Investigation of load-balancing for a network of Suns

Peter Smith

Honours project, 1992

Contents

Abstract

With the advent of networked and distributed operating systems we no longer depend on one centralised computer for our processing power. One recent issue that has not been addressed within our network of Sun workstations is known as *load balancing*. This feature gives the ability to distribute the offered workload among the available computers, to improve system performance and to ensure that machines do not remain idle while others are overloaded.

This report studies the `clb` load balancing system that has been developed for use within our environment. `clb` uses an apparently new method, the *initial placement of users*, rather than the more familiar, but complex methods of initial placement of processes and process migration.

`clb` has proven to be useful in distributing the system workload among the available hosts, and will help to smooth out the peak periods of workload. A novel feature of `clb` is that it can make suggestions about the configuration of the system's fixed resources.

Chapter 1

Introduction

1.1 The history of operating systems

Over the last decade, there has been a trend away from computer systems based on a single, large, expensive machine towards systems based around computer networks. Whereas in the past it was more economical to have all processing power provided by a centralised computer, it has become more cost effective to spread the workload around a number of smaller, cheaper machines. This change in hardware configuration has lead to a corresponding need for software that can control a whole network of computers.

Although many older centralised computer systems had network connections, their use was fairly limited. If a user wished to access data that was stored on another computer's disk, they would explicitly request the transfer of information from the remote machine to their own. For example, the `ftp` and `rcp` protocols were commonly used. If a user wished to run a program on a remote machine, they would use a remote login program such as `rlogin` or `telnet`, or a remote execution program such as `rsh`. This lack of transparency was quite acceptable since it was not common to require services from a remote site.

As the number of machines increased, it became important to share resources around the network. This requirement lead to the development of the *Networked Operating System* in which the existing centralised operating system is augmented with layers of software that handle the remote use of resources. For example, in our network of Sun Workstations the operating system has an extra layer (called the *Network File System*) that controls the sharing of files among the workstations. This allows users to access remote files transparently, that is, as if they were stored locally. Although similar software packages exist for the sharing of terminals, printers and tape drives, transparent processor sharing is not provided.

The most recent advance in operating system design has been the concept of *Distributed Operating Systems*. In this architecture, the ability to communicate with remote machines is a fundamental requirement, rather than an extension. Often each machine will have a small local operating system (known as a Microkernel) that will supply globally transparent communication links between processes. This effectively removes the distinction between local and remote communication, therefore removing the need for the extra layers of software to handle remote services.

1.2 Load balancing

The use of a transparent communication system for sharing files and devices leads us to question whether the same principles can be applied to the sharing of processing power. In a centralised or networked operating system, almost all new processes are started on the machine that manages the creating (or parent) processes. Transparent remote execution of a process is difficult since the process may use centralised features, such as shared memory or disk files, to communicate with its peers. It may be possible to execute processes remotely, although the communication is limited and has high overhead.

In a distributed operating system, all interprocess communication is based on message passing (or a similar method). When a new process is created, it can be initiated on any machine while still maintaining communication with all other related processes. The overhead of remote message passing is greater than that of local communication, however the flexibility of such a system outweighs the costs.

The ability to share processors in this manner has been identified and the algorithm for spreading workload around the system to improve performance is known as *load balancing*. Load balancing has become an important issue in operating system research for several reasons. Firstly, it is now possible to use more processing power than is available locally. Secondly, the utilisation of the processors in a network can be balanced to ensure that users will receive the best possible response for their jobs. Finally, specific features that are only available on remote machines (for example, array processors) are easier to use.

1.3 Description of this report

The remainder of this report will look at a load balancing system that has been designed, implemented and installed at the University of Canterbury. Chapter 2 examines the various components of a load balancing system and will survey a few of the existing implementations. Chapter 3 describes the basic model of `clb` (the Canterbury Load Balancer) and introduces the idea of initial placement of users. Chapter 4 describes the derivation of `clb`'s load balancing algorithm and Chapter 5 examines the software components of `clb`, along with the extra support programs that have been implemented.

The methods used to evaluate the effectiveness of `clb` are discussed in Chapter 6 and the experimental results are given in Chapter 7. Chapter 8 looks at the limitations of `clb` and some possible future improvements. Finally, Chapter 9 presents some conclusions about the effectiveness of this load balancing system.

Chapter 2

A survey of load balancing methods

2.1 System models

The design of a load balancing system must take into account the structure of the system and the type of workload presented by its users. This section will look at the two common models of processor organisation, the workstation and processor pool models, along with some existing load balancing implementations. Some systems follow one or other of the models very closely, while others follow a hybrid model in which both workstations and processor pools are used.

2.1.1 The workstation model

In the workstation model, each user sits at a single workstation that is used for most of their processing needs. The user accessing the workstation via its console is considered to be the owner of the machine and is given priority over its use. An advantage of this is that users have direct access to their machine (rather than using a network) and when they are the only user, the response times are predictable.

The main limitation of this model is that workstations tend to be small and often don't have local disk. If a user wishes to execute a large or disk intensive job, their local processing power may not be sufficient. In this situation the user must explicitly use a larger machine. On the other hand, if the owner is not using the machine, or is only generating a small amount of workload, much of the workstation's power is being wasted.

Load balancing systems that have be designed for the workstation model consider each machine to be either idle or busy. Generally, a machine is considered idle if the owner has been absent for a reasonable length of time. Alternatively, a machine may be classed as idle if its loading is sufficiently small. A workstation is considered busy if it is not idle.

In the Condor [2] load balancing system, users must explicitly submit their large jobs to a queue for execution on a more suitable machine. When an eligible workstation becomes idle, the job will be started. If the owner of the remote workstation returns, execution of the job ceases and it is put back in a queue for continued execution at a later date (possibly on a different workstation).

In the Butler system [9], a remote job is started by the use of the 'rem' command. As with Condor, an idle workstation is located and the process starts executing on it. If the owner of the remote workstation returns, the job will not be saved, but is warned and then terminated.

The Process Server [8] uses a slightly different approach to that of Condor and Butler. Instead of starting large jobs on idle workstations, a predefined set of programs (for example, compilers and translators) remain resident on the network's compute servers. If a user wishes to use one of these programs, it will be executed on its special machines. This has the effect of speeding up the execution of the job by eliminating the start up costs.

Other load balancing systems based on the workstation model are used in the V-System [13] and Sprite [3] distributed operating systems.

2.1.2 Processor pools

In the processor pool model, the processing power is concentrated in a few *compute servers*, each of which may be a single, large machine or a collection of many smaller processors. Users will typically log in through X-terminals rather than workstation consoles, and their processes will be executed on machines within the processor pool.

When a process is to be created, the pool of free processors is searched in order to locate a free machine. If all processors are busy, they must share their time among the active processes. With this style of allocation, the concept of the *home* machine no longer exists and all processors are shared equally among the different users. Unlike the workstation model, remote execution tends to be transparent.

In load balancing systems that follow the processor pool model, it is necessary to estimate the amount of work that each processor is capable of doing, rather than simply saying that it is idle or busy. This would normally involve knowing the speed of each processor and the number of jobs it is currently executing. The Amoeba [11] distributed operating system uses this knowledge to estimate which processor can devote the greatest amount of processing power to a job.

The Utopia [15] load balancing system is a slight variation on this basic model, in which processes are only considered for load balancing if they are classified as being computationally intensive. Normally, small processes are started on the same processor as their creator.

The MOS [1] system will start each new process locally, but may decide to transfer the job to a more suitable processor if it demands too much CPU power. This system will avoid the overhead of remote execution if the process is short-lived.

2.1.3 Our network of Sun workstations

The network of Sun workstations at the Computer Science Department of the University of Canterbury follows a hybrid model of machine configuration. There are eleven Sun-4 computers, six of which are diskless workstations, while the remaining five have local disk and are classified as compute servers. The diskless machines follow the workstation model and are allocated to staff members, students are permitted, but are advised not to use them. The compute servers do not have owners. Instead, they are configured to serve a large number of users (similar to a processor pool).

Four of the compute servers belong to the Computer Science department, although only three of them, *kahu*, *ruru* and *huia* are for general use. The fourth, *whio* is reserved for research purposes, while the fifth server, *cantua*, belongs to the Computer Services Centre and is shared with all other departments. All of these machines have the same type of processor and are controlled by the SunOS Network Operating System (a version of UNIX).

The majority of users are undergraduate students who normally connect to the compute servers via X-terminals. There are also a small number of postgraduate students and staff who use these machines via X-terminals, Sun consoles and Apple Macintoshes (via NCSA Telnet). During the busy periods of the year, up to 80 people may be using the four servers at any one time.

Each of the compute servers is of a different model, that is, a SparcStation 1, a SparcStation 1+, a SparcStation 2 and a SparcServer 690MP. These machines each have their own processing speeds, memory sizes and amounts of local disk. It is believed that the major bottleneck of this network is the speed and availability of disks and memory, rather than CPU power.

Two load balancing systems for use in a network of UNIX machines have been located, although neither of them were installed in our system as part of this project. As previously discussed, the Condor [2] system is designed for a pure workstation model and is useful for executing large batch jobs. This was not desirable in our system where a large number of users perform relatively small jobs. The second system, Utopia [15], allocates moderately sized jobs among the available processes in order to minimise response times. Although this method would have been worth experimenting with, the designers of Utopia have not yet made it publically available.

2.2 The elements of a load balancing system

So far the reasons for using load balancing and the system models in which they must operate have been discussed. This section will take a closer look at the design decisions and goals associated with load balancing systems. Further information is given by Goscinski [6], Tanenbaum [11] and Hac [7].

