Load Balancing by Allocation of User Login Sessions

Peter Smith and Paul Ashton

TR–COSC–05/92

Department of Computer Science
University of Canterbury
Private Bag 4800
Christchurch
New Zealand

Load Balancing by Allocation of User Login Sessions

Peter Smith and Paul Ashton
Department of Computer Science
University of Canterbury
Christchurch, New Zealand
email: paul@cosc.canterbury.ac.nz

Abstract

Most current load balancing systems are based on process placement or process migration. We introduce placement of user login sessions as a load balancing approach, and compare it to the process placement and process migration approaches. A load balancing system based on user placement is described, and some experimental results are given. We conclude that user placement has several advantages over process placement and process migration, and that user placement should provide good load balancing for some types of workload.

1 Introduction

An important function of process management in a loosely-coupled distributed system is the allocation of processes to processors. A major goal in this allocation is to achieve *load balancing*, that is to ensure that the system workload is spread around the available processors so as to make full and balanced use of the computing power available.

Although some experimental distributed operating systems (such as Amoeba [14]) do perform load balancing automatically and transparently, most operating systems currently used on loosely-coupled distributed systems are network operating systems that provide little if any support for load balancing. Several systems designed to achieve some degree of load balancing for network operating systems have been developed, and these include Condor [3], Butler [12] and Utopia [16].

We have developed `clb` (Canterbury load balancer), a load balancing system for use with the SunOS network operating system. `clb` differs from other load balancing systems in that it is based on allocating entire user login sessions to machines, rather than on allocating processes to machines or on migration of executing processes. We have also performed experiments to determine the effectiveness of `clb`. These experiments were done using a production workload, and indicate that `clb` was reasonably effective in evening out the load in our environment.

The rest of this paper is structured in the following way. Section 2 provides background, in terms of: related work on load balancing; and the computing environment for which we wanted to provide load balancing. Section 3 describes the user placement approach to load balancing, and compares it to the process placement and process migration approaches. Section 4 discusses `clb`, a load balancing system that we have implemented based on the user placement approach. Section 5 gives some preliminary experimental results on the quality of the load balancing achieved by `clb`. Finally, we present some conclusions in Section 6.

2 Background

Two aspects of the background to our work are now discussed. First, other work in the load balancing area is summarised. Second, we describe our computing environment, and give our reasons for investigating load balancing.

2.1 Related work

The processor organisation used in a distributed system determines how load balancing can be performed. The two principal models for the organisation of processors in distributed systems are the workstation model and the processor pool model [13].

In the workstation model, each user interacts with the distributed system through a workstation. Each workstation has a screen and keyboard and can run user processes. Typically, the computational resources of a workstation are reserved for the exclusive use of the workstation "owner"—the person currently sitting at the workstation. While a workstation can provide good response for most of the workload generated by its owner, it is desirable to execute compute-intensive tasks on a remote processor. As there are a substantial number of workstations idle most of the time (for load balancing purposes a workstation is idle if it does not have an owner), load balancing can be achieved by off-loading compute-intensive tasks onto idle workstations. Load balancing issues for work-station systems are: deciding which processes should be executed remotely; finding idle workstations; achieving transparent remote execution of processes; and handling the situation where a user begins using a formerly idle workstation that is executing a "foreign" process.

Many load balancing systems are based on the workstation model. Condor places batch jobs on available idle workstations [3]. If a workstation becomes busy, any foreign process executing on the workstation is migrated to a currently idle workstation. Several systems are able to execute interactive processes on idle workstations, including Butler [12], Sprite [6] and V [15].

In the pure processor pool model, users interact with the system through graphics terminals (such as X-terminals) which do not run user processes. All user processes are executed on compute servers. The major goal of load balancing for such systems is to distribute the workload between the compute servers so that performance is maximised. Amoeba [14] is a distributed operating system based on the pure processor pool model. In Amoeba, load balancing is performed when processes are assigned to processors.

