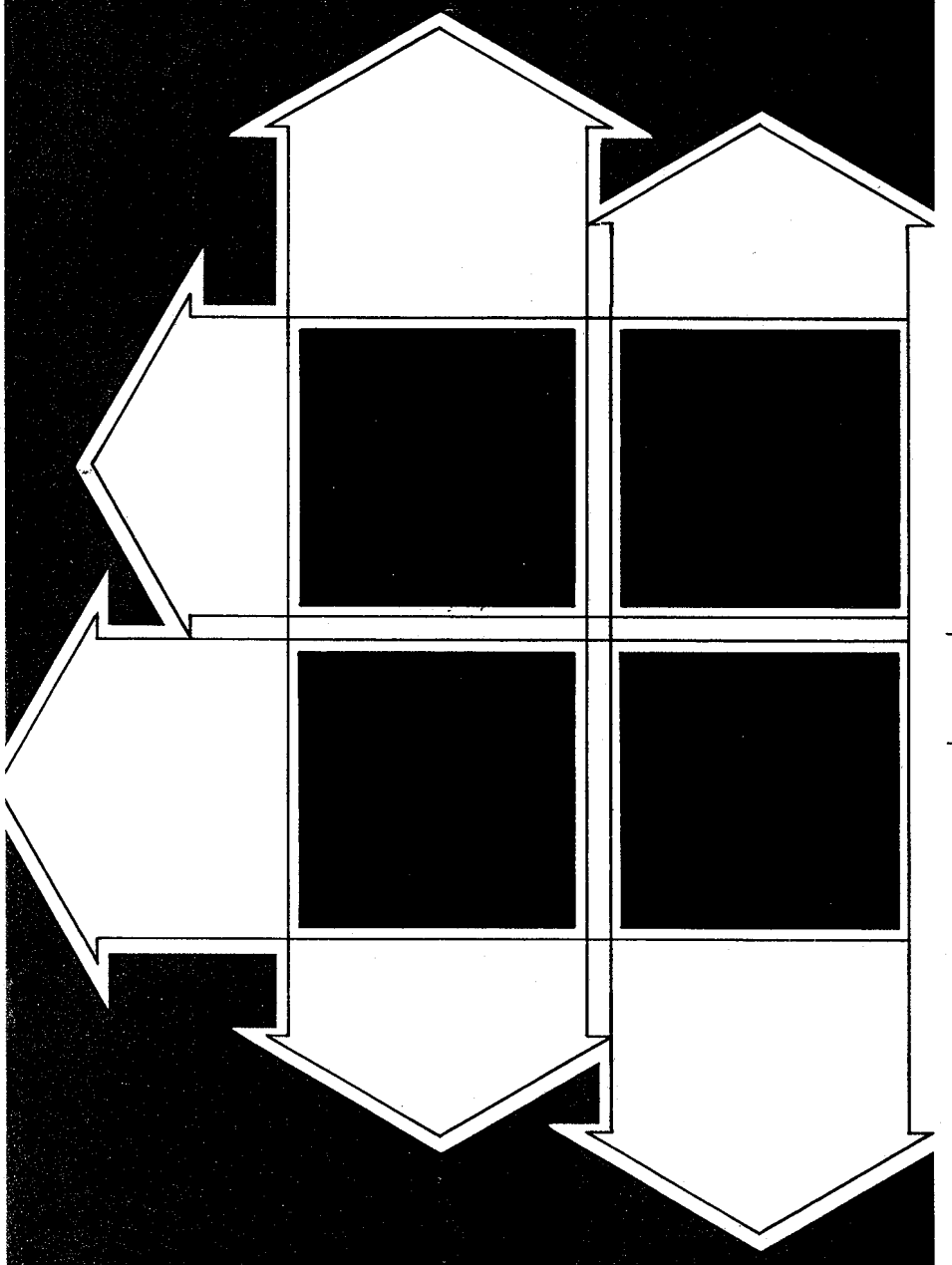
PROCEEDINGS



The 7th International Conference on

Distributed Computing Systems

Berlin, West Germany
September 21–25, 1987

SPONSORED BY

THE COMPUTER SOCIETY OF THE IEEE

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC

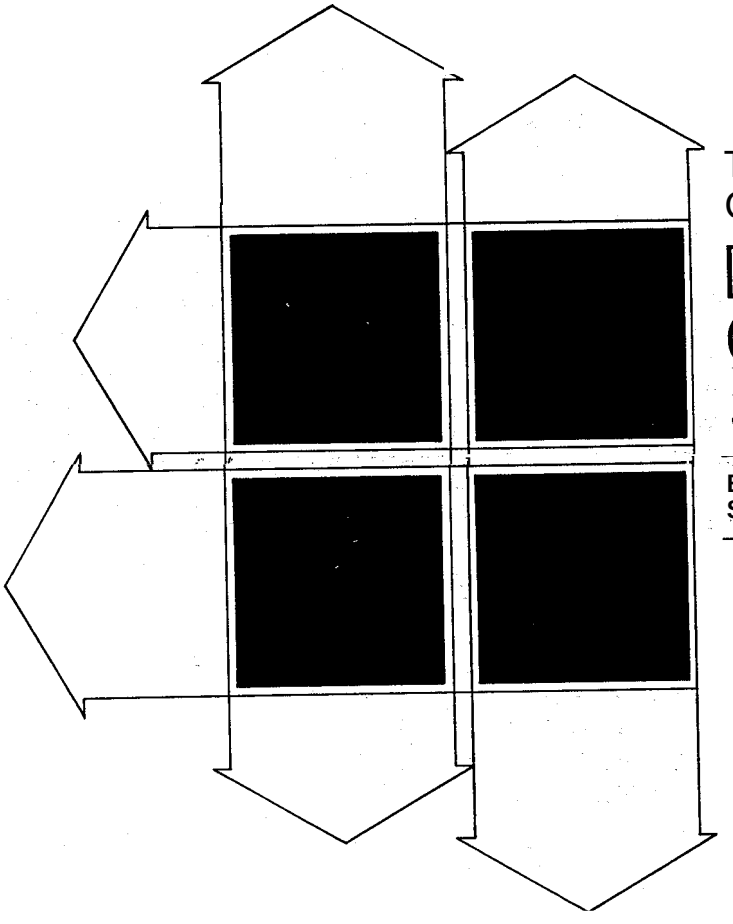
IEEE

Edited by: R. Popescu-Zeletin
G. Le Lann
K.H. (Kane) Kim

Computer Society Order Number 801
Library of Congress Number 87-80437
IEEE Catalog Number 87CH2439-8
ISBN 0-8186-0801-3
SAN 264-620X

L AND ELECTRONICS ENGINEERS, INC

COMPUTER SOCIETY PRESS



The 7th International Conference on Distributed Computing Systems

Berlin, West Germany
September 21–25, 1987

SPONSORED BY

THE COMPUTER SOCIETY OF THE IEEE

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC

IEEE

Edited by: R. Popescu-Zeletin
G. Le Lann
K.H. (Kane) Kim

Computer Society Order Number 801
Library of Congress Number 87-80437
IEEE Catalog Number 87CH2439-8
ISBN 0-8186-0801-3
SAN 264-620X

THE COMPUTER SOCIETY OF THE IEEE

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

IEEE COMPUTER SOCIETY PRESS

The papers appearing in this book comprise the proceedings of the meeting mentioned on the cover and title page They reflect the authors' opinions and are published as presented and without change, in the interests of timely dissemination Their inclusion in this publication does not necessarily constitute endorsement by the editors, Computer Society Press of the IEEE, or The Institute of Electrical and Electronics Engineers, Inc

Published by Computer Society Press of the IEEE
1730 Massachusetts Avenue, N W
Washington, D C 20036-1903

3

Cover designed by Jack I Ballestero

Computer Society Order Number 801
Library of Congress Number 87-80437
IEEE Catalog Number 87CH2439-8
ISBN 0-8186-0801-3 (paper)
ISBN 0-8186-4801-5 (microfiche)
ISBN 0-8186-8801-7 (case)
SAN 264-620X

Order from: Computer Society of the IEEE
Terminal Annex
P O Box 4699
Los Angeles, CA 90051

Computer Society of the IEEE
13, Avenue de l'Aquilon
B-1200 Brussels
BELGIUM

IEEE Service Center
445 Hoes Lane
P O Box 1331
Piscataway, NJ 08855-1331

IEEE