As with most operating system elements, a load balancing system can be divided into two main components, the policy component and the mechanism component. The policy is responsible for making decisions about where a job should execute. In the processor pool model, this would require collecting load information about the available processors and numerically ranking them to determine which would be best to use. In the workstation model, the policy must recognise which of the idle machines have sufficient resources to run a job.

Once a decision has been made, the load balancing mechanism arranges for the job to be executed remotely. This may be as simple as sending a request message to the remote processor, or may involve a large amount of reconfiguration and monitoring of the job or the remote workstation. The operating system structure will have a major affect on the mechanism, with the existing UNIX load balancing systems being rather complex.

The following design decisions affect both the policy and mechanisms of a load balancing system.

- Load sharing or load balancing

  So far the term *load balancing* has been used as a general means of describing two different methods. The first method, primarily used in the workstation model (systems like Condor and Butler), is more accurately known as *load sharing*. In this situation, idle workstations are allocated jobs that are too large to currently be executed on their owner's workstation. The second method, correctly known as *load balancing*, is used to allocate jobs among the available processes with the intent of balancing the workload (eg. Amoeba and Utopia). From this point on, the correct definitions of load balancing and load sharing will be used.

- Initial placement or migration

  With the initial placement (or static) method, a newly created job is allocated to a particular machine (based on the policy decision). This job will remain on that machine and continue executing until its completion. In a migratory (or dynamic) system, the possibility exists for moving the job during its execution. Such systems can adapt more quickly to changing workload conditions, although in practice the advantages may be limited [4]. Systems that use migration, such as MOS [1], Condor [2], V [13] and Sprite [3], generally require more complex mechanisms (and associated costs) than systems that use initial placement, such as Amoeba [11] and Utopia [15].

- Optimal or suboptimal

  If the workload that will be presented to the computer system is known in advance, then it is mathematically possible to calculate the optimal allocation of jobs to machines. However, since prior information is not commonly available, suboptimal policies are in frequent use. Even though suboptimal methods do not always give the best results, they are much simpler to implement and are less computationally intensive than optimal policies.

- Distributed or centralised information

  In order for a load balancing decision to be made, a summary of the load on each available processor must be known. This *load exchange* mechanism can be done in either a centralised or distributed way. With a centralised method, each processor will periodically transmit its statistics to a central machine, so that all the necessary data can be found in one place. Although this method is easy to implement and guarantees that the most up to date information is available, the central machine may become a bottleneck. With distributed load exchange, each machine transmits information to all (or a large subset) of the other machines. This method improves the reliability and availability of the data, but can lead to a higher overhead in transmitting and searching for the most up to date information.

- Adaptive or non-adaptive

  The amount of load information used by the balancing algorithm can affect the accuracy of its predictions. With non-adaptive algorithms only the most recent information about each processor is used. With adaptive methods, the past performance of the machines is also taken into account. Even though adaptive systems would normally give better performance than non-adaptive methods, the amount of information retained and the complexity of the algorithms may make them infeasible.

- Sender or receiver initiated bidding

  A design decision that is relevant only to load sharing methods is that of bidding. When sender initiated bidding is used, an overloaded machine will send out messages to search for idle machines. If a suitable machine responds, some of the workload can be migrated between them. With receiver initiated bidding, it is the idle machines that send requests for extra work, while the overloaded machines respond. The advantages and disadvantages of each method are given in [12]

One final design decision involves determining what the load balancing system should try to improve. Although there are a large number of factors that could be optimised, most systems try to balance either the utilisation of the machines or the response time of the jobs. Balancing the utilisation will ensure that each machine will be performing its fair share of work and that machines do not remain idle while others are overloaded. By balancing response times, users would expect jobs to complete within a minimum, predictable time, with respect to the current workload in the system.

Chapter 3

The clb model

As discussed in the previous sections, our network of Sun workstations is a hybrid of the workstation and processor pool models. Existing UNIX load balancing systems were either not suitable for our environment or were not publically available. The `clb` (Canterbury Load Balancer) [10] system was designed and implemented specifically for use in our network.

3.1 Initial placement of users

`clb` is based on the initial placement of user login sessions, whereas all other load balancing systems we are aware of are based on initial process placement or process migration. In our network of Suns, users log into the system by selecting the compute server they wish to use. They remain logged in to this server for the duration of their session (up to several hours in length). During this time they are permitted to create extra *shells* on any of the workstations.

When `clb` is installed, the same login procedures are followed except that users are advised as to which compute server they should use, rather than asking them to make their own, possibly uninformed, choice. Users are not required to use the "best" machine, but statistics have shown that the majority of users will do so.

The use of the initial user placement method for this project can be justified in several ways. Firstly, it is relatively simple to implement and install (it requires no modification to the SunOS kernel) and could be completed and evaluated within the time allowed for this project. Secondly, it seems to be a fairly new approach to load balancing and a comparison between this method and the more complex methods of initial process placement and migration would be worthwhile.

Load balancing systems that support the initial placement of individual processes normally require a reasonable amount of overhead. Whenever a new process is to be created, a policy decision is made and the remote execution facility is called into action. Although such systems have been beneficial to system performance, the extra work involved in starting each process may not be justified, especially for short-lived processes.

Process migration systems can adjust to the dynamically changing workload of a system more accurately than an initial placement method. Once a process has been started, a migration system is capable of moving that process to another, more suitable machine. However, the overhead involved in migrating a process far outweighs that of initial process placement.

One study [4] has shown that process migration gives no benefit to the majority of processes. Since the life time of many processes is limited to several seconds only, the overhead caused by a migration system is too large compared to the benefits of moving the process to a more suitable machine. In most cases, initial process placement is sufficient, although migration can be effective for long-running jobs.

One of the aims of this project is to determine whether there are similar reasons for placing new users on a lightly loaded machine, rather than placing each new process. The overhead of user placement is certainly less than that of process placement, since load balancing decisions are only made once per user login. If substantial improvements in system performance can be achieved by the use of the simpler method, its use would be justified.

3.2 The structure of clb

`clb` can be divided into three main sections, the gathering of load information, the algorithm for ranking the machines and the remote execution facilities. It is the combination of the first two components that form the load balancing policy. The third section is not a major part of `clb`, but instead existing UNIX remote execution software has been adapted.

The load balancing design decisions (described in the previous section) have been made for `clb` in the following ways.

- Load balancing

  In our network, a load balancing method is more applicable than load sharing since the compute servers are used by a large number of users and are infrequently idle. When allocating users to machines, `clb` will consider the extent to which each machine is being used.

- Initial placement

  Initial placement is performed at the level of user login sessions, rather than on a per process basis.

- Suboptimal

  Suboptimal algorithms must be used when allocating users to machines since it is impossible to know in advance what sort of workload people will be presenting to the system.

- Centralised information

  The `clb` load exchange mechanism requires that each machine determines its current load at regular intervals and stores this information on a globally common file system (that is, a centralised solution is used). The software that makes policy decisions is able to access this data from any machine. Since the rate at which users log in is fairly low (compared to the rate of process creation), the load information does not need to be updated too frequently. Therefore the file server does not become heavily loaded. If the centralised component becomes unavailable it is most likely that the entire network of machines will become unavailable, due to the high degree of interdependence between the hosts.

- Adaptive

  The `clb` policy algorithm uses only the most recently calculated load information from each machine, although it does take past performance into account. In our network, each machine has different capabilities, due to processor speed, memory, disk and network locality. When configuring `clb`, the system administrator will give a single numerical value that states how each machine is likely to perform.

- Bidding

  Since `clb` uses load balancing rather than load sharing, bidding is not an issue.

One restriction presented by `clb` is that the network's machines must be similar enough so that a user will not mind which host they use. For example, the file system must be sufficiently global so that the users' files are accessible from all machines, by using the same file names. Also, each machine must use compatible, or similar system programs. Any differences in this type of configuration may prompt the user to make their own choice of server when they are logging in.

Our network of Sun workstations follows these requirements, although one machine, *cantua*, is administered by a different organisation and sometimes uses different versions of software. Also, there exists a small degree of difference in each hosts devices, for example, a printer that is accessible from one machine may not be accessible from the others. The Condor [2] load balancing system counteracts these problems by referring to the originating workstation if a resource or file is not accessible from the remote host.

Chapter 4

Deriving the clb algorithm

Although the policy component of a load balancing system may be simple to implement, the design of the ranking algorithm can have a major impact on its effectiveness. A good algorithm can accurately predict the performance that a job will receive if executed on each of the available processors. To determine which processor will be used, the predictions are ordered and the processor with the best performance will be chosen.

In contrast to load balancing algorithms, load sharing systems (eg. Condor) typically use a boolean expression for each machine to determine when execution is possible. This may depend on the current number of active processes, the number of users, the amount of available memory or the length of the console's idle time. The combination of these factors produces a boolean decision as to whether the machine is idle, rather than a numerical ranking.

As an example, the Amoeba [11] load balancing system uses predetermined information about the speed (in millions of instructions per seconds) and architecture of each processor along with the available memory and current number of active processes on each. The best destination for a new job is found by determining which processor can allocate the highest number of instructions per second to the new process. This heuristic may work well in Amoeba where virtual memory is not used, but would not be sufficient in our system where disk and memory are a bottleneck.

4.1 System indices

In `clb`, the aim is to locate the machine that will give a new user the lowest response time. It is therefore necessary to derive a method for estimating a machine's future response time given that only information about the machine's past and current performance is available. To do this, an experiment was constructed to compare how well each of the system indices could be used to predict response times.