Hybrid systems also exist, in which each user has his or her own workstation, and compute servers are available for running compute-intensive tasks. Appropriate distribution of workload between the compute servers is a major issue for load balancing in hybrid systems, as is the selection of the processes that should be executed on the compute servers. Load balancing systems that operate in this type of environment include NEST [1], MOS [2], the Process Server [9] and Utopia [16].

Load balancing systems based on any of the three models discussed above distribute load by initial process placement, that is by selecting the processor on which a newly created process will run, and/or by process migration, that is by moving a compute-intensive process from a heavily-loaded processor to a lightly-loaded processor.

In process placement, a decision must be made when a new process is created as to whether that process should be executed locally or remotely. Because of the overhead involved in starting a process remotely, and because most processes are too trivial to be worth executing remotely ([5], [11]), care has to be taken to execute remotely only the relatively few processes that are likely to require substantial processing resources. The three main methods for selecting the processes that are to execute remotely are:

1. To require the user to specify which commands should be run remotely, by providing some sort of `remote` command [1], [12], [15]. This approach is not transparent, however.

2. To modify certain programs to enable them to make use of remote execution facilities [3], [4], [6], [9]. This approach requires programs to be at least relinked, if not modified, before they can benefit from load balancing.

3. To have the command interpreter maintain a database of commands that can be executed remotely [16]. This approach creates extra overhead for the large majority of processes that are not executed remotely.

If process migration is available, then all newly created processes can be started locally, with processes considered for migration only when they seem likely to be compute-intensive. This approach is advocated by Cabrera [5] and has been used in MOS, which considers processes for migration only after they have consumed one second of CPU time [2].

Whether done by process placement or process migration, implementation of a load balancing system requires considerable effort. Of the two, process migration systems are much more complex because of the difficulty of transparently moving a process from one machine to another at some arbitrary time.

2.2 Our environment

The main computing system in the Department of Computer Science at the University of Canterbury is a network of Suns, all of which run the SunOS network operating system. For the most part, access to the Suns is gained in one of three ways: from an X-terminal; through the keyboard and console of a Sun; or through remote login, most often from a Macintosh.

Eleven Suns are available for use. Of these, six can be classified as workstations—discless machines with limited memory used mainly by the single user currently logged into the machine's console. The remaining five machines can be classified as servers—machines with local disc and a substantial amount of memory. File sharing is provided by Sun NFS, with all machines seeing basically the same file system, particularly with respect to user files.

Systems like the one just described are common, particularly in university environments, and are best classified as following the hybrid model. For the most part, users logged into a Sun workstation console run most of their user processes on that workstation, whereas users accessing the Suns from X-terminals or through a remote login run most processes on one of the compute servers.

The major objective for `clb` was that it should balance the load placed on the compute servers by students using the 40 or so X-terminals. A constraint was that it was not possible to modify the SunOS kernel.

3 The user placement approach to load balancing

To achieve load balancing between our compute servers, we decided to implement a load balancing system based on allocation of user login sessions. The basic approach is that whenever a user logs in to the system (through an X-terminal, for example) their initial processes (their command interpreter process in particular) are started on the most lightly-loaded compute server.

This *user placement* approach is quite different from the process placement and process migration approaches used in nearly all other load balancing systems, and has several advantages over these other approaches:

- The user placement approach is considerably less complex than the other two approaches. This means that implementing user placement is much easier than implementing process placement and (in particular) process migration.

- The overhead of the user placement approach is likely to be much lower than that of either of the other approaches. First, although all approaches involve the collection of information on machine loads, the frequency with which load information is updated can be much lower for user placement because of its coarser grain (user placement operates on entire user sessions rather than on entire processes or less).

  Second, the user placement approach uses the load information only when a user logs in. In the process placement approach, every new process must be considered as a candidate for remote execution, and in the process migration approach a host must at regular intervals consider migrating one of its processes elsewhere. The overhead of starting a process remotely and of migrating a process must also be considered.

- In the user placement approach, the "best" machine can be selected transparently for a user at login time. In some systems based on process placement, users must use some sort of `remote` command if they want to have a process created on a lightly-loaded machine.