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC

The 7th International Conference on **Distributed Computing Systems**

Sponsored by :

The Computer Society of the IEEE

The Institute of Electrical and Electronics Engineers, Inc.

In Cooperation with:

Hahn-Meitner-Institut Berlin GmbH

Gesellschaft fur Informatik e.V.

Supported by:

Senat von Berlin

Sparkasse der Stadt Berlin West

Siemens AG

IBM Deutschland GmbH

Nixdorf Computer AG

Digital Equipment GmbH

Deutsche Bank Berlin AG

Standard Elektrik Lorenz AG

Scheduling Remote Processing Capacity In A Workstation-Processor Bank Network

Matt W Mutka and Miron Lzvny

Department of Computer Sciences
University of Wisconsin
Madison. WI 53706

ABSTRACT

This paper addresses the problem of long term scheduling of a group of workstations and a processor bank.. Long term scheduling manages the allocation of remote processing cycles for jobs that execute for long periods and require little interaction or communication., It extends the computing capacity a user sees beyond the capacity of his/her workstation. We assume that each workstation is under the full control of its user, whereas, the processors that constitute the processor bank are public resources.. Therefore, a workstation can be allocated for remote processing only if its user does not perform any local activity.. In the paper we present a new long term scheduling algorithm, the *Up-DownAlgorithm,* and a set of performance criteria for evaluating these types of scheduling algorithms. Using these criteria and traces of usage patterns of 13 workstations we evaluate the algorithm and demonstrate its efficiency and fairness.. We analyze the performance of the Round-Robin and the Random algorithms using the same criteria and workload, and show that the new algorithm out performs the other two. While all the three algorithm provide the same throughput, the Up-Down algorithm protects the rights of light users when a few heavy users try to monopolize all free resources..The two other algorithm do not maintain a steady quality of service for light users in the face of an increasing load of heavy users