Firstly, a small sample of workload containing a few UNIX commands and a C program compilation was created. This sample was designed to be representative of the type of workload with which the system is normally presented. The execution time of this script of commands was kept small (5 seconds during periods of low load) so that it would not place excessive load on the system.

Next, the workload was executed on each of the compute servers at regular (5 minute) intervals throughout the day. The response time of the job was recorded, along with several of the main system indices. The following values were considered as possible response time estimators.

- CPU utilisation

  The utilisation of a processor gives an indication of the percentage of time that it is working, and conversely, the amount of time wasted due to a lack of work. Obviously (if all other factors are equal), a processor that currently has a low utilisation will give a better response time than one that is under heavy use.

  The limitation of utilisation figures is that a value of 100% (quite common during busy periods) does not give much information. For example, a processor may be executing a single CPU intensive job that uses all available processing power, or it may be executing 100 such jobs where each will only be given a fraction of the processor's time. The CPU utilisation figures can not accurately predict response times during periods of high workload.

- Device utilisation

  Utilisation of devices (eg. disks and networks) suffers from similar problems as that of the CPU. However, it is quite important to allow for the usage of devices since they may be the bottleneck of the system and will most certainly contribute to response times.

- Instantaneous queue lengths

  The length of the CPU and device queues gives a fairly accurate view of the workload in the system. However, the instantaneous lengths only describe the number of active processes at one point in time. These values can be dramatically affected by the peaks and troughs of the system workload. Although this may be suitable for short-lived processes, it is not desirable for large jobs or entire login sessions.

- Average queue length

  The concept of queue length can be extended to allow averaging over a period of time. Average queue lengths effectively smooth out the noise in the workload and give a more accurate indication of the system's performance over that interval. In the UNIX system there are three standard queue length averages that are calculated by the operating system. The 1 *minute load average* is a measurement of the average length of both the CPU and disk queues (including the currently executing process) over the past 60 seconds. The 5 and 15 minute averages measure the same queues, but over larger periods.

In `clb`, the UNIX load averages have been used as an estimation of the response time that each machine will offer. It is not important to know the exact response time of a job, as long as a relative ranking of the machines can be obtained. The 5 minute load average is used as a compromise since the 1 minute average can vary too quickly and can be influenced too much by the transient features of system workload. Conversely, the 15 minute average does not adapt quickly enough. Other instances of the use of load averages are given in studies described in [1] and [5].

4.2 Incorporating heterogeneity

Many existing load balancing systems assume that all the eligible machines are of the same type. Not only does this mean that they are assumed to have a common instruction set, but also have the same set of performance characteristics. In our network of Suns, the machines have homogeneous processors, but are *configurationally heterogeneous*, that is, they have different processor speeds, memory sizes, and amounts of local disk. Amoeba and Utopia both take this type of heterogeneity into account.

Since the configuration of a host influences the meaning of the load averages, `clb` accounts for these differences and adjusts load values to obtain a globally comparable evaluation of each machine. The concept of the *power factor* of a machine has been introduced as the amount by which a load reading for that host must be scaled. These factors are calculated by monitoring the response times given by a machine, with respect to its load averages.

To calculate the power factor for each machine, the response time measurements and load values obtained from the previous sample workload were combined in the following way.

$$Power\ factor = \frac{5\ minute\ load\ average}{response\ time}$$

If this result is averaged over a large number of samples a fairly accurate value will be obtained. The initial power factors for our system were found by calculating approximately 100 samples per day over 4 days.

The power factors currently in use for our compute servers are :

Host	Power factor
cantua	45
kahu	27
huia	14
ruru	12

The expected response time for the sample workload can then be computed at any time using the formula :

$$Estimated\,response\ time = \frac{5\ minute\ load\ average}{Power\ factor}$$

Note: The assumption that this relationship is linear would be justified during normal working conditions, however a further study of the load values and response times may result in a more suitable formula.

To make a load balancing decision, the estimated response time is determined for each of the compute servers, with the machine that has the lowest estimated response time being considered to be the "best". These response times are not meaningful as absolute values, but since the sample workload was designed to be a typical example, the ordering of the response times is relevant.

For example, with a load average of 5 on each machine, all the compute servers would be equally desirable if selection was based on load averages alone. By applying these power factors, an ordering occurs.

Host	Evaluation
cantua	1111
kahu	1851
huia	3571
ruru	4166

Note that `clb` evaluations are scaled by a factor of one thousand to make them appear significantly different from the normal unscaled load averages. Without this clarification, users were interpreting `clb` evaluations with their knowledge of load averages.

4.3 Verifying the clb algorithm

To verify that `clb`'s load balancing algorithm is effective in estimating the ranking of machines, the sample workload test was altered. Before executing the workload on each of the compute servers, the algorithm was used to estimate the ranking of response times. Its accuracy could then be measured by finding the real response times and calculating how often it had made the correct decision.

The initial results from this experiment were disappointing. Each of the four hosts were being chosen equally. That is, instead of choosing the most responsive host every time, it was guessing correctly in only 25% of the samples. The other machines that were measured as being second, third and fourth best, were also being selected by the algorithm, with the same probability.

Further analysis of the samples showed where the problem was occurring. During periods of low loading, the machines tended to give a fairly constant response time for the sample workload. For low loads, the load balancing algorithm should be altered to consider the machine's base level of processing power, rather than how it performs when presented with a higher load. If, for example, ruru had a load average of 0.01, then its evaluation would be 8. If cantua had a load of 0.30 it would have a higher evaluation of 66. This suggests that ruru would give much better response time than cantua, although since both machines are very lightly loaded, cantua (the more powerful machine) would be far better to use.

To solve this problem, a base load level was enforced. If a host reported that its load average was less than 2, the `clb` algorithm would be performed as if the load was equal to 2. With this modification, each host has a lower limit on its evaluation and during periods of low load, `clb` will act as if a static ranking system was used.

Host	**Lowest evaluation**
cantua	444
kahu	740
huia	1428
ruru	1666

There is no specific reason for using the cut off load value of 2, but by plotting the response time of the job against the current load average (see Figure 4.1), the point at which the response time starts to increase can be observed as being close to 2. It may be necessary to tune this value to suit each particular system.

4.4 Limitations

All suboptimal load balancing systems have some limitations. Several have been identified in the `clb` algorithm, although it may not be feasible or even possible to correct them.

- Although the new algorithm (with the lower bound) can select the best machine up to 60% of the time, this result may not be meaningful in practice. Since `clb` is used for the initial placement of users, it would desirable to estimate the response time the users would receive in

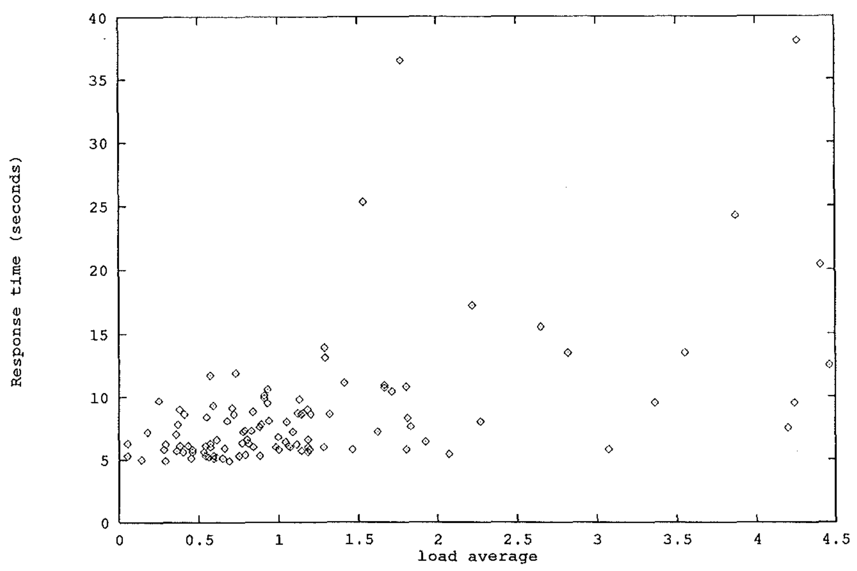

Figure 4.1: Response time vs the current 5 minute load average

several hours time. Although this is not possible in reality, a reasonable turnover of user logins would help to ensure that the system will remain as balanced as possible.

- The response times observed by a user will not only be affected by the load on their current machine, but may be influenced by other hosts in the network. For example, a disk intensive job may be executing on a lightly loaded processor, but the file server it depends on may be heavily loaded and will therefore degrade the response time. It would be difficult to counteract this problem by mathematically combining the load on all hosts, however since the power factors are calculated under normal workload conditions, these dependencies should already be accounted for.

- In most cases, the UNIX load averages are fairly good at predicting future response times, but problems occur in some situations. Firstly, when a CPU intensive process is executing, it will always be on the processor's run queue, hence it will contribute to the load values. However, due to UNIX's priority system, such processes will be given a low priority and will not affect the response time of other jobs to any great extent. Secondly, the presence of processes that are blocked in a high priority state for large periods of time will artificially increase the load.

Chapter 5

Implementation

In this chapter, the implementation of the clb load exchange mechanism, the load balancing algorithm and remote execution facilities will be discussed. The majority of this software has been written from scratch, however some of the existing UNIX remote execution facilities were modified when necessary. Firstly, the three main programs, gather, choose and Xchooser will be discussed. Later, the alterations to these programs and the extra support software will be examined.

