RAID-0 setup for /temp:
#
raiddev /dev/md1
    raid-level                0
    nr-raid-disks             2
    persistent-superblock     1
    chunk-size                8
#
    device                    /dev/sda11
    raid-disk                 0
    device                    /dev/sdb11
    raid-disk                 1
```

A copy of this entry is available in the file `raidtab.PMC` in the resources area. After you create and activate this second MD device, the file `/proc/mdstat` file will show something like this:

```
Personalities : [raid0]
read_ahead 1024 sectors
md1 : active raid0 sdb11[1] sda11[0]
      11880192 blocks 8k chunks

md0 : active raid0 sdb3[1] sda3[0]
      499584 blocks 8k chunks

unused devices: <none>
```

In order to have this new device mounted automatically during the boot you should create the mount point `/temp/` and add to the `fstab` file of the server a line such as this one:

```
/dev/md1        /temp           ext3    rw                              0       2
```

You may also mount it manually with `mount /temp`, of course. With all this in place all the disk partitions available in your server will be in use and its disk configuration finished. You now have swap space, temporary disk space and possibly scratch disk space which are both twice as large and twice as fast as before. In the case of the swap space it is important to emphasize, however, that even at such doubled speed the I/O access is still much, much slower than the access to physical RAM, so that processes which start to paginate intensely will still bring the system to a grinding halt. You

may reboot the machine to see it set itself up correctly from scratch, or you can simply return to multi-user mode typing `[Ctrl]-[D]`.

A final word is due as to what happens to these interleaved areas if one of the disks fails. In the case of the swap areas the system takes care of itself, on a reboot after the disk failure the missing swap area will simply not be used, and the other one will be used in linear mode. On the other hand, the content of the RAID-0 array will be completely lost and the devices will no longer be available, so their mounting will fail in the first boot after the disk failure. In order to deal with this you must boot the system in single-user mode and comment out in the `fstab` file entry corresponding to the failed MD device. Then you should reboot the system once more in single-user mode and then either format the single remaining partition, making the necessary adjustments in the `fstab` file in order to put it in use, or reconstruct the MD device from scratch, using a new disk to replace the failed one.

6.3 Resource Management

In any multiple-user system it is important to have a certain amount of limitation and control of the access to the existing resources by each individual user. This is necessary both to ensure fair access to the resources by all the users and to prevent single users from disturbing the operation of system by overloading one of the resources. Regarding possible disturbances in the operation of the system, the three most common problems are processor usage overload, memory usage overload and the accidental filling up of a filesystem. This makes it necessary to have in the system a basic level of control of the usage of these basic resources. Fortunately, it is quite simple to implement structures to limit and control the usage of these resources by the users.

There are some other less common problems that may also happen, such as a user writing and running a program that spawns sub-processes and that, due to some bug, end up creating a very large number of processes, preventing other processes from starting and effectively locking up the system. Another one is a user process on a parallel processing cluster that writes too much data out to disk too fast, and that may bring down the private network of a PMC. In some cases it is also possible to put up structures to prevent such occurrences, other times it may be necessary to count with a minimum level of discipline on the part of the users, as well as to guarantee that they have a sufficient amount of technical knowledge in order to be able to use the system rationally.

In many occasions promoting the appropriate training of the users in the use of the system may be a far better solution than simply installing enforcement structures. This is specially true in small to middle-size systems, in which most users know each other to some extent. In some cases very large-scale systems, with many users that do not know each other at all, may need a more specialized treatment of some of these aspects. We will describe here only the most basic forms of usage control of the processor, of the memory and of disk space occupation, which are usually sufficient for small to middle-size clusters, and certainly useful in all cases.

6.3.1 Processor Usage Limitation

The two most common forms of processor overload are users running on the background, at normal interactive priority, a large number of intensive jobs, and buggy programs that go berserk, detach from the user's terminal and stay running on the background, fully loading the CPU without the user even noticing it before exiting the system. This second form may be just an annoying waste of resources, not too troublesome unless several processes of such programs gone berserk are running at the same time. Even a fairly small number of intensive jobs running at normal priority can degrade in a significant way the response of the system to any interactive commands. Since the backgrounded jobs are running with the same priority as the interactive commands, the response of the shell can become really sluggish, and the system daemon processes and the associated services may have their performances severely degraded as well. Events of this type are

sometimes referred to as system *trashing*.

Note that processes which are inactive most of the time, such as system daemons, are never a problem. It is normal for any Unix or Linux system to be running many tens, possibly a few hundred such processes at the same time. What counts for our analysis here are the processor-intensive processes which are typically run by users on the background. The submission of a large number of background jobs by the users is usually a sign of the insufficiency of the available number-crunching computational resources, as well as of the existence of some (possibly fierce) competition for those resources among the users. However, it may also be just a mistake by an unexperienced user. Programs gone berserk in the background are very often caused by users hard-crashing X11 application, typically by forcefully killing their windows rather than exiting them in the legal way which usually exists within the programs. Sometimes, however, this is the only way to eliminate the window of a program, due to the fact the it is buggy and has already crashed, but without closing the window. Users can use the `kill` command to kill off such runaway programs from within a shell, so long as they are aware that they are there and know how to do it. This is one instance in which even a minimal level of user training may be very helpful.

On older Unix systems, running on the much slower processors of that time, the submission of too large a number of simultaneous CPU-intensive processes to a single time-sharing processor used to cause the processor to work in a less efficient way due to the processing overhead involved in context switching, that is, passing the use of the processor from one user process to another. What this means is that at some point the system started to spend too much time managing the concurrent processes, as compared to the time it spent actually running the processes of the users. Linux seems to deal with this very well, and since it typically runs on the very fast processors common these days, you have to go to hundreds of CPU-intensive concurrent processes in order to see a measurable amount of system performance degradation, to be compared to the half dozen intensive processes that traditional experience showed to be the maximum that those older flavors of Unix could handle comfortably. About half a dozen is also the number of intensive programs running with normal priority at which the interactive sluggishness of the system may become unbearable.

In modern systems other problems show up well before any significant amount of system inefficiency due to context switching can be felt. One such problem is the fact that all the concurrent processes must fit into the memory of the machine if one wants them all to run with optimal efficiency. If processes start to get swapped in and out with any frequency or, even worse, if some processes start paginating, then the performance of the machine will be very severely degraded. The fact that the RAM of the machine must be shared among the concurrent processes is even more acutely felt on remotely-booted compute nodes with no swap, since in this case the whole set of processes absolutely *must* fit in memory if they are to run at all. We see therefore that the relation of the available RAM on the machine and the memory requirements of each program will

effectively limit the number of concurrent intensive processes running on it. Limiting the maximum RAM size of a job to be below the available RAM is always a necessity, but if each compute node is expected to run more than a job in a routine fashion, setting a more stringent limit may be in order. In the next subsection we will discuss memory limitations for processes.

The other problem relates to the performance expectancy on the part of the user. The full performance of a single node is what the user typically expects to get, it is the intuitive expectation that most users have. It the user actually gets much less than this in a routine way, due to time sharing on a heavily loaded system, frustration ensues and the work being done will suffer. About four or five concurrent jobs, meaning that each user gets from 20 % to 25 % of the performance of a node, is as much as one should accept for the constant and routine operation of the system. If a cluster of compute nodes runs constantly at load 4 or more on all the nodes, this is a sure-fire diagnostic that the group is under-supplied in terms of processing power. In this case consideration should be given to the possibility of expanding or upgrading the cluster. A much more common occurrence are temporary crunches during which the load is very high, interspersed with periods of relative idleness. Although this may also indicate that the group is short of processing resources, depending on the type of work that it is involved with, in this case a simple rationalization of the use of the machine may be enough to solve the problem.

All the basic problems related to CPU overload can be solved in an excellent way by a nice piece of software called the *Auto-Nice Daemon* (AND). It implements a simple and elegant idea that makes it really easy to solve all the major problems of this type. What it does is to monitor all processes and change the *nice level* of various user processes in the system, according to certain criteria contained in its configuration files. The nice level is a number that enters into the calculations that the kernel makes in order to give appropriate running priority for each one of the processes running in the system. The `and` daemon can also send signals to processes, in order to stop or kill them. With the use of this daemon sluggish interactive response is eliminated by its action of decreasing the priority of long-running CPU-intensive processes, an operation known as *renicing*. Besides, programs gone berserk in the background can be dealt with by simply killing them by means of a signal, after they have used up a certain amount CPU time. The daemon can also renice processes to different levels depending on the user and group the process belong to, allowing the system manager to give different priority levels to different users or groups of users.

The `and` daemon is universally useful and should be installed in all the machines of your cluster, including servers, terminals and compute nodes. In fact, it should already be installed in all the machines of your cluster because it is included in our basic list of packages, in the file `common-package-list`. If you find you have to install it somewhere, you of course can do this using the `apt-get` command in the usual way,