The main potential disadvantage with the user placement approach is that because it operates at a coarser grain than the other approaches the user placement approach will be slower to adapt to changing workload conditions, resulting in poorer load balancing for some types of workload. If new users login relatively frequently, then a load balancing system based on user placement will assign users to lightly loaded systems and reasonable load balance should be achieved. If, on the other hand, all users login at the start of the week and remain logged in for the entire week then a load balancing system based on user placement would be much less effective than load balancing systems based on process placement or process migration.

We believe that many systems have high enough user turnover to make user placement effective, particularly if "user logins" are taken to mean both initial logins and the creation of new command interpreter processes. University systems used by undergraduates (such as our own system) have this type of workload as most undergraduates login for an hour or two at a time as their timetables permit.

Also, results reported in [7] indicate that the load balancing achievable by user placement may in many situations be close to that of the other two approaches. In the work described in [7], analytical and simulation models were used to compare the load balancing performance of the process placement approach to that of the process migration approach, that is the performance of a (relatively) coarser grain approach was compared to that of a finer grain approach. Two of the major conclusions reached were: (i) "there are likely *no* conditions under which migration could yield *major* performance improvements beyond those offered by non-migratory load sharing," and (ii) "under some fairly extreme conditions, migration *can* offer *modest* additional performance improvements."

A limitation of the user placement approach is that it is not particularly suitable for use on systems that follow the workstation model, as each user in such systems logs into the workstation at which he or she sits. A way of using the user placement approach in systems that follow the workstation model would be to provide a facility for starting a new command interpreter process on an idle workstation.

4 Implementation

We now describe `clb`, the load balancing system that we have developed. The major components of `clb` are: a program that gathers load information; `clb_eval`, a library function that uses the load information to rank machines; and programs that call `clb_eval` in performing user allocation.

Load balancing systems should preserve "process" transparency and "file" transparency. Unlike many load balancing systems, `clb` does not need to contain any features aimed at providing transparency, because:

- As user placement operates on entire login sessions, related sets of processes execute on the same machine thereby giving a high degree of process transparency.
- The transparency provided by many distributed file systems is such that load-balancing systems can rely on a process having the same view of the file system on all machines. The assumption that the operating system provides a (sufficiently) global file system is made in both `clb` and Utopia [16].

4.1 Gathering of load statistics

In all load balancing approaches, statistics on workload must be gathered and distributed. In `clb`, load information is stored in a directory that is accessible to all machines. Current load information for machine `host` is stored in a file called `host` in that directory. A very similar method of gathering and distributing load information is used in Butler [12], although it would be easy to use instead other popular techniques such as using a "load server" to maintain the information, as in Condor [3] and the Process Server [9], or having a process on each machine responsible for maintaining load information, as in MOS [2] and Utopia [16].

Every compute server in our system runs a `gather` process, which updates every 45 seconds the load information file for that machine. The load information collected provides information on current system load, and includes:

- The 1, 5 and 15 minute load averages. The Unix "load average" is the mean (over some period) of the total number of processes that are running on the CPU, queued for the CPU, or waiting for an I/O operation to complete [10].
- The CPU time spent in user mode, spent in the kernel and spent idling during the previous 45 seconds.
- Rates of page in and page out operations during the previous 45 seconds.
- The amount of swap space available and the amount currently in use.
- The number of process table entries available and the number currently in use.
- The number of file table entries available and the number currently in use.

4.2 Assessing machine load

To assess the current load on a machine, programs that perform user allocation make use of the `clb_eval` library function. Given a host name, `clb_eval` returns: an integer that roughly equates to *expected response time* for that host; and an indication of any resource shortages that exist on the specified host.

The expected response time for a machine is calculated from the current value of its five minute load average and a factor that represents the relative power of the machine. The second parameter is necessary, as in determining the machine that will give the lowest response times, machine capacity must be taken into account as well as current machine load. For example, a 20 MIP machine with 5 active processes will most likely give much lower response times than a 5 MIP machine with 2 active processes. Also, all other things being equal, a machine that has a user's files on local disc will most likely give lower response times than a machine that has to access the files remotely.