1. Introduction

Currently, many computing professionals have personal workstations for research, software development, and engineering applications.. These powerful stations are considered private resources under the contxol of their users. However, in order to provide access to common resources and to enable information exchange, these private resources are interconnected by one or more local area networks to form an integrated processing environment.. The total processing capacity of such an environment can be very large. As an example, a portion of the computing environment at our department consists of 75 private workstations, 8 multiuser hosts, and a 20 node partitionable multicomputer. All of these resources are interconnected through two token ring networks and two Ethernets [1]. Multiuser hosts provide access to resources for users without workstations The partitionable multicomputer, called the Crystal Multicomputer [2], consists of 20 VAX®-11/750s connected by a 80 Megabit/sec Proteon ProNet token ring [3]. Crystal provides a vehicle for research in distributed systems, and extra computing cycles It can be viewed as a *Processor Bank* that serves as a source of computing cycles.,

The total capacity of our research environment is more than 180 MIPS. (see Table 1) This large capacity is comparable to that of several large supercomputers. An analysis of the usage pattern of this distxibuted capacity shows that a large portion of the capacity is not utilized [6]. When workstations are not used by their owners, they can be sources of cycles for users who want additional cycles.. There are users that would like to expand their computing capacity beyond their local workstations and use the available computing cycles.. We call networks that allow users at workstations to expand their capacity *LOcal Computing capacity eXpanded (LOCOX)* networks. Figure 1 illustrates a LOCOX network. Jobs submitted to the LOCOX network can be divided into two categories: interactive and background, Interactive jobs require frequent input and a small amount of CPU capacity.. Background jobs are computationally intensive and run for long periods of time without any interaction with the users,.Users would benefit if they could receive a portion of the remote computing capacity for their background jobs, Experience from observations of the Crystal Multicomputer shows that there are long running jobs that often consume several hours of processing time. One user has been observed to have a single job that has consumed about 2 months of cpu time on a VAX11/750 [7]! We have also observed a steady supply of background jobs from another user,. This user has maintained a queue of 20-30 background job requests over a period of five months where each job ran about 2 hours on a VAX11/750 [8].

Resource	Kind	# of Machines	MIPS Per Machine**	Capacity (MIPS)
Multiuser Host	VAX 11/780	2	2	4
Multiuser Host	VAX 11/750	6	1	6
Workstations	MicroVAXII†	75	2	150
Crystal	VAX 11/750	20	1	20
Total				180

** Based on the values given for individual machines in [4].
† Roughly the capacity of VAX 11/780 [5].

Table 1: Portion Of Research Computing Capacity At Wisconsin.

The management of the huge distributed computing capacity of a LOCOX network creates a wide spectrum of scheduling problems to consider.. In this paper we address one resource management aspect of this environment called *long term scheduling*. Long term schedulers manage the allocation of remote processing cycles for jobs that execute for long periods and require little interaction with the workstation from which the job was submitted for execution.. They *extend* the computing capacity a user sees beyond the capacity of his/her workstation.. The emphasis of long term scheduling is not the balancing of work among computing resources already allocated, as is done in *middle term scheduling,* but the high level view of providing extsa computing service when available, This management is at the user level and not at the job level Unlike *short term scheduling,* it is not concerned with the internal management of the processes of individual jobs. Short term scheduling is the allocation of the processor on a workstation to processes in its run queue. The goal of long term scheduling is to give all users a fair share of avail-

* This research was supported in part by the National Science Foundation under grant MCS-8105904

® VAX is a trademark of Digital Equipment Corporation

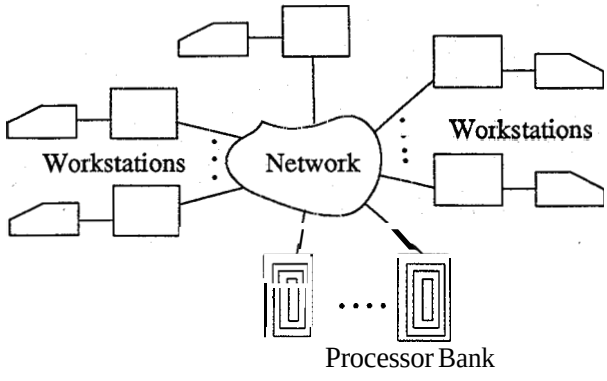


Figure 1.
LOCOX Network

able remote processing cycles The remote cycles are from *private* resources (that are temporarily made available for general use) and *public* resources Private resources are workstations owned by users and under their control If the owner is not using the workstation, the workstation becomes a source of remote cycles, Since we consider a workstation to be owned by a single user, we use the words *user* and *workstation* in the same context Public resources are the processors in a processor bank with the explicit purpose of providing extra cycles This paper considers all the processors within the LOCOX network to use the same instruction set.