5.1 The basic clb software

gather

Load collection and exchange in clb is performed by a daemon process called gather that remains resident on each of the eligible compute servers. At regular intervals (normally 45 seconds), gather reads the host's load indices and resource usage information from kernel memory and stores these details on a globally common file system. Currently gather collects the three load averages (1, 5 and 15 minute), the CPU utilisation, the paging rates and the amounts of file table space, process table space and swap space that are in use. The advantage of creating a specialised daemon process, rather than using the existing rstat facility is that extra indices can be added easily.

The load exchange interval of 45 seconds is fairly arbitrary, however incorrectly setting this value may have effects on the efficiency of clb. A small interval will cause a high amount of overhead due to the calculation and transmission of load data. A long interval will result in out of date information being used for decision making.

[1],[5], [6] and [12] note that a load exchange period of 5 to 10 seconds is optimal for the initial placement of processes. Consequently, they pay great attention to constructing efficient load transfer mechanisms. With the initial placement of users, this high rate of exchange is not justified since load

balancing decisions are made relatively infrequently and the length of each login session is considerably larger than the lifetime of most processes.

choose

The choose program is a straightforward implementation of the clb load balancing algorithm as discussed in Chapter 4. The current load information from gather is combined with the predetermined power factors to derive a globally meaningful load evaluation for each host. The name of the host that has the lowest evaluation will then be displayed.

choose would normally be used from the UNIX command line and can be combined with existing remote execution software such as xon and rlogin. For example, to login to the least loaded machine, a user would type

```
rlogin `choose`
```

To start an new invocation of the xman program one could use

```
xon `choose` xman
```

Two extra command line options can be specified in order to receive more detailed information. If the vals option is used, a complete list of the hosts and their clb evaluations is given. The all option results in a similar list, but also includes the indices that gather has collected.

Xchooser

The xdm (X-windows display manager) software allows users to login to a system via X-terminals. The Xchooser program is the component of xdm that locates the available compute servers and lists them so that the user may select the host they wish to use. No information about the loading of each machine is given.

To implement user placement, Xchooser was modified so that the clb load evaluations are displayed next to the host names, with the machines listed in load order rather than alphabetically. Users are therefore encouraged to always select the host that appears at the top of the list. The system administrator decided that all the hosts should be displayed, rather than only allowing users to login to the best machine.

Initial results showed that people were selecting the best machine about 80% of the time, although depending on the type of workload, this was at times as low as 60%. There are several reasons why people may choose to use a particular machine instead of using the best. Firstly, users may require access to a file or program that is only available on a subset of the machines. Secondly, some programs (such as Smalltalk) require large amounts of memory and processing power, and therefore users of these programs must be careful to use only the larger compute servers. Finally, users may have a favourite machine that they will always choose. This last reason is quite common and normally these users have no valid excuse for selecting their favourite.

5.2 Tuning the basic software

After a period of trial use, a number of modifications to the basic software were decided upon. This was necessary in order to improve the accuracy and reliability of the `clb` algorithm and to make the whole system easier to use.

The `choose` and `Xchooser` programs were modified to account for any fixed resource limitations that may occur. Although a particular host may have a low load evaluation, it may lack an important resource (such as swap space or process table entries) that would be required to support a new user. If this is the case, the host is considered unusable and is assigned the maximum possible load evaluation (similar to the MOS [1] system).

Initially, the `Xchooser` display was modified so that machines with resource limitations would be classified as 'Overloaded'. However, by observing the users' reaction to this message, it became necessary to give a more detailed explanation of the error. This problem arose when users were told that a machine was 'Overloaded' (due to (say) a lack of swap space), but did not believe the message since they were still receiving good response. After the more meaningful error messages were added, it appeared that people's faith in `clb`'s ability to predict response times had been restored.

To record instances of resource shortages, an error logging feature was incorporated into the `choose` program. If a machine is chosen to be the best but it has a fixed resource limitation, an appropriate error is logged to the `clb` error file. The next best machine is then considered as a possible choice. It is important to note that with this system, an error will only be reported if the machine was considered to be giving the best response, whereas the more heavily loaded machines with resource limitations are ignored.

The following list describes the types of limitation that are detected by `clb` and the reasons for doing so.

- **Out of swap space** — In a system that uses virtual memory, each process requires disk space to hold non-resident pages of memory. In SunOS this finite area of disk is known as *swap space*. Typically the processes belonging to one user would require from 1 to 5 megabytes of memory. If a host is close to exhausting its supply of swap space, new users should not be placed on that machine. If this was to happen, programs (owned by any user) may not be able to start or may be aborted if they are already executing.

- **Out of process table entries** — Each machine has a set number of process table slots. If the table is too full, new processes can not be created.

- **Out of file table entries** — As with the process table, the file table has a finite number of entries. If too many files are open, programs may be unable to start or will be aborted when they try to open new files.

- **System time too high** — If the processor spends too much time executing the code within its kernel, rather than user programs, the machine's performance will degrade. This normally indicates that the computer is performing a large number of disk or network operations.

- **Paging rate too high** — When the amount of active virtual memory exceeds the amount of physical memory in the machine, pages must be read and written to disk. If this happens at a high rate, the systems performance will be severely limited.

- **Host information too old** — To detect when a workstation has become unavailable (it has been shutdown or has crashed), `clb` will discard load information that is more than 2 minutes old. Any such machine will be excluded from further load balancing decisions.

- `clb` **evaluation too high** — If the host's load averages and hence, `clb` evaluation reach a certain value, the machine is considered too heavily loaded to use. By disallowing new users, `clb` reduces the chances of the load increasing any further.

5.3 Support software

Although the software that has been described so far forms an adequate load balancing system, several extra programs have been written with the aim of making `clb` easier to use.

At the system administrator's request, `xlb`, an X-windows version of the `choose` program, was created. `xlb` presents a small window that contains a list of the eligible workstations and their `clb` evaluations. If resource shortages occur, short error messages are displayed. By clicking on `xlb`'s title bar, a complete list of all the host information (as given by `choose all`) will be displayed.

To allow the system administrator to specify power factors and resource limitations of each compute server, a configuration file layout has been designed. For example, it is possible to specify that for a machine called `kahu` to be usable, it should have at least 8 megabytes of swap space free and that the process table may only be 95% full. If the administrator does not choose restrictions for all the fixed resources, sensible default values are used.

The configuration file is also used to specify the times at which each of the hosts will be available for general use. It was noted that for certain laboratory classes it was necessary for students to use a particular machine rather than selecting the least loaded. In this situation it is desirable that the remainder of the users in the network be directed away from that machine. Secondly, this time-tabling feature can be used to direct people away from a machine that is scheduled to be taken out of service. For a more detailed description of the configuration file layout, see Appendix A.

To aid the system administrator in calculating each host's power factor, the `perftimer` and `perfest` programs have been written. Initially, the power factors where derived by a rather complex method of collecting response times and load values and then using a statistical package to analyse them. The `perftimer` program automates this process by periodically executing the sample workload, collecting results and calculating the average power factor, all with very little human intervention. The `perfest` program reports the current estimation of the power factors.

`perftimer` obtains an accurate approximation of the power factors by executing a 1 minute long sample workload at 30 minute intervals throughout normal working days. Each power factor is averaged over a user definable period which would normally consist of the most recent 2 to 5 days worth of results. If the configuration of a host is altered in any way (for example, the amount of primary memory is increased), the `perftimer` program should be used to calculate the new power factors.

As an attempt to increase the number of users that can use `clb`, and to improve the transparency of the system, the Berkeley Domain Name Server (DNS) was modified. DNS is a daemon process that is responsible for translating host names to network addresses for the machines in the local domain. The modified version will convert any requests for the imaginary host name "best", into requests about the machine that is currently considered to be the best.

Since DNS is a very widely used service, `clb` can now by used in a large number of ways. Firstly, and quite importantly in our department, the Macintosh NCSA Telnet software can be instructed to access the local name server. Therefore, Macintosh users are able to login to the least loaded workstation by specifying the machine name "best". The only limitation with this feature is that NCSA Telnet will cache the network address associated with the name "best" under the assumption that this information will not change. As long as users choose to exit from Telnet when they have completed their session, the information will not be retained for too long.

A second possibility is that the Sun Network Information Service (NIS) can be configured so it will refer to DNS if it does not have its own information about a particular host. All UNIX programs that refer to host names will then be able to utilise `clb`. For example,

```
rlogin best
finger @best
```

This feature has been partially installed in our system, but due to the difficulty in configuring DNS, some machines (in particular, whio and cantua) are unable to use the modified name server.

Chapter 6

Methods of analysis

To determine whether `clb` is an effective load balancing system, suitable methods of measuring its impact on the system's performance must be derived. If a definite improvement can be seen when `clb` is in use, and these improvements can be attributed solely to the effects of `clb`, then balancing will have been a worthwhile exercise. This chapter includes a description of the two analysis methods that have been used.

To compare how the compute servers perform with and without the use of load balancing, the system was monitored over a six week period. During the first three weeks (13 days of valid results), `clb` was not used and people were required to make their own choice of machine. In the second three week period (13 valid days), the modified `Xchooser` program was installed, and the `choose` program made the initial placement of the undergraduate students' login shells. Also, the students were given menu options that allowed them to create a new command interpreter on either the best machine, or a specific compute server.

The first method of measurement involved the use of the 5 second sample workload (see chapter 4). This workload was executed on each of the compute servers and the response time was recorded. Although only three of the compute servers were eligible for user logins during the six week period, all four of them were monitored. This was necessary since the remaining machine, huia, was an important file server that was previously being over used.