The relative power of a machine is represented by its *power factor* (P). The value of P for each machine was calculated by running many times a script of typical commands, and recording for each run the 5 minute load average (L) and the response time (R) of the script. The value of P is taken as $1000L/R$ averaged over many runs of the script of commands. Values for P are precomputed, and are stored in machine configuration files.

Intuitively, $1/P$ gives, for our script of commands, a number proportional to the expected seconds of response time per unit of the 5 minute load average (the more powerful a machine, the higher its power factor). Therefore, by dividing the current 5 minute load average of a machine by the value of P calculated for it, we get a number directly proportional to the expected response time for the script of commands, had it been run on that machine at that time. The machine with the lowest expected response time can be expected to provide the best performance.

Estimated response times are based on load averages rather than instantaneous loads because of the rapid fluctuations in the instantaneous load [1], [8]. Response time estimates calculated in ways similar to ours are used in MOS [2] and Utopia [16].

In addition to the expected response time, `clb_eval` reports any resource shortages that might cause performance problems. For example, if a machine has very little swap space remaining then there is no point placing a user on that machine even if its load average is currently very low, as the lack of swap space will make it difficult to run any programs. Checks are made by `clb_eval` to ensure: that a machine has sufficient free swap space, process table entries and file table entries; that the paging rate and the percentage of CPU time spent in the kernel are sufficiently small; and that the machine's load file has been updated recently, to check that the machine is not down. These checks are based on thresholds specified for each machine in its configuration file.

4.3 User placement

Two programs make use of `clb_eval` to perform user placement: `chooser`, which is used in placement of users logging in through X-terminals, and `choose`, which is used in placement of command interpreters.

`chooser` is used by `xdm` (the X Display Manager), the program responsible for handling user logins from X-terminals. The standard version of `chooser` presents the user with a list of machines from which the user selects the machine to which they wish to login. The `xdm` process on the selected machine then prompts the user for his or her user code and password. Upon a successful login, a window manager process and a command interpreter process (in our default configuration) are created on the selected machine.

We could have changed `chooser` so that it uses `clb_eval` to evaluate each machine, and then automatically logs the user on to the *best machine*, that is the machine that has the lowest expected response time amongst those machines that have no resource shortages. Our system administration preferred to retain the flexibility of the original `chooser`, so a different approach was taken. The modified `chooser` still lists all of the available machines, but the machines are ordered so that all machines with resource shortages appear at the bottom of the list tagged with an appropriate message (such as "Out of swap space"), and all other machines are at the top of the list in increasing order of expected response time. The *best machine* is therefore at the top of the list. Users have been advised to always select the machine at the top of the list.

To allow a user to run a process on the best machine, we wrote the command line program `choose`. `choose` prints the name of the best machine, and it can be used in conjunction with remote execution programs like `rsh` and `xon` to have a program executed remotely on the best machine. If `choose` finds that the machine with the lowest expected response time has a resource shortage then it writes the details to a log file. This log file should be examined periodically by a system administrator as it contains information on possible configuration problems, such as machines that are underutilised because of insufficient swap space. The logging of instances of resource exhaustion has turned out to be an important and unexpected benefit of `clb`. Examples of the usefulness of the log are given in Subsection 5.4.

The `xbest` shell script uses `xon` to run the specified program on the best machine as determined by `choose`. The standard X-terminal startup script was changed to use `xbest` to start up an instance of the command interpreter on the best machine. Also, the standard window manager menus now have a "Best" option which starts a command interpreter on the best machine.

So far we have discussed user placement for users of X-terminals. Workstation users can also use the "Best" menu option and the `xbest` command. For a user logging in from a Macintosh or a dumb terminal, we are in the process of adding a new entry type to the DNS name server such that, when entries of this type are looked up, the address of the best machine will be returned. Such a user could then simply specify a machine called (say) "best", and he or she will end up logged into the best machine.

4.4 Implementation summary

Load information for each machine is recorded in a file by a `gather` process on that machine. User placement is provided by the following methods, all of which make use of the load information:

- The use of `xbest` to place the command interpreter started up when a user at an X-terminal logs in.