A long term scheduler must be efficient and fair An efficient algorithm gives users access to remote cycles without severe overhead and therefore uses most of the available capacity. A long term scheduler is fair if it treats every workstation as an equal contender for available remote cycles. Fair allocation is achieved by trading off the amount of execution time already allocated to a user and the amount of time the user has waited for an allocation This tradeoff is the basis for the ***Remote Cycle Wait Ratio*** evaluation criterion This criterion guards against the domination of computing cycles by ***heavy*** users. The remote cycle wait ratio is the amount of remote execution time a workstation received divided by its wait time The remote execution time of a workstation is defined as the total remote processing time allocated to a workstation. The wait time is the amount of time the workstation had a need for remote cycles but has no such cycles allocated.

Besides the remote cycle wait ratio, we view the fairness of remote cycle allocation from two other related perspectives They are the ***Remote Cycle Percentage*** and the ***Remote Response Ratio*** The remote cycle percentage of a workstation is the percentage of its background job demand that was met by remote cycles The remote response ratio is the expected turnaround time of jobs that finished from a remote location divided by their service demand The turnaround time is the difference between the time a job finishes its execution and its arrival time. The remote cycle wait ratio differs from the remote cycle percentage because the remote cycle wait ratio gives the expected amount of time a workstation has to wait to receive remote cycles, while the remote cycle percentage gives the proportion of cycles consumed remotely in comparison to the total number of cycles consumed by background jobs The remote response ratio is a criterion that considers the individual jobs while the other criteria look at the total allocation of cycles per user. A fair allocation algorithm should result in steady behavior for all three criteria for lightly loaded users that share resources with heavy loaded users regardless of the demand pattern of the latter

We have developed an efficient and fair long term scheduling algorithm, called the ***Up-Down Algorithm,*** that meets these performance objectives The Up-Down algorithm maintains ***steady access*** to remote cycles for light users in spite of a large continuous demand for cycles by heavy users Naive approaches cause light users' quality of service to ***suffer*** when heavier users increase the number of cycles they consume The difference between the Up-Down algorithm and naive algorithms is that the Up-Down algorithm trades off *reward* (remote capacity allocated) and ***penalty*** (waiting time suffered when a remote resource is wanted but denied), while the other algorithms favor heavy users with better access to remote capacity In section 5, we present a detailed analysis of the algorithm We show that under the Up-Down algorithm, light users maintain a steady share of remote resources even when heavy users keep asking for more

The workloads used for evaluating long term scheduling algorithms are important The evaluation is better justified if it is done using workloads derived from real systems We have evaluated our algorithm by using a ***trace of workstation usage*** The trace was obtained by monitoring the activity of a subset of our workstations over a period of five months

Several other papers have discussed distributed computing systems and have addressed forms of scheduling distributed resources These systems include the Locus System [9], the Cambridge Distributed Computing System [10], the Eden System [11], the Charlotte Distributed Operating System [12], Process Server [13], the **NEST** project [14], and the remote execution facility in the V-Kernel [15] Locus is a distributed Unix® operating system with multiple hosts It supports transparent access to a distributed file system with the ability of the user to explicitly schedule a job at the lowest loaded machine The Cambridge Distributed Computing System provides transparent access to distributed resources A central concept to the system is to provide access to a remotely located machine as a personal computer where the user explicitly schedules work for the machine. The Eden System consists of distributed workstations for a high degree of sharing and cooperation among the users Each machine is part of a larger system, and no single user of a workstation has complete control of their workstation The Eden system kernel determines on which workstation of the system a process will execute Foreign processes can be placed on a workstation that resides in a particular user's office even though that user is actively working on the workstation in that office The Charlotte Distributed Operating System runs on the Crystal Multicomputer and supports closely interacting processes cooperating to solve a computationally intensive problem [2] Processes are placed on machines explicitly by users and will stay there until they tenninate or are explicitly migrated Process migration is the movement of processes during their execution among different machines in the system depending on each individual machine's load Papers describing Process Server, the **NEST** project, and the preemptable remote execution facilities of the V-Kernel discuss facilities for the remote execution of programs on idle workstations These papers discuss how to implement the remote execution facilities, but issues of scheduling are not addressed

Except for the Eden System, these papers describe systems that require the users to initiate the placement of processes at machine locations. The Eden System kernel determines where to place processes, but it does not consider the workstations as private resources In the Eden System, foreign processes can be placed at workstations even though the workstation's owner is actively submitting jobs.