To calculate the balance of the system at any given time, the variance in the response times over the four machines was calculated. If all machines were giving similar response times, the variance would be small and the system would be considered balanced. To ensure that this balance figure was comparable under all workload levels, the variance was divided by the mean workload at that time (the average of the response times over all hosts).

Figure 6.1 shows a typical example of both the mean response time of the job over the four machines, and the variance of the response times divided by the mean. These values could also be thought of as the total amount of workload in the system and the extent at which this workload is balanced

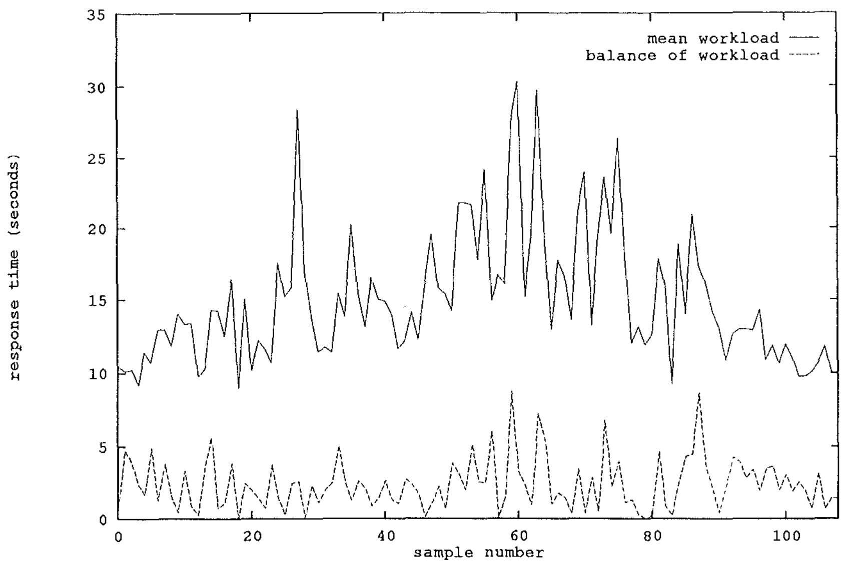

Figure 6.1: Average response time and balance of workload over 4 hosts

among the machines. If the average of these balance figures was less in the second three week period than in the first, then load balancing would have had an effect.

The second type of information that was recorded during the six week period involved the accumulation of most of the data files produced by the `gather` processes. This can be used to get an accurate idea of how the system's indices vary throughout the day, and to see whether any balance can be noted, that is, a balance in the load averages or resource usages. It is also possible to examine the under or over utilisation of the fixed resources.

Before any results can be considered valid, it must be confirmed that the majority of the users of the system are actually using `clb`, rather than making their own uninformed load balancing decisions. As previously discussed, most of the system's workload is generated by undergraduate students who normally use X-terminals and the `Xchooser` program. This type of login session tends to only last for a hour or two, depending on the student's timetable, hence load balancing is performed on a regular basis.

On the other hand, two groups of users don't always or can't take advantage of `clb`. Firstly, the postgraduate students and staff members are normally allocated their own Macintoshes or Sun workstations. They would generally use their own machines or the special research machine, rather than using `clb`'s advice about which compute server they should login to. Also, they tend to leave themselves logged on for large periods of time, but only generate a small amount of load when compared to that of an undergraduate class.

The second group of people are the non-Computer Science users on *cantua*, the SparcServer that belongs to the Computer Services Centre. These people do not have access to the Computer Science machines and must place their entire workload on cantua, possibly disrupting the balance of the network (from our point of view). For this to happen, the non-Computer Science users would need to create more than cantua's fair share of workload. This has not proven to be a problem, since the majority of the workload on cantua is generated by Computer Science students.

Chapter 7

Results

7.1 Balance of response times

One of the initial aims of clb was to find a improvement in the response time of the system as perceived by its users. Many other researchers (including [8], [5], [15] and [1]) have discovered that load balancing can give a dramatic decrease in the response times of their benchmark tests. Unfortunately, the realistic measurements taken within our system over the six week period do not conclusively show whether clb can perform in a similar way.

Figure 7.1 gives an indication of the average workload in the system for each day of the six week period of experimentation (both with and without load balancing). This diagram also shows these results after being averaged over each of the three week periods. It should be noted that the workload had increased slightly for the 'after' case, but due to the variability of the daily measurements, it would not be valid to estimate the amount of increase.

Figure 7.2 shows the balance of the machines and the average balance over the two periods. The graph suggests that in the second three week period, the balance has degraded slightly. However, this can not be considered as a valid result, since like the workload graph, the daily fluctuations are too great. The increase in workload and type of workload could also have an effect on the balance.

Although the lack of a definite result in the improvement of response times is disappointing, considering the reasons for the lack of evidence is useful. The following explanations have been considered and could be used for the future development of more accurate methods of measuring the effects of load balancing.

- It is not clear from the response time measurement whether any change in balance should be attributed solely to the effects of clb, or whether it was because the offered workload in the system had increased. Judging by the number of student assignments that were due in the second three week period, compared to the activity in the first three weeks, it was

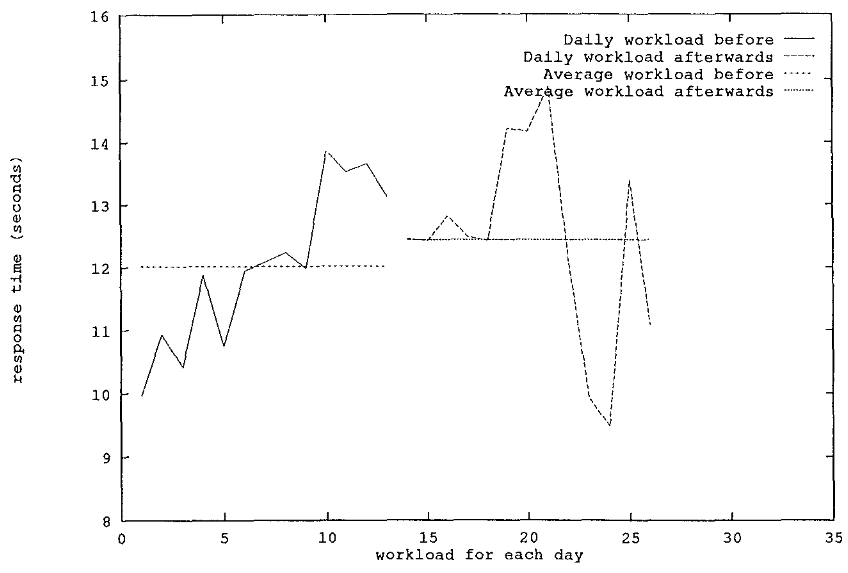

Figure 7.1: Daily workload over the six week period of experimentation

intuitively obvious that more work was being performed. No significant increase in the number of users could be noted, but the type of software being used in the second period (such as Smalltalk and Minix) tends to place considerable strain on the system.

- During the periods of low system load, the response time for the sample workload is fairly constant. It seems most likely that any improvement in system performance due to load balancing would only become apparent during peak periods of workload when response times can degrade severely. However, analysis of these periods is not always valid since the response time of the job during periods of high workload quite frequently depends on the length of time that a server is unavailable.

- For efficiency reasons, the sample workload was limited to being about 5 seconds in length. It is possible that a larger sample would give a more accurate measure of the response times, hence a better indication of the system's balance.

- It is not surprising that the initial placement of users can not give the same results as the initial placement of processes. In the small grain approach of many other load balancing systems, most jobs will complete within a few seconds or minutes after the placement decision has been made. With `clb`, a user may remain on the chosen machine for several hours after the decision. During such a login session, the dynamic workload

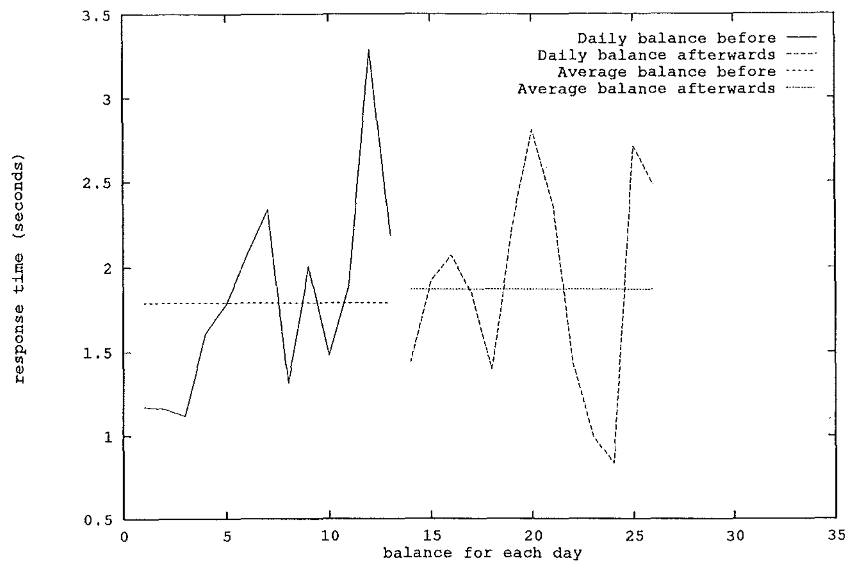

Figure 7.2: Daily balance over the six week period of experimentation

of the system may change considerably, especially in the situation where undergraduate laboratory classes arrive and depart.