```
apt-get install and
```

To install it on a cluster you should use the tools `apt.chroot` and `multi-apt.chroot`, of course. The package comes with a nice default configuration, so that there need be no rush to reconfigure it right away. The way in which it works is this: first of all, it leaves alone all processes owned by root and other system users, and therefore does not interfere with system programs; it establishes three CPU timeout levels, which by default are 2 minutes, 20 minutes and 1 hour; it established three nice levels, to which the programs will be taken after their CPU time exceeds each one of the three timeout levels; it the nice levels are negative, the daemon will understand that the corresponding signal should be sent to the process, instead of the renicing operation.

The standard nice value of programs running under normal priority is 0, and the largest possible value, corresponding to the lowest possible priority, is 19. The larger the nice level, the lower the priority of the process. Any user can change the nice value of his own programs using the `renice` command, but normal users can only decrease the priority of the processes, they can never increase it. The `and` daemon also will never increase the priority of processes, but will only decrease them. The daemon will check the processes at regular intervals, by default at 60-second intervals. The configuration file `and.conf` in the `/etc/` directory is where the checking cycle is defined, as well as the three CPU timeouts. Another thing defined in this file is the relative priority of the three possible types of criterion for the action of the daemon: by user name, by group name and by command (or program) name. The default is `cug`, meaning first by command or program name, them by user name and last by group name. The criteria are defined in another configuration file, the file `and.priorities`. The first active line in this file defines the default renicing strategy. In the default configuration this line reads

```
#user   group   job             nice1   nice2   nice3
*       *       *               4       8       12
```

where we included a comment line with headers in order to show the meaning of the fields. The first three fields give the names of the user, the group and the command to which this rule applies. The wildcard `*` means that this default line applies to all users, all groups and all commands. The other three fields give the nice levels for each one of the three CPU timeouts. This default action consists of a moderate renicing of all long-running programs, in order to keep the system with a lively interactive response. A common example of a rule by command name is the one meant to keep berserk-prone programs under control, such as

```
#user   group   job             nice1   nice2   nice3
*       *       .*netscape.*    4       -9      -9
```

This will apply to all users and all groups, for any programs which contain the word `netscape` within their names. Note that one can use regular expressions in the command name field, such as the string `.*`, which matches any string of characters. Since

the second and third nice levels are negative, a command of this type found to be running for more that 20 minutes of CPU time will be sent signal 9, which is a fully-prejudiced kill, that is, a kill which is unrelated to the internal state of the program. Note that the relevant time here is not elapsed time, but CPU time, the time spent running on the processor. The program may stay up for as long as desired, so long as it does not spend more that 20 minutes of CPU time.

The sending of the signal by the `and` daemon is equivalent to the use in an interactive shell of the command `kill -9 <pid>` or `kill -KILL <pid>` to send the given signal to the process with a certain PID (Process Identification Number). The signal number 9 has the symbolic name KILL. Other signals useful for use within the shell, and sometimes also in the configuration of the `and` daemon, are: signal number 15, which is the TERM signal, indicating that the program should terminate according to its own internal rules; signal number 19, which is the STOP signal, indicating that the program should temporarily stop processing but should not terminate, staying in memory; and signal number 18, which is the CONT signal, indicating that a stopped process should resume execution. Unfortunately, the signals TERM and CONT are not very useful within the context of the `and` daemon. The signal STOP can be used to stop processes after they run for a while. For example, in order to inhibit the submission of long intensive jobs in a general-purpose server where users should not run this kind of job at all, one may use a default line such as

```
#user   group   job             nice1   nice2   nice3
*       *       *               12      19      -19
```

This will send a STOP signal to any user processes which run for more than 1 hour of CPU time, having first severely reniced them, first after 2 minutes and then after 20 minutes. These processes will show up in the output of the `ps` command with status `T`, thus informing the user (and the manager) of what has happened. Here is an example illustrating a criterion by user name. This line is meant for running the jobs of a certain user at very low priority,

```
#user   group   job             nice1   nice2   nice3
luser   *       *               12      16      19
```

This kind of rule can be used to define low-priority users on a processing cluster, for example users which are not part of the group that owns the cluster, but which are allowed to use some idle time it may have. Another possible use it to punish delinquent users, which insist on running long intensive jobs on machines in which this is not allowed. You may set up a rule to first radically renice their jobs after two minutes, then to kill them after 20 minutes,

```
#user   group   job             nice1   nice2   nice3
duser   *       *               19      -9      -9
```

Of course, this is a rather extreme last-recourse measure that should be avoided as much as possible. Other things may be done with the rules of the `and` daemon, such as keeping screen savers and compilers under control, all included in the default configuration. Complete configuration files `and.priorities`, with the examples above and the addition of many other berserk-prone programs you should keep an eye on, can be found in the resources area, in the subdirectories `etc/`, `pmc/0000/etc/` and `trm/00/etc/`, for the server, the compute nodes and the remote-boot terminals respectively. The files for the server and the terminals are identical, but the file for the compute nodes is different in that it uses a somewhat softer default rule,

```
#user   group   job             nice1   nice2   nice3
*       *       *               4       6       8
```

This is adjusted so that there is a wider gap between the default priority and the low priority mentioned above for users external to the group. With these priorities one high-priority job and one low-priority job would share time at a 80 % – 20 % rate when competing only with each other for CPU time.

Using the possibility of configuring the priorities and CPU timeouts, you should have no difficulty in finding a configuration of the `and` daemon that satisfies all your needs in terms of control of CPU usage. Usually the default CPU timeouts are quite satisfactory and all you have to do is to configure the `and.priorities` file. After you edit this file, you should send a HUP signal (signal number 1) to the daemon in order to instruct it to reload its configurations. You can first find the PID of the daemon with a command pipeline such as

```
ps aux | grep and | grep -v grep
```

then send the signal do the daemon with the command

```
kill -HUP <pid>
```

or, equivalently, you can use the much simpler command

```
/etc/init.d/and reload
```

You may use `kill -1` as well. When configuring the compute nodes of a cluster, you must remember that the default configuration files of all the nodes are linked together, and that this allows you to change them all at once if you are careful about how you do it. You can do this on the server, just go to the directory `/pmc/0000/etc/` of the first node and make a copy of the file with a temporary name, using

```
cp -p and.priorities and.priorities.N
```

You may then edit the file `and.priorities.N` with your favorite editor and fix it the way you want. After it is finished you can replace the content of the file which is visible to all nodes through its multiple hard links with a command such as

```
cp -pf and.priorities.N and.priorities
```

You should then delete the `and.priorities.N` file, as well as any backup files left by the editor. Last, you should send the HUP signal to the daemons on all the nodes. A convenient way to do this is to use the distributed shell like this, for the case of the PMC nodes:

```
dsh -g @pmcnodes /etc/init.d/and reload
```

With this structure in place the system will pretty much take care of itself with regards to CPU usage. The only type of maintenance that may be needed once in a while is the eventual inclusion of a low-priority user in the configuration files. This can also be automated to some extent, as will be discussed in later chapters.

We end this subsection with some comments on queueing systems. The more traditional way to manage jobs in a resource-starved environment, in terms of processing power, is the installation of a system of batch job queues. In older Unix systems the software usually used for this purpose was the Network Queueing System (NQS), unfortunately most of the time in some non-open-source version. Although there is in fact an open-source implementation, it seems that its development and maintenance have stopped, and it is not available in the Debian package format. While the NQS system was common, one might even say inevitable, in the older Unix high-performance computer centers, it does not seem to play a very important role in the Linux clusters common today.

While in this type of queueing structure one can do almost anything in terms of controlling the use of the processors and the distribution of resources among the users, it does also have the disadvantage of being somewhat over-structured and rather large, quite unlike the typical Unix-style tool or subsystem. This kind of structure usually brings about with it quite a bit of additional administrative work. While something like this may be inevitable in very large systems with a large number of users competing for relatively scarce processing resources, we feel that it is not really appropriate for small to middle-sized clusters, in an age characterized more and more by processing resources which are not particularly expensive or scarce.

There is an interesting alternative form of queueing system, distributed by the GNU-queue project. This software has some interesting features and is designed in a way which allows it to merge very easily into a typical Linux environment, but it seems to lack the capability to implement controlled resource sharing among the users. In addition to this, it also seems to be developing very slowly, if at all. This software is available in the Debian distribution, the name of the package is `queue` and you might want to

install it in order to play around with it for a while. A sufficient capability, in terms of controlled resource sharing, could be achieved by some fairly simple developments in the `and` daemon. This subject will be discussed later, in the "wish list" chapter, dedicated to the things we would like to see available for clusters.

One interesting aspect of the GNU-queue project, apparently now yet fully implemented, is the capability to do *checkpointing*, that is, to cause jobs which are killed by a signal to write to disk an image which is restartable from the point at which the process was stopped. This can be used to migrate a process from one node of a cluster to another, allowing one to dynamically balance the load on a compute cluster. This capability would come in very handy for optimizing the use of clusters, and it is not yet available in a simple enough form. The OpenMosix project does have this capability, but it must be implemented by means of kernel patches and seems to be also a bit over-structured.

6.3.2 Memory Usage Limitation

Memory over-usage is one of the most serious operating problems a user can cause in a system which uses swap. If a user program grows in memory to the point where it does not fit into RAM anymore and, as a consequence, the system starts paginating in an intense way, it will make the whole system slow down to a crawl, stopping everything from running and possibly even making it quite difficult for the operator to get into the system and kill the process responsible for the problem. If this happens in a central cluster server, crucial network services may become inoperant and the event may bring down the whole cluster. Contrastingly, in systems that do not use swap the growing process should simply crash itself, with an out-of-memory error message, disturbing the operation of the system only for a short time just before its crash. On the other hand, if a diskless system uses swap through the network, the crisis can bring down not only that system but its network connection as well, and this may even make the remote-boot system hang or crash.

In this day and age RAM is a relatively cheap commodity and the use of swap should be restricted only to allowing idle programs to reside in the address space of the system, freeing RAM space for other active programs. The pagination of intensive processes to and from swap should always be avoided. If you need more address space in our system, do *not* buy more disk to put swap on, buy more memory modules. In order to avoid intensive pagination, there should be memory usage limits for the user processes. Fortunately it is quite simple to implement this, using the standard PAM (Pluggable Authorization Modules) authorization structure. It is very important to have this implemented in all your servers, but it is also a good idea to have it implemented in the compute nodes and remote-boot terminals as well. The relevant configuration file resides in the directory `/etc/security/` and is named `limits.conf`. It can be used to implements usage limits for several system resources. In order for it to be active, a configuration line such as