Section 2 describes the workload model for our study In section 3 we present mechanisms that have been established to support efficient scheduling of our LOCOX network The system model for our study is presented in section 4 The model allows users to have control of their workstations, but enables others to use workstations that would otherwise be idle We describe in section 5 our design of the Up-Down algorithm for allocating remote capacity and compare

® Unix is a trademark of AT&T Bell Laboratories

its performance and behavior with the Random and Round-Robin algorithms Section 6 presents our conclusions and a description of our on going work on LOCOX network resource management

2. Workload Of Workstations

We have monitored the usage patterns of 13 DEC MicroVAX II workstations running under Berkeley Unix 4 2BSD over a period of five months The stations observed are owned by a variety of users They are 6 workstations owned by faculty, 5 by systems programmers, and 2 by graduate students

We have obtained the profile of *available* and *non-available* periods of workstations so that we can use an actual workload in our evaluation study. An unavailable period, ***NA,*** occurs when a workstation is being used, or was recently used by its owner The station is considered as NA if the average user cpu usage was above a threshold (one-fourth of one percent [16]) within the last 5 minutes The average cpu usage follows the method the Unix Operating system uses for the calculation of user load. This load is a decaying average that includes only the user processes Activities resulting from programs such as time of day clocks or graphical representations of system load do not generate user loads that arise above the threshold An available period, ***AV,*** occurs whenever a workstation's state is not ***NA***

The workstation usage patterns were obtained by having a monitoring program executing on each workstation The monitor on each station executes as a system job and does not affect the user load The monitor looks at the user's load every minute when the workstation is in the ***NA*** state If the user's load is below the threshold for at least 5 minutes, the workstation's state becomes ***AV*** During this time the workstation's monitor will have its "screen saver" enabled. The monitor looks at the user's load every 30 seconds when the workstation is in the ***AV*** state Any user activity, even a single stroke at the keyboard or mouse, will cause the "screen saver" to be disabled and all user windows on the workstation's screen to be redrawn. This activity brings the user load above the threshold, and causes the state to become ***AV.*** If no further activity occurs, approximately seven minutes pass before the station's state changes to ***AV*** This is because it takes the user load average 2-3 minutes to drop below the threshold, and an additional 5 minute waiting time is imposed. The waiting period is imposed so that users who stop working only temporarily are not disturbed by the "screen saver" reappearing as soon as they are ready to type another command The waiting time is adjustable, but it has been observed that the five minute value is a good value to choose without causing an annoyance to users [17] This conservatively decides whether a station should be a target for remote cycles. Stations are idle much more than what appears in the ***AV*** state The user load with the imposed waiting time is used as a means of detecting availability because the station should not be considered a source of remote cycles if an owner is merely doing some work, thinking fox a minute, and then doing some more work Otherwise a station would be a source of remote cycles as soon as the owner stopped momentarily. The workstation's owner would suffer from the effect of swapping in and out of his/her processes, and the starting and stopping activities of the remote processes

An analysis of the traces showed that the monitored workstations were available approximately 70% of the time This means there are a lot of extra cycles to use fox long term scheduling. The average ***AV*** and ***NA*** state lengths were about 100 minutes and 40 minutes respectively. Long ***AV*** intervals are desirable since background jobs placed remotely will have a good chance to stay there for a long time. One might expect that long ***AV*** intervals occur only in the evening hours. We have observed a high percentage of such intervals during working hours The busiest time during the working week was observed to be between 2-3 PM. Even during this time, the average amount of time the workstations are in the ***AV*** state is approximately 50%. A detail analysis of workstation usage patterns is given in [6]

3. Mechanism For Long Term Scheduling

In order to implement a long term scheduliig policy on a LOCOX network, a number of mechanisms are needed. A mechanism for remote placement of jobs? checkpointing, restarting jobs from the checkpoint, and monitoring the activity of the LOCOX network has to be in place in order to carry out any long term scheduling policy. Checkpointing is required since a remotely executing job must be stopped when a user resumes using the workstation on which it is running. This job will either be moved to another location to resume execution, or returned to its origin workstation to wait until a new location becomes available

Within our LOCOX network, we have implemented checkpointing for the remote Unix [18] facility of the Crystal Multicomputer in order to determine the feasibility and the cost of such a mechanism The Crystal Multicomputer is designed to be used as a tool for research in distributed systems It consists of 20 VAX-11/750s connected by a 80 Megabit/sec Proteon ProNet. Crystal has been used for many projects in distributed systems which include distributed databases, algorithms, operating systems, and others [12, 19-21]. The remote Unix facility allows the Crystal Multicomputer to be used as a "cycle server". A cycle server provides computing capacity beyond what the local workstations provide. It extends the idea of Unix forking of a background process so that the new process executes on a Crystal machine. When remote Unix is explicitly invoked, a shadow process on the host machine runs locally as the surrogate of the process running on the remote machine Any Unix system call of a program running on the remote machine causes a trap. A message indicating the type of system call is sent to the shadow process on the host machine This remote Unix facility serves CPU-bound type jobs well Long running simulation programs are an obvious application for it