7.2 Balance of the load averages

Since the initial experiments of the improvement in the average response times were not conclusive, a study of the system indices, eg. load averages and resource utilisation, was performed. It seems intuitive that the system's load averages can give a more accurate (numerical) indication of the workload presented to the system. Whereas the response time of a job (in particular the 5 second sample workload) can be highly variable during periods of high activity, the load averages give a definite value as to how much work is present in the system over larger periods of time.

Table 7.1 shows the average workload (the average of the 5 minute load values) in the system for both the three week periods. Assuming the `clb` did not have a adverse affect on the system, these results give a fair indication that the system's workload had increased. Three of the hosts recorded an increase in load, while the fourth machine remained at the same level.

Contrary to the increase in system load, the peak load values have tended to decrease. Table 7.2 shows the ranges of the top 0.5% of load values for each machine, both before and after `clb` was installed. On three machines, the high peaks have definitely been avoided, whereas the fourth machine, kahu, suffered

Host	Load before	Load afterwards
cantua	5.6	6.0
kahu	2.3	3.6
huia	2.2	2.2
ruru	1.0	1.3

Table 7.1: Average loads with and without load balancing

Host	Loads before	Loads afterwards
cantua	13.6 to 27.0	13 to 15.7
kahu	8.8 to 11.4	11.5 to 18.8
huia	17.9 to 27.6	7.2 to 12.5
ruru	4.7 to 9.9	4.8 to 6.1

Table 7.2: Peak load values with and without load balancing

several peaks of very high load. The probable reason for kahu's increase was that one class (COSC303) was limited to using this machine for one of their major assignments.

The main reason for the smoothing of the peak load values, was that with `clb` installed, users were given advance information about the hosts' loading. Without load balancing, a user would make an uninformed choice when logging in, possibly compounding the workload of a heavily used machine. During these periods of 'sky rocketing' load values, a machine will become unusably slow for several minutes. By examining the frequency of these events in the 'before' and 'after' cases, `clb` has proven to be very useful. In particular, the massive decrease in peak load on huia, a small but important file server, was independently noted by the system administrator.

An examination of the balance in load among the machines has also shown an improvement. Table 7.3 shows that with `clb` installed, the system's load (see table 7.1) when scaled by the appropriate power factors become much more evenly spread. These figures represent the balance of the load among the machines, with consideration of the processing capacity of each. Without load balancing, cantua was presented with more than its fair share of work, while kahu and ruru were underutilised. This was probably because users are aware that cantua is the fastest machine and will always choose it, regardless of how heavily loaded it may be. With load balancing installed, both kahu and ruru were used to a greater extent.

These effects can be verified by comparing Figures 7.3 and 7.4. These graphs are typical of the type of system balance experienced with and without load balancing. An attempt has been made to compare two days of similar

Host	Scaled loads before	Scaled loads afterwards
cantua	0.124	0.133
kahu	0.085	0.141
huia	0.157	0.157
ruru	0.083	0.108

Table 7.3: Average loads scaled by the power factors

workload, although in Figure 7.4 the overall loads are obviously higher.

Figure 7.3 shows a situation with load balancing in which cantua was presented with a considerable amount of work throughout the whole day. The three other machines were performing a relatively small amount of work over this period and their workload does not reflect the peaks and troughs of the whole system's load. At one point in the early afternoon, the loads on cantua reached an exceedingly high peak. This however was not shown by the other machines.

With `clb` installed (Figure 7.4), the system's load was more evenly spread among the machines, in particular, the less powerful machines were being better utilised. The peaks and troughs in the system's workload were being followed equally by all of the machines. If the power factor of each host was taken into consideration, the workload in this example is very well balanced.

7.3 Resource exhaustion

The use of `clb` in our network has given the advantage of restricting the periods of machine overuse and in detecting situations where configuration problems prevent a machine from operating to capacity. Because the `choose` and `Xchooser` programs consider each host's resource limits, many instances of resource exhaustion have been avoided. This has been a major advantage, especially since hosts can become temporarily unusable when fixed resource limits are reached.

During the six week period of system performance monitoring, resource limitations often became a hindrance. Without `clb` installed, there were 10 incidents where the monitoring software on ruru could not complete its job due to a lack of swap space. This resulted in invalid response time measurements since the sample workload would terminate with an error. With `clb` installed, this problem occurred only once, despite the increase in workload on ruru.

From the system administrator's point of view, `clb` can be a useful tool. The errors generated by the `choose` program give an indication of which hosts are being underutilised due to a resource limitation. For example, when `clb` was first in use (before the six week period), it was noted that about 24 user

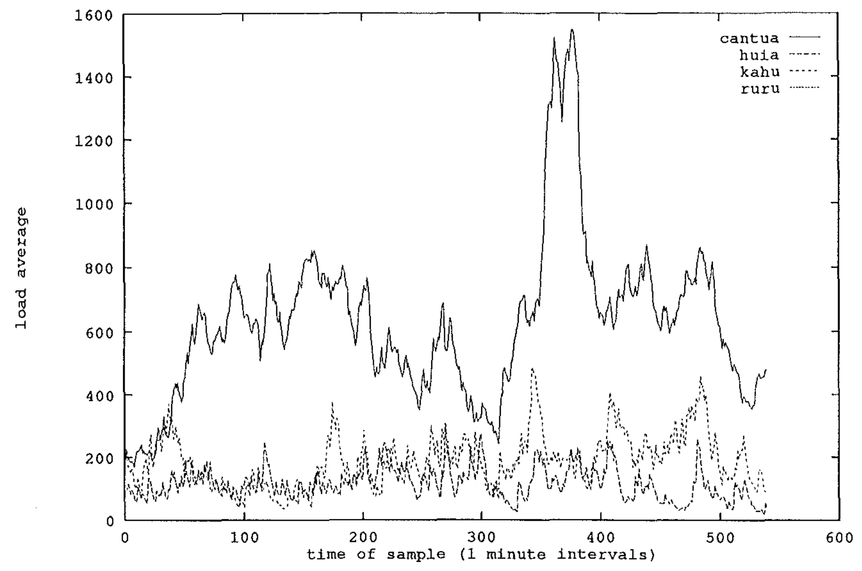

Figure 7.3: Load averages on a typical day without load balancing (starting at 9am)

logins per day were being transferred away from ruru, not because it was giving poor performance, but because of the lack of swap space. After the swap space had been increased, only 2 logins per day were being transferred. Similarly, the swap space and number of terminal lines on cantua were (indirectly) increased because of the extra workload placed on it by `clb`. In both cases, processing power was being wasted before the resources were upgraded.

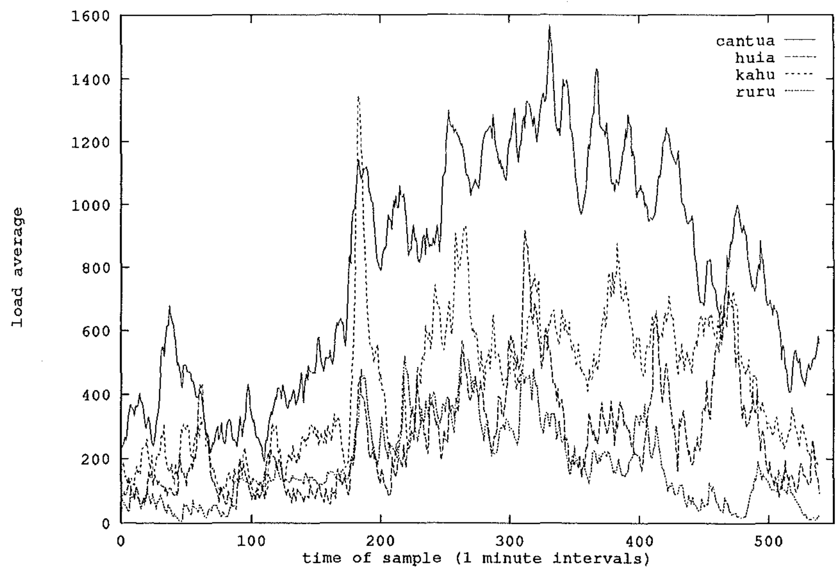

Figure 7.4: Load averages on a typical day with load balancing (starting at 9am)

Chapter 8

Limitations of clb and future work

The `clb` system has proven to be useful in several ways, but would benefit from further improvement. Also, since the methods used to analyse `clb` have not always given conclusive results, more accurate attempts at evaluation would be an advantage. In this section, the limitations of `clb` and possible future work will be outlined.

`clb` has only been studied in a real, working environment and as a consequence many difficulties were experienced when attempting to collect performance results. Our network of machines is constantly changing. Not only are the computers taken out of service at inconvenient times of the day, but the configuration of these machines has altered considerably throughout the year. These disruptions caused a large percentage of results (about 20%) to be lost or invalidated. By good fortune, no major modifications were made to the system during the six weeks of experimentation, although several days were lost due to machine down time.

To avoid such limitations of a real system, some researchers [1] [5] have evaluated their systems by the use of artificial workloads. With this method, the activity of users is emulated by automated sample login sessions. This will ensure that the workload remains constant throughout the experiments and that response times can be measured accurately. However, artificial workloads may not give a true indication of a machine's performance. It may be possible to evaluate `clb` in this manner by generating workload during off peak periods, (on weekends and at night).

So far, `clb` has only been analysed in one network, with one type of workload. The balance of the system may depend on the number of users and how often they log in. In a different situation, for example, where a small number of users create a large amount of workload, the effects may be different. The `clb` source code has been made publically available via the USENET news service, so it should be possible to make further judgements about this.