```
session         required        pam_limits.so
```

should be included in the files within the `/etc/pam.d/` directory that are related to programs involved with starting user sessions in the system. These are at least the files `login`, `su`, `ssh`, `xdm`, `gdm`, `gdm-autologin`, `kdm`, `kdm-np` and `cron` that you may find within that directory. If you do not find some of them, that is not a problem, it just means that the corresponding program has not been installed. Since some of the files may have this line commented out of absent, you should check them all and fix them as required. Do this in all the systems.

By default there are no limits implemented in the `limits.conf` file, but you should read it anyway, since there is some documentation within it. Much more complete documentation for the PAM subsystem can be found in the directory `/usr/share/doc/libpam-doc/`, which is made available by the package `libpam-doc` which is already included in our lists of packages for servers and terminals. The limitation of memory usage on a server with 1 GB of RAM could be implemented by the inclusion in this file of the lines

```
#
# Limiting memory usage to 512 MB (RAM/2) per process.
#<domain>       <type>  <item>          <value>
*               hard    as              524288
*               hard    rss             524288
```

The idea here is to implement only a moderate memory usage limitation, to prevent the types of incident described before. In this case we are limiting memory usage to half of the available RAM on the machine. Each line contains four fields, which are indicated by the comment line containing the headers. The first field determines which user or group the rule applies to, in this case the use of the wildcard `*` means that it is a default line, applying to all non-system processes. The second field gives the type of rule, `hard` means that the limit is an absolute one, enforced by the kernel. The third field gives the resources which are being limited, in this case the address space and the resident set size. The fourth and last field gives the limit, in these cases in kB units.

By and large, the address space relates to the amount of *virtual memory* that the process can use, which includes both real memory occupation and pages of memory written out to the swap area. The resident set size is just the amount of real physical memory used by the program. By limiting the virtual memory usage to be below the actual physical memory existing in the system, we are preventing the process from paginating, at least if it is the only such process running on the machine. In any case, this should make it very unlikely that a user process will ever cause the system to start paginating in an intense way. As one can see in the documentation contained in the file, there are other types of memory usage that can also be limited, but these two are the ones relevant for our purposes here. Since Linux manages memory in a rather

complex (and very efficient) way, it is not always very easy to calculate or understand the values that each one of these types of memory occupation will have. Unfortunately the documentation on this issue is rather scant, scattered in various places and not too complete.

Although the implementation of memory usage limitations is less important on compute nodes and remote-boot terminals, it is not a bad idea to do it in those cases as well. In compute nodes of a PMC, having 256 MB of RAM each, one could use the following lines

```
#
# Limiting memory usage to 240 MB (RAM - 16 MB) per process.
#<domain>       <type>  <item>          <value>
*               hard    as              245760
*               hard    rss             245760
```

Note that we are reserving only 16 MB for the system, including the kernel and the system daemons. This is quite enough on a compute node, which does not run an X11 server. In nodes with larger memory you would still just subtract the same 16 MB from it. For a remote-boot terminal with 128 MB of RAM one could use the following lines,

```
#
# Limiting memory usage to 96 MB (RAM - 32 MB) per process.
#<domain>       <type>  <item>          <value>
*               hard    as              98304
*               hard    rss             98304
```

in which we reserve 32 MB for the system. This should be enough even for terminals having 19-inch monitors, with their X11 servers running with fairly high resolution modes, but in very extreme cases it may be a good idea to increase this reserved amount to 48 MB or even to 64 MB. Complete files with these configuration can be found in the resources area, in the subdirectories `etc/security/`, for the case of the server, and `pmc/0000/etc/security/` or `trm/00/etc/security/` for the cases of compute nodes or remote-boot terminals respectively. In this last case there is also a file `limits.conf.LARGERAM`, meant for a terminal with 256 MB or RAM.

A word of warning should be given here that some programs may not be able to deal properly with a system with memory usage limitations. This is very rarely the case, but it may happen. One known example is the LISP interpreter from CMU (Carnegie Mellon University). The version contained in the package `cmucl` distributed in the "woody" version of the Debian distribution will refuse to run if there are any memory usage limitations. The command line option which is supposed to deal with this seem not to work either. This is just a defect of that software, however, and not any kind of unsurmountable problem. For example, the GNU Common LISP interpreter in the

package `clisp` works perfectly in systems with memory usage limitations, as does, in fact, the newer version of the `cmucl` package distributed in the "sarge" version of the Debian distribution, in which the problem seems to have been solved.

It is also possible to limit the number of processes that each user may run at the same time, using the methods which were explained here for the case of memory usage limitation. Although this is not usually necessary, it may be useful in some circumstances. In all the files in the resources area that were mentioned above you will find an extra block containing a line that implements this limitation,

```
#
# Limit the number of processes as a safety measure.
#<domain>       <type>  <item>          <value>
*               hard    nproc           100
```

This rule limits any non-system user to having at most 100 processes running simultaneously on the system, a high limit, unlikely to ever be reached under normal circumstances, and meant only at avoiding accidents due to some bug in a program.

6.3.3 Disk Usage Limitation

Although in the the case of the home filesystem and other user areas the limitation of disk space occupation is basically a form of resource distribution among the users, in other filesystems it may become a necessity for system safety and stability. Any filesystem to which users have write access in any form should have disk quotas installed. This means that the root and `/usr/` filesystems do *not* need quotas, while the `/tmp/` and `/var/` filesystems do, as well as those that provide disk space for users, such as the `/home/` and possibly the `/temp/` filesystems. In each one of these filesystems the disk quotas have a specific function and the quota amounts should be chosen according to a correspondingly appropriate strategy. In terms of the reasons for the installation of disk quotas, of all these the only one requiring some explanation is the `/var/` filesystem, since in all the other cases it is obvious that the users can write to the filesystem.

The `/var/` filesystem does have a temporary subdirectory that the users can write to, but it can also hold files belonging to users due to operations involving *spooling* and *caching*. The temporary area is used, for example, to hold `vi` recovery files. The files involved in spooling operations are typically temporary and the two most common types of operation that use spooling are printing and outgoing mail. The incoming mail may also be stored in the `/var/` filesystem until it is read by the user, but this case can be handled much better by the strategy of mail reception directly to "maildir" directories located within the user accounts. The files involved in caching may not be temporary, in the sense that they will not be deleted automatically by the system once their function is accomplished. They may have to be manually cleaned up or changed to root ownership

once in a while. One common case of cached user files are TEX font files which are generated automatically by that software when they are needed.

Let us go through the components of the disk quota system. There are two versions of the system, the older and more traditional version 1, and a newer version 2 which uses a different format for the quota files. We will describe here the older version of the quota system, in the way of an example, but the general explanation of the structure applies to the newer version as well. First of all, the quota subsystem requires support in the kernel, but since all the quota control operations of the kernel happen on the disk server where the disks are physically located, this is necessary only on the cluster server. The quota support is already included in the default Debian kernels, so we do not have to worry about this. Although all the quota-related action happens on the server, information about quotas will be available to the users on all machines of the cluster, because there is a `rpc.rquotad` daemon which will run on the server and provide quota information to all machines, for those filesystems on the server which are exported to the other machines by the NFS protocol and mounted by them through the network.

Second, some support packages are needed. In order that the command `quota`, which allows users to look up their disk quotas, be available in all systems, the package `quota` should be installed in all of them. This package includes also the tools to activate and manage quotas on the server, as well as the `rpc.rquotad` daemon. There is very little to configure during the installation. If you do not yet have a mail structure installed on your server, accept the default of "no" to the question about running daily the application `warnquota`, since it would do its work through the mail system. By now it should already be routine that when you install packages on the compute nodes or remote-boot terminals after they have all been cloned from the first one, it should be done with the use of the tools `apt.chroot` and `multi-apt.chroot`. On the server there is a second support package to install, the package `quotatools`. This package contains a lower-level tool for manipulating quotas. Although it is not strictly necessary for the use of the quota system in the usual way, we will use this tool for some additional quota-related structures to be described in a little while.

Before we go on to describe how to configure and activate the quota system, a word is necessary about user and group quotas. These two types of quota are available, one takes in consideration, when calculating the total disk occupation, the user ownership of the files, while the other uses the group ownership for the same purpose. Usually the use of user quotas is much more common than that of group quotas, which in fact are seldom used. We will recommend here, however, a very unorthodox approach, in which *only* group quotas will be used. In a system that uses the strategy that each user has his own group, with the same name and identification number (a strategy which we also recommend), this approach is equivalent to the usual one from the point of view of each user. On the other hand, it allows for the creation of group accounts to be used by groups of users, so that each such group account has its own disk quotas, which are independent of the quotas of the individual users that are members of the group. This

is a significant capability to have in environments where small groups of users can be expected to work cooperatively in certain projects. In what follows we will use group quotas as our example, but the sequence of operations is the same if you decide to use user quotas, or both user and group quotas.

In order to configure and activate the quota system the first thing to do is to have those filesystems on which we want to install quotas mounted with the quota option turned on. There are two possible quota-related mount options, `usrquota` and `grpquota`, which will be used depending on the type of quota to be implemented. The way to do this is to include the appropriate option in the `fstab` file, using the options field. Using the `/tmp/` filesystem as an example, its line in the `fstab` file, which looks like