- The use of a version of `chooser` that has been modified to encourage people to login to the best machine. The user's window manager process is placed on the machine selected.
- The "Best" menu item for starting up a new command interpreter process.
- (In the future) a DNS entry type that evaluates to the address of the best machine.

5 Evaluation

We performed an initial evaluation of the effectiveness of `clb` by making "before" and "after" measurements on a production workload. Evaluation using a production workload seems to be unusual, with most reported evaluations of load balancing systems being based on analytical or simulation models ([7], [11]), or on use of artificial workloads ([1], [2], [8]).

The major difficulty in using production workloads in evaluation studies is that the variation in workload between measurement sessions must be taken into account as a secondary factor. Nevertheless, the truest test of a load balancing system is its performance for a production workload.

We describe first our evaluation method and difficulties encountered with it, then give some preliminary results from the experiments that were performed.

5.1 Evaluation method

The basic evaluation method was to record each minute the current load information for each compute server. Because `gather` is executed every 45 seconds, one quarter of its updates were not recorded. Information was gathered between 9 a.m. and 6 p.m. on weekdays during term time. Data was collected on thirteen days with `clb` not running on the system, and on eleven days with `clb` running. User placement was applied only to users logging in from X-terminals, so some portion of the Computer Science workload was not under the control of `clb`. Also, one machine (cantua) is operated by the Computer Services Centre and is the only Sun available to many non-Computer Science users.

The four compute servers monitored were cantua, kahu, ruru and huia (the fifth, whio, is not accessible to most students). Although only cantua, kahu and ruru were considered as candidate machines for user placement by `chooser` and `xbest`, huia was also monitored because it is an important file server that we felt was being overused and would become less used with the addition of `clb`. The power factors calculated for these machines were: 45 for cantua, 27 for kahu, 12 for ruru, and 14 for huia.

One problem with the data collected was that it did not capture adequately information on the characteristics of the workload given to the system by its users. Nevertheless, we felt that the user workload was greater during the period after `clb` was added because: in that period students were working on assignments that involved use of resource-intensive programs such as Smalltalk; and the period of `clb` use that was monitored was at the end of a term and contained several deadlines.

Table 1: Before and after average loads and load to P ratios

	Before		After	
Machine	Load	Load / P	Load	Load / P
cantua	5.6	0.124	6.0	0.133
kahu	2.3	0.085	3.8	0.141
ruru	1.0	0.083	1.3	0.108
huia	2.2	0.157	2.2	0.157

Despite the lack of user workload information, preliminary analysis gives reason to believe that `clb` has had a positive effect on system performance.

5.2 Overall effect on five minute load averages

Load information is summarised in Table 1. Overall load averages were greater for the period in which `clb` was running. Taking machine capacities into account the load was 14% higher. Despite this overall increase in load, the peak loads after `clb` was added to the system are in the main lower than before it was added. Evaluation of the top 0.5% of load averages in the "before" and "after" cases showed that: on cantua the peak loads were a little lower despite an increase in average load; on huia and ruru the peak loads were much lower (18 to 27 before, 7 to 13 after; and 5 to 10 before, 5 to 6 after); and on kahu the peak loads were higher (9 to 11 before, 11 to 19 after). A major reason for kahu's increased peaks was that a third year class could use only kahu for part of a major project. Overall then, despite higher workload after `clb` was added, the workload peaks for most machines were lower.

The load to power ratios in Table 1 indicate that overall the distribution of load became more balanced after the addition of `clb`. In the "before" case, cantua carried a proportionally greater load than kahu or ruru. After `clb` was added, kahu and cantua were carrying about the same load. The load on ruru had increased, but it was still carrying less than its share. Part of the reason for this was that some logins (about two per day) were being diverted from ruru because of a shortage of swap space.

5.3 Comparison of two load profiles

Load profiles provide detailed information on the variation of the five minute load average during a single day. Figure 1 is a load profile for a "before" day, and Figure 2 is a load profile for an "after" day.