If `clb` is accepted as being useful and reliable, it may be possible to remove almost all choice from users, forcing them to use the best machine. This would be as simple as restricting `Xchooser` to display only one machine name, and removing the remote host menu from the undergraduates' window managers. In exceptional circumstances, they should be allowed to make their own choice (for example, in operating systems laboratories), although it could be made more difficult to do so. People seem to choose a machine because of personal preferences rather than a legitimate reason. Future students that are brought up with the idea of load balancing may be more willing to change.

A few improvements to the `clb` software have been suggested. Firstly, the system could be altered to detect and give warnings about more resource limitations, for example, the number of pseudo-terminals and the amount of free disk space. Also, the unnaturally high load averages due to the occurrence of 'stuck' processes could be counteracted to give a more accurate machine evaluation.

A proposed extension to the concept of initial placement of users is the *migration of users*. To do this, the standard UNIX command shells would be modified to allow their migration. When a machine becomes heavily loaded, a particular user of that machine would be chosen and their shell would be transferred to a lightly loaded host. This system would have the advantage of dynamically adapting to the workload as well as causing little overhead.

In the near future, `clb` is to be incorporated into a distributed simulation package [14]. When this work is complete, the choice of which machines will perform the simulation will be under the control of `clb`.

Chapter 9

Conclusion

With the implementation of `clb`, the initial user placement model of load balancing has proven to be worthwhile. The system was relatively easy to implement and install, with only a few changes to existing user level software. The load balancing mechanism results in only a small amount of overhead. Such a system would be a great benefit in a network of machines where all the available processing power was needed.

Unfortunately, the method used to detect `clb`'s impact on *response times* was not accurate enough to give a conclusive result. Since the user placement model affects the performance of the system as a whole, rather than the performance of single benchmark processes, the traditional method of measuring improvement was not sufficient. It is most likely that the response times have improved during peak periods, rather than in the average case.

The *distribution* of the offered workload has shown an improvement with the use of `clb`. Instead of allowing one machine to become overloaded, while others remain idle, `clb` will ensure that each host is allocated its fair share of work. This gives the major advantage of avoiding the situation where the load on a machine becomes exceedingly heavy.

An novel advantage of `clb` is that it can help avoid the undesirable situation where programs are aborted due to the lack of a fixed resource. If one of a host's resources is almost exhausted, the load balancing algorithm will direct new users away from that machine. The error file generated during these situations can be used by the system administrator as a guide to allocating the available resources among the machines.

This project has shown that `clb` can easily be added to a Networked Operating System, and will bring about a definite improvement in the balance of the load. Also, the initial placement of user login sessions does not suffer from the significant overhead of process placement and migration.

Appendix A

clb installation and user guide

A.1 Introduction to clb

`clb` (Canterbury Load Balancer) is a simple, but effective load balancing system for use in a network of Sun workstations. It is designed for an environment where there are a small number of computers serving a large number of users, rather than the more common model of one user per workstation. Instead of allocating users to a completely idle machine, `clb` allocates users to the least loaded machine, possibly sharing it with others users.

`clb` uses the idea of initial placement of user login sessions and requires no modification to the SunOS kernel so is easy to install.

`clb` does not guarantee a remarkable increase in response times during periods of average workload, but if set up correctly it will improve the utilisation of the machines and should help to avoid poor response during peak periods. That is, it will not allow one machine to become heavily overloaded while others remain idle. It is also capable of informing the system administrator of limitations in a machine's configuration. For example, if a machine is capable of handling a higher workload, but is limited due to the amount of swap space available, then the system administrator will be informed.

It is assumed that the network of machines can provide a sufficiently global file system so that users will not mind which machine they are placed on.

A.2 The components of clb

`gather`

In order to be part of `clb`, each compute server must run a daemon process called `gather`. At regular intervals, `gather` will read the host's current statistics (load averages, CPU utilisation, paging rates etc) from kernel memory and store this information in a publically available disk file. Client programs are free to use this information to make their own load balancing decisions.

`choose`

The most basic client program, `choose`, uses the information about each machine to rank them in the order of expected response time. If no parameters are passed to `choose`, then the name of the machine with the lowest expected response time (the host that should give the best performance) is displayed. If the `vals` option is given, then a list of hosts and their numerical ratings is given. Finally, the `all` option will display the complete set of information about each host. For example,

```
% choose
cantua

% choose vals
cantua = 524 kahu = 740 ruru = 1666

% choose all

Summary for cantua

Swap space used is 151292/386912
Process entries used is 220/2058
file entries used is 808/7110
Page in Kb = 7
Page out Kb = 19
idle time = 34%
user time = 50%
system time = 14%
load 1 = 2.32, load 5 = 2.36, load 15 = 2.46
Age of data file is 34 seconds
Eval function = 524


Summary for kahu

Swap space used is 30932/102912
Process entries used is 122/1034
[...]

Summary for ruru

[...]
```

`choose` can be used in the following ways,

```
% rlogin `choose`
[...]

% xon `choose` [...]
```

The choose program will also detect the situation where the best host is unusable because of the near exhaustion of a fixed resource. For example, the host may not have sufficient swap space to accommodate a new user. If this happens, the next best host is considered and an error is logged to an error file that should be periodically examined by the system administrator.

The following type of output is given.

```
[...]
cantua: Out of swap space at Thu Jun 4 16:38:04
cantua: Out of swap space at Thu Jun 4 16:38:34
cantua: System time too high at Thu Jun 4 18:30:06
cantua: System time too high at Thu Jun 4 19:15:40
cantua: Out of swap space at Fri Jun 5 12:59:33
cantua: Out of swap space at Fri Jun 5 13:39:07
cantua: Out of swap space at Fri Jun 5 13:39:40
cantua: Out of swap space at Fri Jun 5 13:41:06
ruru: Out of swap space at Fri Jun 5 13:41:06
cantua: Out of swap space at Fri Jun 5 13:41:32
ruru: Out of swap space at Fri Jun 5 13:41:32
cantua: Out of swap space at Fri Jun 5 13:41:44
ruru: Out of swap space at Fri Jun 5 13:41:44
[...]
```

xlb

`xlb` provides a graphical interface to the `choose` command. `xlb` is an X-windows program that displays a list of all of the currently available hosts and their `clb` evaluations. If any host has a resource shortage, then a short error message is displayed next to the machine name. Clicking on the title bar will give a more detailed list of the machines (similar to `choose all`). This program will be primarily used by a system administrator.

DNS

The Berkeley name server (the Domain Name Server) has been modified to recognise a new virtual machine name. If a DNS request is for the name *best*, `clb` is queried to determine which machine should give the best response time. The information about this machine is returned to the requester.

This service can be used by any software that queries DNS, for example, `nslookup` and `telnet`. If NIS is configured to use DNS, then the following commands should also be possible,

```
% ping best
% rlogin best
% finger @best
etc.
```

Xchooser

The `Xchooser` program from the `xdm` (X-windows display manager) system has been modified to display `clb` information. The original `Xchooser` simply lists the hosts in alphabetical order, whereas the modified version lists them in the order of their `clb` evaluation, with the best host at the top. If any machine has a resource shortage, the appropriate error message is given. A new "LoginBest" button is supplied to make logging in easier.

A.3 Setting up the configuration file

Before `clb` can make any meaningful load-balancing decisions, it must know the configuration of your network. A configuration file is used to specify the following information about each of the available hosts.

- The system administrator must decide on a *powerfactor* for each machine to inform `clb` of the relative processing power of each machine (see later).
- A time-tabling feature allows machines to be available or unavailable at various times.
- Optionally, resource limitations for each machine may be specified.

The following is an example of a configuration file. The first half lists the machines and their limits, while the second half specifies the availability times.

```
kahu    perf 27 swap 8000 proc 5% limit 3500
cantua  perf 45 swap 10% page 1000 limit 5000
huia    perf 14 cpu 80% page 200 limit 2000
ruru    perf 12  proc 5% file 5% limit 3000

from monday 1:45pm to monday 4:00pm
           use kahu ruru
from friday 1:45pm to friday 4:00pm
           use kahu ruru
else use cantua kahu ruru
```

The four available machines, *kahu*, *cantua*, *huia* and *ruru* have the relative power (*perf*) factors of 27, 45, 14 and 12 respectively. That is, cantua is almost twice as powerful as kahu, which in turn is twice as powerful as huia and ruru. The remaining values on each line state the desired fixed resource limits for the host.

- `swap` – the minimum amount of swap space that must remain for a machine to be considered usable, either specified as an absolute value (in Kbytes) or a percentage. If this figure is set too low, the host may frequently run short of swap space. If set too high, the host may be underutilised. The default is 10%.
- `proc` – the minimum number (or percentage) of process table entries that must remain for a host to be considered usable. Defaults to 10%.
- `file` – the minimum number (or percentage) of file table entries that must remain. Defaults to 10%
- `cpu` – the maximum allowable CPU system time. Defaults to 80%
- `page` – the maximum allowable paging rate in Kbytes/second (sum of paging in and out). The default is to ignore the paging rates.
- `limit` – the maximum `clb` evaluation before this machine will be classified as overloaded. This value should be chosen by observing the hosts performance and determining the point at which it becomes too heavily loaded to use. Initially, no limit will be used.