```
#<file system>  <mount point>   <type>  <options>              <dump>  <pass>
/dev/md0        /tmp            ext3    defaults               0       2
```

should become

```
#<file system>  <mount point>   <type>  <options>              <dump>  <pass>
/dev/md0        /tmp            ext3    defaults,grpquota      0       2
```

With this done, one can re-mount the filesystem with the quota option enabled, using the command

```
mount -o remount /tmp
```

Next we must create a quota file for that filesystem. The quota file is a binary file which must reside in the root of the filesystem and which will contain all the quota and current occupation information for each user who has some data in that filesystem. The name of the quota file depends on the type of quota which is being used, it will be `quota.user` for user quotas and `quota.group` for group quotas. These are the names for the case of the old type of quotas, for the newer type the names would be `aquota.user` and `aquota.group`. Note that it is not only a question of creating the file, one must also create its contents. The `quotacheck` command can do both things, if used with the appropriate options. It will scan the whole filesystem looking for the files, so it may take a while to run. It will create a record in the quota file for each user found to have some content in the filesystem, and will record in it the disk occupation of that user in the filesystem. In our example here one could do this with the command

```
quotacheck -v -g -m -c -F vfsold /tmp
```

This will create the file `/tmp/quota.group`, which will be protected against reading and writing by all users except root. The meanings of the options used are: `-v` means verbose, will cause the program to write an activity report to the screen; `-g` will

make the program calculate group quotas; `-m` will make the program work even if the filesystem is not mounted read-only, making it clear that this should only be done on a quiescent filesystem, possibly in single-user mode; `-c` instructs the program to write a new quota file, ignoring any old files; `-F vfsold` tells the program to use the older quota format, for the new one use the argument `vfsv0` instead of `vfsold`. If you want both user and group quotas, you should run the program twice, once with `-u` and once with `-g`. You may also create the quota file with the `touch` command and then run the `quotacheck` command without the `-F` option, since it will then get the type of quota from the name of the file. However, if you do this do not forget to protect the file, changing its mode with the `chmod` command or with the `hide` alias of our standard root shell configurations.

Once the filesystem is mounted with the quota option and the quota file exists in the root of the filesystem, we are almost done. If this is the first filesystem to have quotas enabled, you must still run the command `quotaon` with the appropriate option in order to activate the quota system, which can be done indirectly with the command

```
/etc/init.d/quota start
```

The quotas are now active for the `/tmp/` filesystem. From now on the kernel will keep track of all changes in the filesystem which incur in changes of occupation by any user, enforcing the limits and recording the changes in the quota file, which will change is a more or less continuous way. This same initialization script used with the argument `stop` will run the `quotaoff` command in order to disable the quota system, for example during a system shutdown. The quota system will be automatically activated for that filesystem in all future system boots, and whenever a system crash occurs the `quotacheck` command will run during the next boot, in order to fix the disk usage information. You may now repeat the procedure for each filesystem on which you want to have quotas, re-mounting each one with the appropriate quota options and creating the quota files with the `quotacheck` command. It should no longer be necessary, however, to start the quota system with the script `/etc/init.d/quota`, that should only be needed the first time around.

The only quota-related tool needed by common users is the command `quota`, which allows them to look up the state of their quotas and disk occupation in each relevant filesystem. The users can only look up their own quotas, but the root user may use this command to look at any user's quota. In order to do this the command should be used like this:

```
quota -vg <username>
```

The options `-v` and `-g` will make the command show all group quotas, even for filesystems in which the user is not over the quota. Besides the commands `quotacheck`, `quotaon` and `quotaoff` already mentioned before, the root user has access to two other

very useful quota administration tools, the commands `repquota` and `edquota`. The first of these takes the mount point of a filesystem as an argument and shows all the quotas and occupations in that filesystem. In order to see all entries, even the entries for those users that have no occupation at all in the filesystem, use the command like this:

```
repquota -vg <filesystem>
```

The `edquota` command takes an username as an argument and lets the root user edit all the quotas of that user. It uses the editor defined by the environment variable `EDITOR`, just like the `crontab` command discussed in Section 6.1, so in order to use the `emacs` editor to edit quotas you should first set that variable with

```
setenv EDITOR emacs
```

then make sure that the environment variable `DISPLAY` is set correctly so that the window of the editor will open up in your terminal, and finally, in order to edit the group quotas of a user, use the command

```
edquota -g <username>
```

If you are not using a X11 terminal but are running this on a text console instead, you can use `jed` instead of `emacs` for the `EDITOR` variable, or you can just unset the `DISPLAY` variable,

```
unsetenv DISPLAY
```

which will cause the `emacs` editor not to try to open an X11 window. The edquota command also has a very useful option `-p` which allows you to copy the quotas of a "prototype user", a paradigmatic account with a standard set of quotas. This is very useful when opening new accounts. If you have in the system an account with username `stdquota` holding the standard quotas, you can set all the quotas of a user to be this standard set with the command

```
edquota -p stdquota <username>
```

When used in this way the command will not use the editor given by the `EDITOR` variable and will not, therefore, try to open an X11 window. Let us now give some examples of how to choose the quotas for each type of filesystem. There are in fact several type of limitation that can be imposed. One can limit the use of block of disk space and of file inodes, which is effectively a limitation on the number of files that the user can have. Usually the usage of inodes is not an important issue and we will, therefore, not impose any quotas on inodes, but only on the block usage, which is what

in fact limits the usage of storage space. There are also *soft* and *hard* limits for both block usage and inode usage. The soft limits can be eventually surpassed by the user, for a limited period of time called the *grace* period, which by default is 7 days. The hard limits are enforced in an absolute way by the kernel, if they is reached further writes by the user to that filesystem are simply blocked. Writes are also blocked if the user is over only the soft quotas but the grace period has expired. Since the quotas on the system filesystems are meant just as a safety measure, they can be the same on all systems and for all users. For example, for the `/tmp/` filesystems we may use something like

```
Filesystem            blocks       soft       hard     inodes     soft     hard
/dev/md0                   0     120000     132000          0        0        0
```

as seen on the display of the `edquota` command, for a typical user. Note that the mount point itself does not appear in this display, what appears instead of it is the device holding the filesystem, in this case the RAID device `/dev/md0`, which forces you to know which device holds which filesystem in order to edit the quotas. The "blocks" and "inodes" entries show the corresponding occupations, there is none in this case, the other columns give the soft and hard quotas in either case. Quota values of zero indicate in fact the absence of limits. The numbers for the block limits are in units of 1024-byte blocks. The soft quota is about 120 MB, the important things about it are that it be enough for regular use of the system and smaller than the size of the filesystem; in this case it is about a quarter of the total size. The hard quota is 10 % larger that the soft quota, just to give the users some maneuvering space and time to react to correct the situation if they run overquota. The case of the `/var/` filesystems is significantly different, in this case one might use something like

```
Filesystem            blocks       soft       hard     inodes     soft     hard
/dev/sda5               1928       2000      20000        436        0        0
```

One can see here that this `/var/` filesystem is in the fifth partition of the first SCSI disk, corresponding to the device `sda5`. We see that in this case there is some occupation, in fact 436 files occupying about 1.9 MB of space. The soft quota is quite small, much smaller than the size of the filesystem, since there should be no permanent user occupation in these filesystems. The hard quota, however, is ten times larger, at about 20 MB, to allow the users to have larger files on a temporary basis, in order that they may print large files and send large messages through the mail system, for example.

The filesystem holding space for users are a different story altogether. In this case the quotas are likely to be different for different users, in order to accommodate the needs of each one. Let us just give here a reasonable set of defaults and comment on some of the relevant ideas involved in dimensioning these quotas. For the temporary filesystem `/temp/` we could use

```
Filesystem            blocks       soft       hard     inodes     soft     hard
/dev/sda11                 0    1200000    2400000          0        0        0
```

Note that the soft quota is rather large, at about 1.2 GB. Although this should be smaller than the size of the filesystem, in this case the user quotas may be overlapping, meaning that the quotas of all users added up can be significantly larger that the size of the filesystem. This gives all the users plenty of access to temporary disk space. If necessary, a CRON job can be set up to clean this filesystem at regular intervals, according to some given set of criteria. Note also that the hard quota is twice the size of the soft quota, although it should still be smaller than the total size of the filesystem. In the case of the `/home/` filesystem we could have

```
Filesystem           blocks       soft       hard     inodes     soft     hard
/dev/sdc1                 0     120000     132000          0        0        0
```

This soft quota of about 120 MB is sufficient for the more routine use of the system, and quotas above this should be determined by the specific needs of the projects each user is involved with. In this case the hard quota is just 10 % larger than the soft quota, just like in the `/tmp/` filesystem. These quotas are tied up with the quotas of the backup home filesystem `/bckp/home/`, which would be