These figures highlight two observed differences between the "before" and "after" load profiles. First, Figure 1 contains a big workload peak on cantua without any corresponding increase in workload on the other machines. A similar peak was observed on another "before" day, but none were observed for "after" days, indicating that `clb` helped to avoid such peaks. Second, Figure

Figure 1: Load levels during a day in which `clb` was not operating

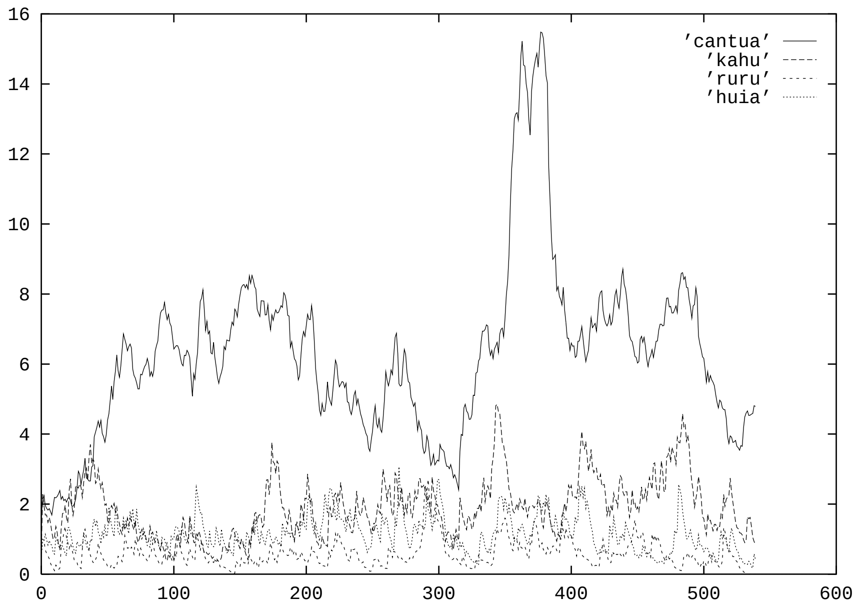
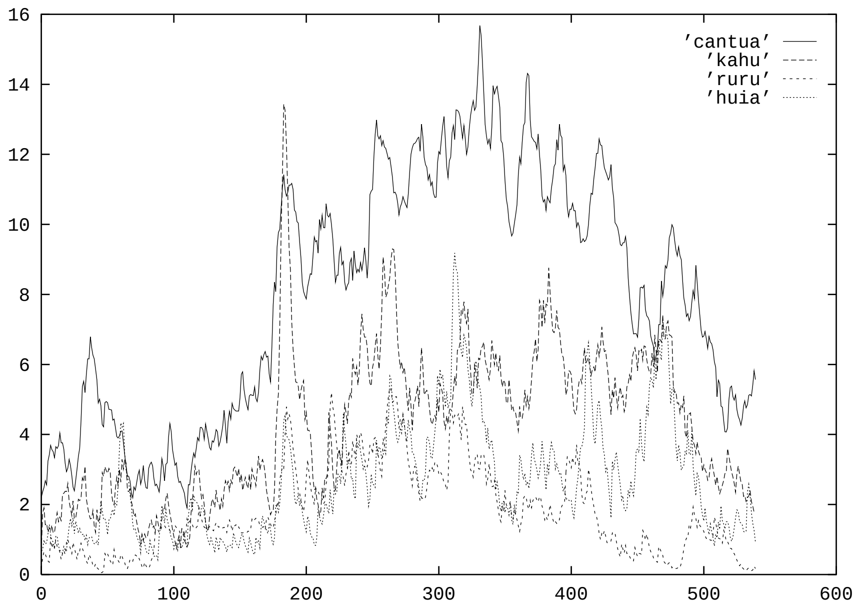


Figure 2: Load levels during a day in which `clb` was operating



2 shows a better distribution of load, with better use made of huia, kahu and ruru. This trend was observed over many "before" and "after" load profiles.

5.4 Instances of resource exhaustion

Experiments have indicated that `clb` helps to avoid resource exhaustion and that it also helps to detect poorly configured machines. During the "before" period, on 10 occasions load information could not be recorded on ruru because of a lack of swap space, whereas in the "after" period this happened only once (despite an increase in average load on ruru), indicating that `clb` was effective in avoiding swap space exhaustion on ruru.