The checkpointing of a program is the saving of an intermediate state of the program so that its execution can be restarted from this intermediate state. The state of a remote Unix program is the text, data, bss, and the stack segments of the program. Along with the registers being used by the program, any messages sent by the program that have not yet been received have to be saved The text segment contains the executable code, the data segment contains the initialized variables of the program, and the bss segment holds the uninitialized variables. The implementation copies the data, bss, and the stack segments, and the program registers from the remote processor to the origin workstation The text segment is kept in the load module file, and is not copied from the remote processor since we assume the text segment will not be modified. A checkpoint will not be taken whenever there is an outstanding message from the remote end. This can be guaranteed since every message that the remote processor sends expects a response. If at checkpoint time the remote node has not received a response to a message it sent, then the checkpoint will be delayed for a short while until there are no outstanding messages

The implementation allows two different ways to initiate the saving of a checkpoint. The checkpoint can be triggered externally by a signal, or internally by the expiration of a checkpoint timer. The program can be restarted from a checkpoint file by setting a command line parameter to do so A newer version of remote Unix checkpointing has been implemented by Litzkow [16] for both the Crystal Multicomputer and the network of MicroVAX 11 workstations which has an added feature of spooling background jobs. The delay caused by checkpointing on the Crystal Multicomputer has been determined to be about ½ minute for a checkpoint file size of 1 megabyte The capacity consumed by a local workstation in order to checkpoint a remote jobs in a network of workstations was measured to be approximately 5 seconds of CPU time per 1 megabyte of checkpoint file

4. The Simulation Study Model

We need to model our LOCOX network with a scheduler in order to study its behavior and evaluate its performance. Our model for the study of long term scheduling algorithms consists of a network of workstations and a processor bank. Table 2 shows the parameters of the model. The number of workstations is designated by *NumWorkstations* and the processor bank size by *BankSize*. Workstations will either be in the ***AV*** or ***NA*** state. These states are determined by the workstation workload pattern discussed in section 2.

Parameter	Meaning
servemean(i)	Exponentially distributed mean service time at workstation *i*
arrive(i)	Exponentially distributed mean interarrival time of jobs at workstation *i*
NumPermanent(i)	Number of jobs workstation *i* permanently wishes to execute
SchedInterval	Periodic scheduling interval of the coordinator
JobTransferCost	Time it takes to checkpoint a job and move it remotely
SimTime	Length of simulated time
NumWorkstations	Number of workstations simulated
BankSize	Number of processor bank nodes

cessor sharing scheduling with a system of one remote processor to be shared by two workstations. Suppose each workstation's state is in the ***AV*** state 50% of the time. The remote processor is scheduled using processor sharing. The local workstations would only be used as extra remote capacity if their state is ***AV*** and the workstation had no local background jobs to run. Suppose User **I** has two background jobs ready to execute all the time. User II has one background job ready to execute all the time. User II would use its local workstation for its background job 50% the time and the remaining time it would be forced to share the remote processor with User I. User **I** would use its local workstation whenever it is available, and it would request use of the remote processor all the time. User **I** would have no contention for the remote processor's cycles 50% the time and share it the remaining time. The remote cycle wait ratio for User I would be 3 since **75%** of the time the heavy user would receive all the computing capacity of the remote processor, and 25% of the time User **I** would wait without being allocated remote capacity. User II would be allocated ***25%*** of the remote cycles, and would have to wait 25% of the time to receive those cycles without any allocation given. Its remote cycle wait ratio is 1. In this example, we see that processor sharing causes User II to wait 3 times more for each cycle allocated than User **I.** User II is also allocated less remote cycles in proportion to what is given to User **I**. User II would like to receive 50% of the remote processor's cycles, but it only receives 25%. User **I** would like to receive 100% of the remote processor's cycles, but receives 75%. User II receives **50%** of the remote processor's cycles it requested while User **I** receives ***75%*** of its request.

We consider past history when allocating remote cycles to provide fair access for users with different loading patterns. To be fair, the algorithm considers past behavior by trading off the amount of execution time allocated to a user and the amount of time the user has waited for an allocation. Since we assume that all workstations are entitled to equal rights, heavy users should not be allowed to dominate the remote cycles at the expense of light users. Algorithms that do not consider past behavior do not protect light users. Light users with steady demand will have increasing remote cycle wait ratios as heavy users increase their demand. Heavy users will have greater throughput with a naive algorithm when compaxing it to the Up-Down algorithm, but the overall throughput of the system will be as good when using the Up-Down algorithm. A description of the Up-Down algorithm follows.