```
Filesystem           blocks       soft       hard     inodes     soft     hard
/dev/sdd1                 0     240000     312000          0        0        0
```

The soft quota is exactly twice that of the `/home/` filesystem, which is the ratio of the size of the two filesystems. The hard quota is proportionally as bit larger in this case than in the case of the `/home/` filesystem, with an extra 30 % relative to the soft quota, rather than 10 %. This is to give the users a larger margin while they react to an overrun of their backup quotas, in case they fail to clean up their backup areas on a regular basis. Maintaining their backup areas is a responsibility of the users, and we usually impose some strict measures if the backup quotas go over the hard limit or the grace period expires, locking up the account until its user shows up to correct the situation. Since this is a rather harsh measure, it is important to give the users plenty on maneuvering room so they can fix the situation in time on their own. As we will see below, it is possible to automate the attribution of the quotas in the backup home filesystem, based on the quotas on the home filesystem.

The quota system described above works quite well, but it does have what can be considered a major flaw: there are no ASCII configuration files for the quotas. The goes very much against the Unix/Linux tradition, and is a serious problem. This is not just a radical position based on empty ideologism, but a rational conclusion based on painful experience. The problem is this: the information about the quotas attributed to each user in each filesystem in encoded only in the quota files located in the root of each filesystem. These are binary files that cannot be read and edited meaningfully with a normal text editor. Now, it is not intrinsically wrong that these files are in a binary format, in fact it must be so because the kernel must be able to read and write to them very often very fast, as the disk occupations are updated during the operation

of the system, and a properly formatted binary file is the right way to do that. Due to this, these files are often held open by the kernel, and this makes it much more likely that they will be corrupted in the event of a system crash. A bad enough crash may render their contents completely useless.

It is true that in the next boot the `quotacheck` tool will run and recover the disk occupations of each user, but the quotas themselves, which were attributed to each user, may be lost for good. Quotas in `/tmp/` and `/var/`, which tend to be standard values equal for all users, may be easily recovered, but if you have a system with hundreds of users with customized quotas in the `/home/` and `/temp/` filesystems, carefully attributed one by one during a long period of time, after individual requests and the ensuing negotiations, loosing all that information in one fell blow can be a *very* painful experience, take it from one that has had to go through that experience. What is important here is to add to the quota structure a set of text files which can be kept safely stowed away and which contain the valuable information which is important to you, as the manager of the system. We will describe in what follows an additional set of quota administration tools, that will be associated to a set of ASCII configuration files located in the directory `/usr/local/cetc/quotas/`, which will hold the quota information.

These extra quota tools are implemented by means of shell scripts and use the low-level tool of the package `quotatools` in order to access and update the binary quota files. The two main tools are named `edquotas` and `fixquotas`, and there is an additional one named `fixbckpquotas` to deal automatically with the quotas of the backup home filesystem. A the name implies, the `edquotas` tool is meant for editing quotas, but instead of doing that for all the quotas of a single user, it does it for all the quotas of a particular filesystem, and requires the full path to that filesystem as an argument. It accepts also the options `-u` and `-g` in order to deal with user and group quotas, just like the usual tools. It will use the text editor defined by the environment variable `EDITOR`, but unlike the case of the usual `edquota` command, in this case you can include options for the editor as part of the value of the variable. By default it will use the `emacs` editor, if the `EDITOR` variable is not set.

The script `edquotas` will change the ASCII quota file of the filesystem it works on and then will call the `fixquotas` utility in order to update the quotas in the binary quota files. It manipulates the ASCII files so that the `fixquotas` script will update only the quotas that have actually been changed. If the quotas of the `/home/` filesystem are edited it will also call the `fixbckpquotas` utility, in order to set the quotas of the `/bckp/home/` filesystem as well, fixing both the ASCII and binary files. It edits the ASCII files with the `sed` stream editor, in an non-interactive way. Both the `fixquotas` and `fixbckpquotas` utilities use the `quotatool` command to update the binary files. The `fixquotas` tool by itself can be used to re-encode into the binary quota file all the user quotas of a filesystem, using the information contained in the corresponding ASCII

quota file. It also requires the full path to that filesystem as an argument and accepts the options `-u` and `-g`. This is the tool you should use if your server suffers a bad crash for any reason and as a consequence some binary quota files become corrupted.

All these extra quota utilities can be found in the resources area, in the subdirectory `usr/local/cbin/`. You will find there also a couple of convenience utilities, `multifixquotas` and `multirepquota`. The first one is meant at fixing all at once the quotas of all filesystems mounted with quotas enabled. The second lists all the users which are overquota in any of the mounted filesystems which have quotas enabled. You should have no difficulty in reading and understanding the scripts `edquotas`, `fixquotas` and `fixbckpquotas`, which are fairly well documented with comments and use the same language of other scripts discussed in detail before, so that there is no need for a line-by-line discussion of these scripts here. They use some of the same structures we examined before. For example, they check and handle various error conditions, do file-based locking and implement a clean exit procedure. One important aspect is the standard for the names and the content format that they expect for the ASCII quota files. The files should be located in the directory `/usr/local/cetc/quotas/` and the syntax for the names is

```
<hostname>:<pathname_of_filesystem_root>.<user|group>
```

The hostname should be that of the host that the file applies to. This allow one to have the `/usr/local/cetc/quotas/` directory exported to all the servers in the cluster, containing all the quota files for all the servers, and helps to implement a centralized quota management structure. In the pathname all intermediate `/`'s should be exchanged for `_`'s, and there should be no `/`'s at the beginning or at the end of the path. The termination determines whether the file contains user of group quotas. Here are a few examples for the `fit` server:

```
fit:tmp.group
fit:var.group
fit:home.group
fit:bckp_home.group
```

The structure of the content of these files is very simple. All lines starting with a `#` are ignored as comments. The syntax for the active content is just lines with 5 columns, starting with the username in the first column, with the remaining four containing only numbers, giving the quotas. The block quotas are to be given in unit of kB. Here is an example of a quota file for the `/tmp/` filesystem:

```
# JLdL 18Nov04.
#
# Group name    Block limits                    File limits
#               soft            hard            soft            hard
```

```
stdquota        120000          132000          0               0
support         120000          132000          0               0
public          120000          132000          0               0
```

There is no internal format difference between a user quota file and a group quota file. The usernames shown are just generic examples, in each file there should be a line for each existing user. Examples of these files for the `/tmp/`, `/var/`, `/temp/`, `/home/` and `/bckp/home/` filesystems can be found in the resources area, in the subdirectory `usr/local/cetc/quotas/`, with the names `tmp.group`, `var.group`, `temp.group`, `home.group` and `bckp_home.group`. These names do not conform to the syntax given above, but one may use these simplified names to establish a cluster-wide quota policy, using soft links to them for the quota files for each host. Here is an example of some output from the `ls -l` command on the `/usr/local/cetc/quotas/` directory, illustrating this in the case of the `fit` server:

```
-rw-r--r--   1 root   root   173 Nov 18 15:41 bckp_home.group
lrwxrwxrwx   1 root   root    15 Nov 18 15:44 fit:bckp_home.group -> bckp_home.group
lrwxrwxrwx   1 root   root    10 Nov 18 15:44 fit:home.group -> home.group
lrwxrwxrwx   1 root   root     9 Nov 18 15:44 fit:pmc_tmp.group -> tmp.group
lrwxrwxrwx   1 root   root     9 Nov 18 15:44 fit:pmc_var.group -> var.group
lrwxrwxrwx   1 root   root    10 Nov 18 15:44 fit:temp.group -> temp.group
lrwxrwxrwx   1 root   root     9 Nov 18 15:44 fit:tmp.group -> tmp.group
lrwxrwxrwx   1 root   root     9 Nov 18 15:44 fit:trm_tmp.group -> tmp.group
lrwxrwxrwx   1 root   root     9 Nov 18 15:44 fit:trm_var.group -> var.group
lrwxrwxrwx   1 root   root     9 Nov 18 15:44 fit:var.group -> var.group
-rw-r--r--   1 root   root   173 Nov 18 15:41 home.group
-rw-r--r--   1 root   root   179 Nov 18 15:41 temp.group
-rw-r--r--   1 root   root   173 Nov 18 15:40 tmp.group
-rw-r--r--   1 root   root   164 Nov 18 15:42 var.group
```

Note that when new accounts are opened the `adduser` program should add lines to these short-named files, giving some default set of quotas to the new users. Later on the quotas can be edited and customized with the `edquotas` script, in order to accommodate the needs of each new user. Once you install this new set of quota tools you should never use the standard `edquota` command to change quotas, unless you want to make only temporary changes. If you use it, the ASCII quota files will not be updated and the changes you implement will be lost if you run the `fixquotas` script.

The implementation of disk quotas has some consequences in the automatic backup structure of the `/home/` filesystem. Since there are quotas on the backup home filesystem as well as on the home filesystem, there is the possibility that a user will go overquota in either one. Since the user backup was designed to not delete things in the backup area when they are deleted in the home area, so that the users can recover things that are deleted unintentionally, it must be maintained regularly by the user, which should look at it once in a while to delete things no longer wanted. If this is not done, the occupation in the backup area tends to grow until the quota is exceeded.

The first thing we must do in order to handle this new situation is to adapt the `myrror-user` script, which executes the user backups. A new version of this script can be found in the resources area, in the subdirectory `root/bin/`, with the name `myrror-user.QUOTAS`. What it does is to check the quota status of the user in both the `/home/` and `/bckp/home/` filesystems, and it only does the backup if the user is within quotas on both. Otherwise it just records the overquota events in a log file in the `/root/log/` directory. You should read this new script and try to understand how it works. Later we will be able to improve on this rather simplistic approach, but that will require that the mail system be in operation.

In order to help the users with their activities of cleaning their backup areas, an utility script is available. It is named `clean-backup` and can be found in the resources area, in the subdirectory `usr/local/cbin/`. It gives the users a few options and tries to guide them through the cleaning operation, with plenty of warnings about the dangers they face when they do this. It starts by warning the users that files deleted from their backup areas cannot be recovered at all. Then it gives the users the following alternatives:

1. Quitting the utility script and doing the cleanup by hand. This is the recommended and safest thing to do. However, users may be reluctant to do this because of the often large amount of work involved.

2. Deleting all files that have white spaces within their names, since these typically tend not to be very important files. A common case here are MP3 sound files, of which it is unlikely that a backup is needed.

3. Deleting all the files which are in the second layer of the backup. That layer contains older versions of the files in the home area, marked with the termination `.~P~`. This is probably the most commonly used alternative.

4. Deleting all files in the backup area that no longer correspond to the files currently found in the home area. This is the most radical alternative, and will prevent the user from recovering files subsequently found to have been unintentionally deleted in the home area.

Whenever it is about to delete anything, the script will give the user the alternative of confirming the deletion of each file. This, of course, involves more work and the user may be reluctant to choose to do it. In any case, it is up to the user to choose the risks which are to be taken in exchange for saving some careful selection work. Of the file deletion alternatives above the second one is just a convenience, useful because it is so common for users to have MP3 sound files in their accounts. The files downloaded from common sites on the Internet very often have blank spaces within their names. Of the two remaining alternatives, the fourth and last one is *much* more radical than the

third, and includes it. Therefore, users who do not wish to clean their backup areas by hand are strongly encouraged to try the third alternative before risking the fourth one, since that may be enough to put them within quota. Users are also emphatically warned to be sure they are not missing any files in their accounts before they use this radical alternative.

If you read the script `clean-backup` you will find that it works mostly through the use of the `find` command, which makes it quite efficient. The only alternative which is not straightforward in terms of the programming involved is the last one. In this case the `find` command is used to list all the common text files in the backup area. Then another script is used to verify whether or not each one of these files exists in the home area, and to make a list of files to be deleted. This subroutine script is called `clean-backup.sub` and is located in the directory `/usr/local/clib/scripts/`. This script uses the `bash` shell because this makes it easier to handle files with special characters within their names. Files downloaded from music and video sites often have parenthesis and other special characters within their names. Since the list of files to be deleted may be too long for a normal shell argument list, `rm` commands are then embedded into the file containing the list and it is sourced by the shell. Needless to say, since this alternative involves a lot of rather slow shell programming, it takes somewhat longer than the other ones to run.

6.4 Building a Local Mirror

Having a local mirror within your cluster may significantly improve the performance of some administrative operations. Unless you have only a very small cluster very well connected to the Internet and there already is an equally well-connected public mirror fairly close to you in network terms, it is probably advantageous to install a local mirror. In a very large cluster it may be imperative to do so. When a package installation operation is performed in such a cluster, many accesses to the mirror will be needed in quick succession if the operation is to be completed in a reasonable amount of time. Due to this it is important that there be good connectivity and bandwidth from your LAN to the mirror. A mirror is always the right way to go about making free software available in situations where a large number of machines are behind a relatively slow connection. By making available on the other side of the connection copies of the files which are going to be downloaded by many different machines, the mirror significantly lowers the traffic load on that connection. Instead of every file being transported many times, each one may be transported just once by the slow connection. The mirror functions therefore as a type of free software network caching server.

What determines whether or not a mirror is required is the relation between the bandwidth of the connection and the number of machines it serves. If the connection is feeding a whole country, a mirror may be required even if the connection is very fast. If it is feeding just your LAN it may be satisfactory even if it is much slower, so long as there is a mirror sufficiently close to you. Since usually LAN's are faster than long-distance connections and large organizations such as universities and corporations tend to have solid LAN infrastructures, in such cases the whole thing may boil down to whether or not there already is a free software mirror within the LAN of that organization. It there isn't and you do install one, you should fully expect to become *it*. In order to decide whether or not you need a local mirror, you may start by using the public mirrors which are closest to you and checking out the speed with which the package installation and upgrade operations proceed. If you find that you keep waiting a lot for the network to transport the control files and packages, then the installation of a local mirror may be a good idea.

6.4.1 Putting the Mirror Together

Installing a local mirror is not difficult, but will of course require some investment in hardware, for a significant amount of disk space will be needed. We will consider here only mirrors of the various parts of the Debian distribution and of the Linux kernel sources. In Table 6.3 you will find the current sizes of the mirrors you might consider installing locally in your cluster. The Debian mirror listed there are not complete mirrors, they are just for the i386 architecture, without the sources. The Linux mirror has only the `bzip2`-compressed kernel files, for versions 2.2, 2.4 and 2.6, without any patch files. The whole thing adds up to approximately 21 GB. In order not to exceed a

2/3 occupation rate of the disk, you see that about 30 GB of disk space will be needed. A standard 36 GB SCSI disk will do just fine, but a 18 GB disk is already too small.

Mirror	Exact size in MB	Rough size in GB
debian	16851	16.9
debian-non-US	107	0.1
debian-security	1069	1.1
debian-backport	1433	1.4
pub/linux/kernel	1959	2.0
Totals:	21419	21.5

Table 6.3: Current sizes of the mirrors one might consider installing locally in a cluster. In this case the Debian mirrors include only the i386 architecture, without sources. The sizes are correct as of November, 2004.

If you want to include the sources of the Debian packages, the disk space requirements increase very significantly, as can be seen in Table 6.4. Now you need 41 GB of disk to have the whole thing fit in, and 60 GB for a 2/3 occupation rate. We are looking here at a standard 72 GB SCSI disk. It is important to have extra room on the disk, because mirrors tend to present a vegetative growth. They may grow slowly, but they will surely grow in time. There may be a great temptation to use IDE disks for the mirror, since so much space is needed and it is not used very often. However, unless you can install these disks in a machine that will be used for no other purpose, you should refrain from doing this since it may degrade the performance of your server, specially if many machines will be using the mirrors, such as in a public mirror server.

Mirror	Exact size in MB	Rough size in GB
debian	34295	34.3
debian-non-US	302	0.3
debian-security	2169	2.2
debian-backport	2485	2.5
pub/linux/kernel	1959	2.0
Totals:	41210	41.3

Table 6.4: Current sizes of the mirrors one might consider installing locally in a cluster. In this case the Debian mirrors include only the i386 architecture plus the sources. The sizes are correct as of November, 2004.

We will assume that once you obtain your disk you will assemble it in your server, format it and mount it on a mount point `/sft/` on the server. Formatting the disk with the second extended filesystem should go exactly as it did before when you formatted other disks. Do not forget to use the option `-j`, in order to make a journaling `ext3`

filesystem. Remember also to label the filesystem, then create the mount point, make the appropriate changes in the `fstab` file and mount the disk. Although it is possible to update mirrors by means of tools using the FTP or HTTP protocols, such as the `mirror` and `wget` programs, by far the best tool for this purpose is the `rsync` command. Like all the other tools, it requires that an appropriate server be running on the machines you are going to mirror from, but this is not at all a problem because `rsync` is becoming the standard mirroring tool and is supported by the most sites containing the mirrors listed in the tables.

Setting up and maintaining the mirrors is just a question of running a few appropriate scripts regularly by means of a CRON job. Once you have your mirror filesystem in place in the `/sft/` directory, you should create within it a subdirectory `sbin/` to hold the mirror scripts. You should also create a subdirectory `log/` for the log files produced by these scripts and a subdirectory `lock/` to be use for locking purposes. Finally, you should create the directories to hold each one of the mirrors. For example, for the Linux kernel mirror you should create the subdirectory `pub/linux/kernel/`. In this case it may be convenient to also make in the `/sft/` directory a symbolic link called `kernel` pointing to this subdirectory, in order to document what the `pub/` subdirectory is being used for, as well as to make it easier to locate this mirror. Here is some output of the `ls -l` command within the `/sft/` directory:

```
drwxrwxr-x    8 root     root         4096 Oct 27 18:26 debian
drwxrwxr-x    4 root     root         4096 Oct 12 16:11 debian-backport
drwxrwxr-x    6 root     root         4096 Aug  2 09:24 debian-non-US
drwxrwxr-x    5 root     root         4096 Nov 21  2002 debian-security
lrwxrwxrwx    1 root     root           16 Feb  8  2004 kernel -> pub/linux/kernel
drwxr-xr-x    2 root     root         4096 Nov 27 14:46 lock
drwxrwsr-x    2 root     root         4096 Oct 28 23:41 log
drwxr-xr-x    2 root     root        49152 Feb  8  2004 lost+found
dr-xr-xr-x    3 root     root         4096 Jul 29  2000 pub
drwxrwxr-x    3 root     root         4096 Oct 28 21:09 sbin
```

In the other cases the subdirectories simply have the name of the corresponding mirrors, such as `debian/` for the main part of the Debian distribution, `debian-security/` for the site of the Debian security team, `debian-non-US/` for the part of the Debian distribution which cannot be legally exported from the US, and `debian-backport/` for an interesting project associated to the Debian distribution, which is dedicated to back-porting packages from the testing distribution to the stable distribution.

Mirroring scripts appropriate for each one of these mirrors can be found in the resources area, in the subdirectory `sft/sbin/`. You should copy the scripts for the mirrors you want to have into the `/sft/sbin/` directory of the server. You may use them as they are, since they are configured to mirror from well-known public servers, but it is a good idea to edit them and exchange these generic servers for those which are closer to you in networking terms. The scripts are named after the mirrors they

implement, and those for the Debian mirrors include only the i386 architecture, without the sources. You will also find in the resources area versions of these scripts marked with the termination `.SOURCES`, which include both the i386 architecture and the sources. The scripts for each Debian mirror are separated in two parts, one for the pool directory and another one for the rest of the distribution. For each script that you put in the `/sft/sbin/` directory you should also make within that same directory a symbolic link with the same name and a termination `.batch`, with commands such as