Also, during an earlier period of use, `clb` showed that ruru had too little swap space. During that period, there were an average of 24 logins per day that should have gone to ruru but did not because ruru was low on swap space. During our experiments, which were performed after the swap space had been increased, this average was 2 per day.

6 Conclusions

We have introduced user placement as an alternative to the well established load balancing approaches of process placement and process migration. User placement is much simpler to implement than either of the other two approaches and has much lower overhead. Also, each major way of implementing process placement has one of the following problems associated with it: lack of transparency, limitation of scope, or overhead for every process created. The user placement approach does not suffer from any of these problems.

The effectiveness of user placement as a load balancing technique depends on the system workload. We have concluded that where there is a reasonable user turnover, user placement should be very effective. Analytical and simulation studies are needed to determine the types of workloads that are balanced well by user placement, and how effective user placement is on such workloads compared to process placement and process migration. Experiments using `clb` indicate that a student workload is one which user placement can distribute in a well balanced way.

Experience with `clb` has helped in verifying some of our expectations of user placement. `clb` was not difficult to implement, compared to process placement and process migration systems. Initial analysis of data from experiments indicates the `clb` has improved load balance on our system. Further experiments, where the workload is either artificial or a production workload whose composition is recorded accurately, will help to confirm our initial results.

A novel feature of `clb` is that the availability of resources of a fixed size (such as swap space, process table entries and file table entries) is taken into account when the best machine is selected. This feature ensures that users are not placed on a machine that has a low load, but which has a severe shortage of (say) swap space. Also, a log is maintained that records instances of the best machine having a shortage of some fixed size resource. The information in this log can be used in identifying machines that are poorly configured, and has proved to be a valuable and unexpected benefit of running `clb`.

User placement may not have been explored in the past because of the widespread use of systems that follow the workstation model. With X-terminals becoming more common, however, interest in the user placement approach can be expected to increase, primarily because user placement can be easily added to existing systems that may not provide any other form of load balancing.

References

[1] R. Agrawal and A. K. Ezzat. Location independent remote execution in NEST. *IEEE Transactions on Software Engineering*, SE-13(8):905–912, August 1987.

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

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

[4] David Butterfield and Gerald Popek. Network tasking in the Locus distributed system. In *Proceedings of the 1984 Summer Usenix Technical Conference*, pages 62–71, Salt Lake City, Utah, USA, June 1984.

[5] Luis-Felipe Cabrera. The influence of workload on load balancing strategies. In *Proceedings of the 1986 Summer Usenix Technical Conference*, pages 446–458, Atlanta, Georgia, USA, June 1986.

[6] F. Douglis. Experience with process migration in Sprite. In *Workshop on Experiences with Building Distributed and Multiprocessor Systems*, pages 59–72, October 1989.

[7] 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 Measurment and Modeling of Computer Systems*, pages 63–72, Sante Fe, New Mexico, USA, May 1988.

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

[9] 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.

[10] Samuel J. Leffler, Marshall Kirk McKusick, Michael J. Karels, and John S. Quarterman. *The Design and Implemenation of the 4.3BSD UNIX Operating System.* Addison-Wesley, 1988.

[11] W. E. Leland and T. J. Ott. Load-balancing heuristics and process behaviour. In *Proceedings of Performance '86 and ACM SIGMETRICS 1986*, pages 54–69, Raleigh, NC, USA, 1986.

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

[13] A. S. Tanenbaum. *Modern Operating Systems.* Prentice-Hall, Englewood Cliffs, New Jersey, 1992.

[14] A. S. Tanenbaum, R. van Renesse, H. van Staveren, G. J. Sharp, S. J. Mullender, J. Jansen, and G. van Rossum. Experiences with the Amoeba distributed operating system. *Communications of the ACM*, 33(12):46–63, December 1990.

[15] M. M. Theimer, K. A. Lantz, and D. C. Cheriton. Preemptable remote execution facilities for the V-system. *Proceedings of the Tenth ACM Symposium on Operating System Principles*, pages 2–12, December 1985.

[16] 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.