The second section of the configuration file specifies when each machine will be available. This is useful for when a machine is to be taken out of service at a particular time. The general format for each line is,

```
from <start-day> <start-time>
to <end-day> <end-time> use <hosts>
```

where <start-day> and <end-day> should be one of `mon`, `monday`, `tue`, `tuesday`, `wed`, `wednesday`, `thur`, `thursday`, `fri`, `friday`, `sat`, `saturday`, `sun` or `sunday`, and <start-time> and <end-time> are of the form,

```
hour:minute [ am | pm ]
```

If neither am or pm is specified, then a 24 hour clock is assumed. The final line of the form

```
[else] use <hosts>
```

specifies which hosts should be used by default.

To compile this configuration file, use

```
clbconfig [configuration filename]
```

If no file name is given, the file `config` will be used.

A.4 Calculating the power factors

To aid in calculating the power factor for each machine, the two programs `perftimer` and `perfest` have been written. The `perftimer` program should be run on each machine to determine what sort of response that machine is capable of giving. To do this, a sample script of programs is executed and the response time of that job with respect to the current load averages is recorded. If this is done at regular intervals over normal working days, the average of these samples will give a good representation of the machine's power.

Due to the way it is calculated, the power factor will take all the system's components into consideration, rather than one single factor such as the raw CPU speed. This is quite important since for some machines it is the network or the file servers that can add significantly to response times. If one of the system's component is changed, it may be beneficial to recalculate the power factor. `perftimer` requires the user to specify the number of samples that should be kept, so as to only use the newest information.

The `perfest` program will display the current power factor for each machine. It is not necessary to supply these exact values in the configuration file, although they should be given in their correct proportions. It may also be necessary to artificially alter the power factor for a machine if that machine has a resource limitation that can't be easily fixed. For example, if a fast machine has a very small of amount of memory, it may give good response (according to `perftimer`), but will page heavily if used too much. In this case, the amount of memory should be increased, although decreasing its power factor could be used as a temporary measure.

The script that `perftimer` executes should contain about 1 minute (response time as measured at low loads) of work that is typical of the type of workload handled by your system. Any temporary files should be created on the file server that is used by the majority of people. If set up correctly, the power factor of each machine will reflect the power of the machine as perceived by most of the users.

To use `perftimer`, put the following entry in the *crontab* of each of the machines. You should use a suitable user code eg. *daemon* rather than *root*

```
0,30 9-17 * * 1-5 perftimer 50
```

This will execute `perftimer` at 30 minute intervals throughout the normal working days and will only remember the last 50 samples. It would be an advantage to run the job on different machines at different times to avoid the effects of concurrency, that is, a second machine would have the entry,

```
5,35 9-17 * * 1-5 perftimer 50
```

As a guide to what typical power factors would be, here is a description of each of our machines. *cantua* (45) is a SparcServer 690MP (2 processors) with

128Mb of memory and a large amount of local disk. *kahu* (27) is a SparcStation 2 with 64Mb of memory that holds the majority of user files. *ruru* (12) is a SparcStation 1+ with 40Mb of memory and local disk. *huia* (14) is an ELC with 16Mb of memory that holds most of the system binaries. The absolute value of these power factors may be meaningless in your environment as only the ratio between them is important.

A.5 Installation

The file `CLB.tar` contains the following directory structure,

src	– the source code for `gather`, `chooser`, `xlb` etc.
xdm	– the diffs for `chooser`
named	– the diffs for `named`
man	– extra manual pages for `clb`
perf	– the sample workload files for `perftimer`

The majority of `clb` can be compiled by putting the correct pathnames into both the Makefiles and the header file `src/clb.h`. However, the modifications to `xdm` and `named` require that you already have their source code in an appropriate directory and know how to install them. The following steps should be taken to install `clb`.

1. type `cd src` and edit the file `clb.h`. Each of the 6 file names or directories (`DUMPPATH` etc.) should be created by hand and their details put into this file.

    - `DUMPPATH` is the name of the directory that `gather` will store the host information in. You must create this directory on a file system that is readable and writable by all machines that run the `gather` process.
    - `ERRORFILE` - The file that `choose` puts its error information into. Again, this must be writable by all hosts. The system administrator should regularly examine and truncate this file.
    - `CONFIGDIR` - The directory containing the configuration file.
    - `PERFDIR` - The directory used by `perftimer` when estimating power factors. This must be writable by all hosts.
    - `WORKLOAD` - The script of sample workload used by `perftimer`

2. type `make`. The complete set of binaries will be built but not installed. Most of the programs could be placed in `/usr/local/bin` although `xlb` would be better in `/usr/local/X11/bin`.

3. The `gather` process should be started on each eligible machine at boot time (ie. in `/etc/rc.local`) (it would be worth thinking carefully about which machines should be made eligible). The following type of entry should be used,

```
if [ -f /usr/local/bin/gather ]; then
    /usr/local/bin/gather
          > /dev/console 2>1 &
fi
```

4. If you want to, install `xlb.resources`, in your `.Xresources` file or in `/usr/local/X11/lib/X11/app-defaults/XLB`

5. If you wish to use the modified `Xchooser`, change into `clb`'s `xdm` directory and edit `Makefile`. The constants `XDM`, `XDMBUILD` and `TOPDIR` must be changed to point to the directory containing your existing xdm source code, the directory in which you build xdm, and the top directory as used by `xmkmf`. Typing `make` in `clb`'s xdm directory will make the necessary alterations and then recompile the `chooser` program. You should install the new `Chooser.ad` file in the appropriate Xresources file. NOTE: These diffs are for the xdm system that comes with X11R5.

6. If you wish to modify the DNS system (Version 4.0), change into `clb`'s `named` directory and edit `Makefile`. Set up `NAMED` to point to your directory that contains the existing source code. Typing `make` will make the necessary changes and recompile `in.named`. It is assumed that you already know how to use DNS.

7. The manual pages (in the `man` directory) should either be copied into the correct man directory eg. `/usr/local/man/man1` or your `MANPATH` environment variable can be altered to include `clb`'s man directory.

8. The configuration file should be written. To determine the power factor for each machine, you should use the `perftimer` program. As a temporary measure, you could make an educated guess about the relative processing power of each machine. The following type of configuration file would be a good start. Remember that `clbconfig` should be used to compile this file every time it is changed.

```
host1 perf ??
host2 perf ??
host3 perf ??
...
hostn perf ??

use host1 host2 host3 ... hostn
```

A.6 Known bugs

- Due to the fact that `clb` depends on UNIX's load averages, a system that has processes caught in a high priority state will have an unnaturally high evaluation.
- This system works well with a small number of compute servers, however if a large number ($>$ 50) hosts were used, the amount of load data being transferred to the file server may become excessive. To limit this problem, the constant `DUMPINTERVAL` in `clb.h` could be increased.

Bibliography

[1] Amnon Barak and Amnon Shiloh. A distributed load balancing policy for a multicomputer. *Software: Practice and Experience*, 15(9):901–913, September 1985.

[2] Allan Bricker, Michael Litzkow, and Miron Livny. Condor technical summary. Technical Report 1069, Computer Sciences Department, University of Wisconsin-Madison, January 1992.

[3] F. Douglis. Experience with process migration in Sprite. In *Workshop on Experiences with Building Distributed and Multiprocessor Systems*, October 1989.

[4] D. L. Eager, E. D. Lazowska, and J. Zahorjan. The limited performance benefits of migrating active processes for load sharing. In *Proceedings of the 1988 ACM SIGMETRICS Conference on Measurement and Modelling of Computer Systems*, pages 63–72, Sante Fe, New Mexico, USA, May 1988.

[5] Domenico Ferrari and Songnian Zhou. An empirical investigation of load indices for load balancing applications. In *Performance '87*, pages 515–528, North-Holland, 1988.

[6] A Goscinski. *Distributed Operating Systems: The Logical Design.* Addison-Wesley, 1991.

[7] Anna Hac. Load balancing in distributed systems: A summary. *Performance Evaluation Review*, 16(2-4):17–19, February 1989.

[8] Robert Hagmann. Process server: Sharing processing power in a workstation environment. In *Proceedings of the 6th International Conference on Distributed Computing Systems*, pages 260–267, Cambridge, MA, USA, May 1986.

[9] D. A. Nichols. Using idle workstations in a shared computing environment. In *Proceedings of the Eleventh ACM Symposium on Operating System Principles*, pages 5–12, Austin, Texas, 1987.

[10] Peter Smith and Paul Ashton. Load balancing by allocation of user login sessions. Technical Report COSC 5/92, Department of Computer Science, University of Canterbury, New Zealand, 1992. (Submitted to the 1993 Australian Computer Science Conference).

[11] A. Tanenbaum. *Modern Operating systems.* Prentice-Hall, Englewood Cliffs, New Jersey, 1992.

[12] M. M. Theimer and K. A. Lantz. Finding idle machines in a workstation-based distributed system. *IEEE Transactions on Software Engineering*, 15(11):1444–1458, November 1989.

[13] M. M. Theimer, K. A. Lantz, and D. C. Cheriton. Preemptable remote execution facilities for the V-system. In *Proceedings of the Tenth Symposium on Operating System Principles*, December 1985.

[14] V. Yau and K. Pawlikowski. AKAROA: a package for automating generation and process control of parallel stochastic simulation. Technical Report COSC 04.92, Department of Computer Science, University of Canterbury, New Zealand, 1992.

[15] Songnian Zhou, Xiaohu Zheng, Jingwen Wang, and Pierre Delisle. Utopia: A load sharing system for large, heterogeneous distributed computer systems. Technical Report CSRI-257, Computer Systems Research Institute, University of Toronto, November 1991.