```
ln -s mirror-linux-kernel mirror-linux-kernel.batch
```

There are examples of these links in the resources area too. The reason for this is that, depending on the name with which these scripts are called, they will run in interactive mode, writing a progress report to the screen, or in batch mode, without the progress report. Here is some output of the `ls -l` command within the `/sft/sbin/` directory, edited to fit within the margins, showing only the name, type, ownership and access mode of each file:

```
-rwxr-xr--  root  mirror-debian-backports_dist
lrwxrwxrwx  root  mirror-debian-backports_dist.batch -> mirror-debian-backports_dist
-rwxr-xr--  root  mirror-debian-backports_pool
lrwxrwxrwx  root  mirror-debian-backports_pool.batch -> mirror-debian-backports_pool
-rwxr-xr--  root  mirror-debian-non-us_dist
lrwxrwxrwx  root  mirror-debian-non-us_dist.batch -> mirror-debian-non-us_dist
-rwxr-xr--  root  mirror-debian-non-us_pool
lrwxrwxrwx  root  mirror-debian-non-us_pool.batch -> mirror-debian-non-us_pool
-rwxr-xr--  root  mirror-debian-security_dist
lrwxrwxrwx  root  mirror-debian-security_dist.batch -> mirror-debian-security_dist
-rwxr-xr--  root  mirror-debian-security_pool
lrwxrwxrwx  root  mirror-debian-security_pool.batch -> mirror-debian-security_pool
-rwxr-xr--  root  mirror-debian-us_dist
lrwxrwxrwx  root  mirror-debian-us_dist.batch -> mirror-debian-us_dist
-rwxr-xr--  root  mirror-debian-us_pool
lrwxrwxrwx  root  mirror-debian-us_pool.batch -> mirror-debian-us_pool
-rwxr-xr--  root  mirror-linux-kernel
lrwxrwxrwx  root  mirror-linux-kernel.batch -> mirror-linux-kernel
-rwxr-xr-x  root  run-the-mirrors
```

All these scripts work in essentially the same way. Let us examine one of them as an example. In the beginning of the script `mirror-linux-kernel` you find some preparatory code, in which the script changes to the directory containing the mirror, log, error and lock files are defined and a file lock is checked and executed, to make sure that two instances of the same script never run at the same time. A clean-exit procedure is also defined in this part. Following that the variable `rurl` is defined, containing the rsync-URL with the address of the server to be used, as well as the directory to be mirrored,

```
set rurl = 'rsync://rsync.kernel.org/pub/linux/kernel/'
```

The block of code that follows defines the set of subdirectories and files to be mirrored. This is done with the options `--include` and `--exclude` of the `rsync` command. In the case of the mirror of the Linux kernel we choose to mirror only complete kernel files in the `*.bz2` compressed format, and the corresponding signature files. This list of options is stored in the variable `excl`,

```
set excl = ""
set excl = "$excl --include v2.2/"
set excl = "$excl --include v2.4/"
set excl = "$excl --include v2.6/"
set excl = "$excl --include v2.2/*.tar.bz2"
set excl = "$excl --include v2.2/*.tar.bz2.sign"
set excl = "$excl --include v2.4/*.tar.bz2"
set excl = "$excl --include v2.4/*.tar.bz2.sign"
set excl = "$excl --include v2.6/*.tar.bz2"
set excl = "$excl --include v2.6/*.tar.bz2.sign"
set excl = "$excl --exclude *"
```

The versions 2.2, 2.4 and 2.6 of the kernel are included here. If you lack enough disk space, you can eliminate the older version. The next block of code defines a list of options for the `rsync` command, in order to make it operate in mirroring mode, storing the list in the variable `opts`,

```
set opts = ""
set opts = "$opts --recursive"
set opts = "$opts --times"
set opts = "$opts --links"
set opts = "$opts --hard-links"
set opts = "$opts --delete"
set opts = "$opts --delete-after"
set opts = "$opts --compress"
set opts = "$opts --verbose"
```

Each one of these options deserves a few words of comment. The option `--recursive` makes `rsync` descend into subdirectories, copying all files and subdirectories recursively. The option `--times` causes it to preserve the time stamps of files. The option `--links` make it reproduce the symbolic links it encounters. The option `--hard-links` makes it preserve the hard links of files; this is a rather expensive option in terms of running time and, since it may not be necessary, you may want to eliminate it. The option `--delete` tells the program to delete files in the local mirror which are no longer present in the original mirror. The option `--delete-after` causes it to delete the files only after all the new files have been downloaded; there may not be enough extra space for this, in which case you should eliminate this option. The option `--compress` causes the program to compress the data during the traffic across the network; since most of the

content of the mirrors is already compressed this option may not be too helpful, but it does no harm. The option `--verbose` used only once will cause the program to write a reasonable amount of output for logging purposes.

One option that you should *not* use in these scripts is the one named `--delete-excluded`. This option is *very dangerous* and should *never* be used in the Debian scripts with names terminated with `_dist`, because these scripts run above the pool directory and exclude it. Therefore, the use of this option in this case would cause the irretrievable deletion of the *whole pool directory*, which is where the bulk of the disk content of the distribution is located. Next in the code of the mirroring script you see a command to disable the shell expansion of the `*` character which is contained in the `excl` variable. The last block of code actually runs the `rsync` command in mirroring mode, using all the options and exclusions defined before,

```
if ( ‘basename $0 | cut -d. -f2‘ == batch ) then
    ( echo -n "Starting on " ; date ) >&! $log
    rsync $opts $excl $rurl . >>& $log
else
    rsync $opts --progress $excl $rurl .
endif
```

In the last part of the file you can see the shell expansion of the `*` wildcard being restored and the clean-exit target label and command, removing the lock file. The Debian scripts work much in the same way, the only relevant differences being the construction of the exclusion lists for the selection of a single architecture. For example, in the script `mirror-debian-us_pool` one finds a loop over the existing architectures in order to build an exclusion list for the packages of all unwanted architectures,

```
set excl = ""
foreach arch ( alpha arm hppa hurd-i386 ia64 m68k mips mipsel powerpc s390 sh sparc )
    set excl = "$excl --exclude *_$arch.deb"
    set excl = "$excl --exclude *_$arch.udeb"
end
```

This piece of code will need maintenance when new architectures are included in the distribution, of course. One also finds in the same script a block of code to exclude the source directories and files,

```
set excl = "$excl --exclude source/"
set excl = "$excl --exclude *.dsc"
set excl = "$excl --exclude *.tar.gz"
set excl = "$excl --exclude *.diff.gz"
```

In the script `mirror-debian-us_dist` the loop over the existing architectures has a larger content in terms of lines of code, due to the fact that in this case it is necessary to exclude some more architecture-specific files and directories,

```
set excl = ""
foreach arch ( alpha arm hppa hurd-i386 ia64 m68k mips mipsel powerpc s390 sh sparc )
    set excl = "$excl --exclude disks-$arch/"
    set excl = "$excl --exclude binary-$arch/"
    set excl = "$excl --exclude installer-$arch/"
    set excl = "$excl --exclude Contents-$arch.gz"
    set excl = "$excl --exclude *_$arch.deb"
    set excl = "$excl --exclude *_$arch.udeb"
    set excl = "$excl --exclude *_$arch.changes"
    set excl = "$excl --exclude *_$arch.nondebbin.tar.gz"
    set excl = "$excl --exclude *_$arch-static.nondebbin.tar.gz"
end
```

One also finds in this script the code to exclude the pool directory and a few more unwanted branches of the tree of the distribution,

```
set excl = "$excl --exclude pool/"
set excl = "$excl --exclude project/trace/*"
```

In the resources area you will also find a convenience script called `run-the-mirrors` which just runs all the mirrors at once. They are all run simultaneously, except for the fact that for each Debian mirror the `_dist` and `_pool` scripts are run sequentially in this order. This script is useful for setting up a root crontab entry to run the mirrors regularly. It is recommended that you update the mirrors daily. In order to do this at zero hours and 30 minutes the crontab entry should be something like this:

```
30 0 * * * /bin/tcsh /sft/sbin/run-the-mirrors
```

The first time you run the mirrors the scripts will take considerable time to finish, since they will be downloading the full content of the mirrors you chose. Subsequent runs should be much shorter, since only new or changed files will be downloaded then. The larger and most active mirror is the main Debian distribution, the package pool in particular. You may expect to have several hundred files downloaded every day.

6.4.2 Making the Mirror Available

Having installed the mirrors, there is now the issue of making its contents available to other machines. The mirror server may be used exclusively for your cluster, it may be set up to serve your department or the whole institution you are in, or it may even be open for general public use. What you will do about this depends on several things, including how well connected to the network the machine is, what level of load it can handle, how often you choose to update it, and the Internet policies adopted by the department or institution. Once it is installed a mirror requires very little maintenance, but there may be need of some once in a while. There is no sense in having a mirror available for general use if it is not online 24 hours a day, 7 days a week. If the local

network the machine is connected to is not well connected to the Internet, the mirror may not be of much use to people outside that LAN. After all these things are dully considered, and in the interest of the general cooperation within the Internet community, please do consider opening up your mirror as much as the available infrastructure will allow without disturbing the operation of your server and LAN.

A mirror can be made available through the network in several different ways, using several different protocols. Basically there are three ways in which it can be used by individual machines for package updating and installation: by means of anonymous FTP, by means of the HTTP protocol the web is based on, and by means of read-only NFS exportation. The use of the mirrors by means of the NFS protocol is the more convenient, but it requires a fast and solid network and does not include particularly strong security mechanisms, so it should only be used within your cluster or at most within your LAN. Anonymous FTP is the most traditional method for the distribution of free software, it has very good security and it is easy to control the access to it if that is the case. Its installation must be done carefully in order to avoid security holes, but when you install an FTP server package the installation tools will probably give you the choice of having an anonymous FTP structure set up automatically in the right way. The use of a web server and the HTTP protocol is fast becoming the standard way to use a Debian mirror. It is efficient, quite easy to implement and very secure. It also has the advantage that people on the web will be able to browse the contents of the mirror very easily and comfortably, using any standard web browser.

The simplest way to make your mirrors universally available is by the installation on your cluster server, or whatever machine holds the mirrors, of a web server such as the Apache server. Just install the `apache` package and configure it to make the directory `/sft/` the document-root of the server. The configuration file for the server is `/etc/apache/httpd.conf`. If you already have a server on the machine with its document root elsewhere, or if you want to have the root elsewhere in order to house a set of web pages at the machine, you can use a simple HTTP alias for the directory containing the mirrors. A simple line such as

```
Alias /sft /sft
```

in the `httpd.conf` file will cause the `/sft/` directory to be visible on the web at the URL

```
http://fit.free.univ.com/sft
```

Read the `httpd.conf` file and look for the examples of the use of the `Alias` configuration keyword that are contained within it. Using a Debian mirror by means of the HTTP protocol is what we have been doing when we installed the cluster server from a public mirror.

You may also use a "virtual host", which is sometimes referred to as a "virtual domain", and which will make it look like there is an independent machine on the LAN which is dedicated to the mirror. In this case the mirror will be available at a different address, associated to a different hostname in your LAN, which for the sake of example we will assume to be `sft`. This would make the mirror accessible at the URL

```
http://sft.free.univ.com
```

The use of a virtual domain has the advantage that, if later you decide to change the mirror from the original machine to another, the virtual domain can be moved from one machine to the other together with the mirror, making the change transparent to the users of the mirror and avoiding any interferences with the other functions of either machine. This is specially useful in a public mirror. The way to do this is to ask the LAN administration for another IP address, to be associated to the hostname `sft`. If we assume, just as an example, that this address is `172.16.129.31`, then you should include in the file `hosts` of the server the line

```
172.16.129.31   sft.free.univ.com       sft
```

This example is contained in the files `hosts.4.PMC` and `hosts.4.TRM` within the subdirectory `etc/` of the resources area. It is also necessary that the LAN administration provide an appropriate set of DNS records for this hostname and this address in the primary DNS server of the LAN, of course. Having the address, you will now use it on your server, registering within the system a second address associated to its network card. This is done using the "IP-alias" structure of the kernel, which is available in any kernel that has networking turned on. If the network interface of the server to the LAN is `eth0`, then the second address will be associated to a virtual interface `eth0:0` on the same network card. This will allow the kernel to receive, interpret and manipulate the packages sent to this second address. The way to set up this virtual interface is to include in the file `interfaces` within the `/etc/network/` directory a configuration block such as

```
#
# A second address on the first network card.
auto eth0:0
iface eth0:0 inet static
        address 172.16.129.31
        netmask 255.255.255.0
        network 172.16.129.0
        broadcast 172.16.129.255
```

and then to raise the interface, issuing the command

```
ifup eth0:0
```

The interface will then be active, as the `ifconfig` command will show. This configuration block can be found in the resources area, in the files `interfaces.2.PMC` and `interfaces.2.TRM`, within the directory `etc/network/`. Note that this virtual interface configuration has no "gateway" entry. With this entry included in the `interfaces` file the virtual interface will be automatically raised at every boot. The last step needed in order to set up the virtual domain is the configuration of the Apache web server. You should include in the `httpd.conf` configuration file a configuration block such as

```
#
# A virtual domain for the mirror.
#
<VirtualHost sft.free.univ.com>
ServerAdmin root@fit.free.univ.com
DocumentRoot /sft
ServerName sft.free.univ.com
ErrorLog /var/log/apache/sft.free.univ.com-error.log
TransferLog /var/log/apache/sft.free.univ.com-access.log
</VirtualHost>
```

Note that you should fix the administrative email address that appears in this configuration block. Use your own address, or the address of whoever will be in charge of the mirror, rather than the address of the root account of the server. You must also create the two log files that appear within this block. There is an example of virtual host block in the standard `http.conf` file distributed by Debian for the Apache web server. Once you have done all this the command

```
/etc/init.d/apache reload
```

will activate the virtual domain, which you can easily test by using a browser to access the address `http://sft.free.univ.com`.

The best way to make the mirrors available to the local machines in the LAN is by the NFS protocol. In order to do this you should export the `/sft/` filesystem to all systems in the cluster. You can easily do this using netgroups. Entries for the `/etc/exports` file of the server should be

```
#
# The mirror to the complete group of compute nodes.
/sft     n0000(ro,async) @pmcnodes(ro,async)
```

for the case of compute nodes, and

```
#
# The mirror to the complete group of remote-boot terminals.
/sft     fit00(ro,async) @fitterms(ro,async)
```

for the case of remote-boot terminals. These entries can be found in the files `exports.4.PMC` and `exports.4.TRM` in the subdirectory `etc/` of the resources area. Having included them in the `exports` file of the server, remember to make the NFS server reload its configurations, with the command

```
/etc/init.d/nfs-kernel-server reload
```

This will make it possible to mount on the clients the filesystem containing the mirrors, by means of commands such as

```
mount fit.free.univ.com:/sft /mnt
```

Since this will make the `/sft/` filesystem available on the client as if it were a local filesystem, the Debian package-manipulation tools may then be reconfigured to use this local filesystem rather than a remote server by means of HTTP or FTP. This is the first example of a *service* filesystem export, as opposed to the system exports we have done so far. In the normal operating environment of the cluster the corresponding imports on the clients will be done by means of automatic on-demand mounting using the *autofs* system and the `automount` daemon, rather than by manual mounting as described above. The autofs structure, as well as the corresponding adjustments that must be done on the configurations of the clients, will be examined in Chapter 7.

The traditional way to make a mirror generally available is through the FTP protocol, by means of what is called "anonymous" FTP. In order to do this you must install an FTP server and set up an anonymous FTP account with the `/sft/` directory as home, that is, as the root of the anonymous FTP service. If you have not done it before, you may begin by installing the `wu-ftpd` package, which contains a well-known FTP server. The package will ask you a few questions. Answer with "yes" to the question about creating an anonymous FTP account. When prompted for the home of the account use `/sft` instead of the default presented to you. Answer with "no" to the question about creating a directory for user uploads. This will complete the job. If you already had the server installed but did not create the anonymous account during the installation, you can do it now. First change the entry of the `ftp` user in the `passwd` file, it should look like

```
ftp:x:110:110::/home/ftp:/bin/false
```

and you should change it to

```
ftp:x:110:110::/sft:/bin/false
```

With this done the anonymous FTP structure can be created by simply running the command

```
addftpuser
```

It is important to do it this way because, for security reasons, the ownerships and access modes of all the files and directories involved must be set carefully in the appropriate way. Have a look at the new contents of the `/sft/` directory in order to see this. You may now remove from the `/home/` filesystem the `ftp/` subdirectory and its contents, if it exists at all. Finally, you may want to customize the welcome message contained in the file `welcome.msg` that you will find within the `/sft/` directory. Once the anonymous FTP account is set up, all the contents of the `/sft/` directory will be available at the URL

```
ftp://fit.free.univ.com
```

and will be accessible by a browser, by the `wget` command we used before, or by any standard FTP client. The access to the FTP server can be controlled on a host-by-host or network-by-network basis by means of *TCP wrapper* access control structure to be discussed in Chapter 7. This will allow you to limit the access to the FTP server to you LAN or to any subset of the Internet, in case you need to do that for some reason.

You may also consider installing on your mirror server an `rsyncd` server, so other machines can mirror from yours. However, if you want to do this, it is likely you are already a more experienced user and do not need too much detailed help. In any case, here are the main pointers: the only package needed is the `rsync` package, which is already included in our standard set of packages for servers; you should read the man page `rsyncd.conf`, which contains all the explanations; you will need to add an appropriate line to your `inetd.conf` file; finally, a configuration file `/etc/rsyncd.conf` will have to be created. You may look at the sample configuration file `sample-rsyncd.conf.gz` you will find in the directory `/usr/share/doc/rsync/`. Also, a configuration file for the mirrors suggested here can be found in the resources area, in the subdirectory `etc/`.

Chapter 7

Further Cluster Structures

In this chapter we will describe a series of structures that are involved in the integration of a whole set of machines into a cluster. The idea is that, from the point of view of the user, the whole set of machines works in such a way that if effectively forms a single system, presenting to the user an homogeneous operating environment. Form the point of view of the manager, what counts is that the administration of the whole set of machines can be done centrally, also in a very homogeneous fashion.