5.1. The Up-Down Algorithm

In this algorithm, the scheduling coordinator bases its decisions on an allocation table called the schedule index table, *SI*. An entry ***SI[i]*** is the schedule index for workstation *i*. The values of the *SI* table are used to decide which workstation is next to be allocated remote capacity. Workstations with smaller *SI* entries are given priority over workstations with larger *SI* entries. Any time two or more workstations with the same schedule index contend for cycles, the workstation allocated the cycles is randomly chosen. Initially, each entry of the table is set to zero. The values of the *SI* table are updated on a periodic basis and whenever new capacity becomes available to the system. This occurs when

- a scheduling interval *SchedInterval* has expired
- a station's state goes from ***NA*** to ***AV***
- a remote job completes and leaves the system

The scheduling interval should not occur too often due to the overhead of placing and preempting jobs on a remote processor, and it should occur often enough to give stations with low *SI* entries access to a node without too much waiting. The workstation that wants a node but does not have one will preempt a workstation that has one if its ***SI[i]*** is smaller than the workstation with the node. No workstations with smaller *SI* entries that have already acquired remote cycles for the next interval can preempt other workstations with larger *SI* entries. Table 3 outlines the allocation algorithm for

remote cycles.

We want the algorithm to adjust to changes in background load patterns For each scheduling interval, each station is given a reward or penalty depending whether the station has been granted remote cycles or has waited but has not received any. Light users that increase their loads will have their priority decreased so that they will be considered heavy users, and heavy users that decrease their loads will be considered light users However, the algorithm should be sensitive only to significant changes Table 4 summarizes the update policy for the *SI* table. Four schedule index functions (*f*, *g*, *h*, and *l*) are employed to adjust for changes in load patterns The function *f* is the assessed reward per remote processor received by workstations (amount *SI* entry is increased) for using remote cycles This function would cause a light user that increases its load significantly to have its priority decreased until it is viewed no differently from a heavy user The function *g* is the accessed penalty received by workstations (amount *SI* entry is decreased) fox waiting for cycles. The *SI* entry is decreased by the amount of the function *g* if a station

```
Allocation Of Remote Processing Nodes For Background Jobs

    at each scheduling interval (
        S1 ← [bag of workstations that have nodes allocated ]
        S2 ← [set of workstation that want nodes allocated 3
    }
    for the number of nodes free(
        if S2 <> EMPTY (
            s = workstation in S2 with smallest SI entry
            allocate a node to s
            s2 ← s2 - [s]
        }
        else break;
    }
    while S2 <> EMPTY (
        s = workstation in S2 with smallest SI entry
        t = workstation in S1 with largest SI entry
        if SI[s] < SI[t] {
            preempt a node from t
            allocated a node to s
            s2 ← s2 - s
            S1 <- S1 - t
        }
        else break;
    }
```

Table 3

wants a remote processor but was denied one The functions *h* and *1* stablize the priority of the workstations when they do not want cycles Any station that does not want a remote processor and has a positive *SI* entry will have its schedule index decreased by the function *h* each time the *SI[i]* table is updated until the entry reaches zero. This means that a heavy user that significantly decreases its load will have its index lowered until it is viewed no differently from a light user. Any station that does not want a remote processor and has a negative *SI* entry will have its entry increased until it reaches zero according to function *1*. Once a station's *SI* entry reaches zero, it will stay there until it asks for a processor

Figure 2 illustrates how the **Up-Down** algorithm modifies the *SI* table The figure represents the value of the index of station *i*, *SI[i]*, over time The figure shows how the index for a station changes when a station waits to receive remote cycles, when remote cycles are allocated, after a job has completed, and when there is no need for remote cycles Depending on the accessed reward and penalty received by each station, the index for each station goes up and down. This gives the name to the Up-Down algorithm. In figure 2, the *SI[i]* is initially zero. When a job arrives and there is no allocation given, the index decreases according to *g(SI[i])* After an allocation is made, the index rises according to *f(SI[i])*. If two allocations are given, the index rises twice as fast, namely, *2*f (SI[i])* The completion of one of the jobs causes the index to rise according to *f(SI[i])* When the second job completes and there are no allocations or jobs waiting, the index decreases to zero by the function *h(SI[i])*.

5.2. Algorithms Used For Comparisons

For comparison with the Up-Down algorithm, we have selected two algorithms that do not use past behavior when deciding how to allocate remote capacity The two selected are the Random and Round-Robin algorithms, which are non-preemptive algorithms We

```
Modification Of The Schedule Index Table (SI)
  During Each Scheduling Interval
    For Each workstation (i)

    fox each i (
        if i wants a remote processing node (
            if i has a node then SI[i] :=
                SI[i] + NumProcessors*f(SI[i]);
            else SI[i] = SI[i] - g(SI[i]);
        }
        elsif SI[i] > 0 then SI[i] = SI[i] - h(SI[i]);
        elsif SI[i] < 0 then SI[i] = SI[i] + l(SI[i]);
    }
```

Table 4

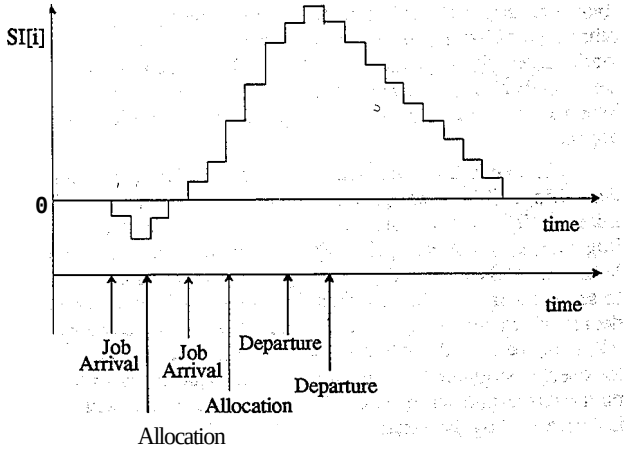


Figure 2: Modification Of Station *i* Schedule Index

compare the Up-Down algorithm with the Random and Round-Robin algorithms to find out whether less sophisticated algorithms are adequate to fairly allocate remote cycles and to provide steady performance results. The Up-Down algorithm, Random, and Round-Robin algorithms are simple to implement and are efficient in providing remote cycles. However, they differ in the way they treat users with different workloads.

The Random algorithm's name tells a lot about it. All of its decisions are made without reference to past decisions. Whenever there is contention for cycles and remote cycles are available, a requester is randomly picked to be allocated. Whenever there is no contention, a requester is given all the available remote processors that it wants. Whenever a requester is allocated a remote processor for a background job, the requester will keep the allocation until either the job terminates, or the remote processor becomes busy by its local user

The Round-Robin scheduling algorithm requires the scheduling coordinator to maintain a cyclic order whenever it allocates remote cycles to requesters Each workstation is given a chance in a particular order to receive remote cycles If the station does not need them when its chance occurs, it will have to wait until everyone else gets one chance Once allocated a remote processor, a workstation keeps the allocation until either the job terminates or the remote processor becomes busy with some local activity When a remote job completes, the available processor returns to the pool of free processors If remote processors are available but no workstation wants them, the scheduling coordinator periodically checks for new background jobs, with interval *SchedInterval*

5.3. Simulation Study Results

In order to evaluate the performance of the Up-Down algorithm and the two naive algorithms, we conducted a simulation study using the DeNet [23] simulation language DeNet is a discrete event simulation language built on top of the general purpose programming language Modula-2 [24]. The simulation study was done with the simulation parameter settings shown in Table 5 We have assigned *NumWorkstations* with a value of 13 which is the number of workstations we monitored We ran two sets of experiments One with a *BankSize* of zero which represents an environment with no dedicated processors The second has a *BankSize* of 5 These two experiments enabled us to examine the impact of augmenting a system of private workstations with additional cycles

To show how workstations with a light background load perform in the face of different demand levels from heavy users, we made all but two of the workstations in the experiments to have a light user, labeled by *LightStations* One of the two remaining stations is designated as *Mediumstation* and the other is designated as *HeavyStation*. The *MediumStation* has two permanent jobs ready for execution (*NumPermanent(MediumStation)* = 2) For the *HeavyStation*, we varied *NumPermanent(HeavyStation)* from 2 to 13 jobs The background jobs have a mean service demand of 2 hours (*servemean(i)* = 2, for all *i*)

Table 6 summarizes the schedule index functions selected for the Up-Down algorithm We have observed that at high utilizations, the *SI* of a station might become very large and would slowly reach zero when the station changes from a heavy to a light load We chose to focus on the function *g* to control the time it takes for station's index to reach zero Functions *f*, *h*, and *l* will always return one as their value We ran the simulation for about two years of simulated time to observe steady state behavior

Our results show that the Up-Down algorithm maintains a fair allocation of resources to all types of users Whereas under the Random and Round-Robin algorithms, heavy users receive favorable treatment We compared the performance of the three algorithms using the criteria defined in the introduction which are: (1) the remote cycle wait ratio, (2) the remote cycle percentage of light users, and (3) the remote response time The remote cycle wait ratio of *Lightstations* is computed by averaging the local remote cycle wait ratios of the individual *Lightstations* Likewise, the remote response time of *LightStations* is computed by averaging the local remote response times of the individual *LightStations* On the basis of these criteria we will show that the quality of service *LightStations* and *MediumStation* enjoy in face of increasing loads of the *HeavyStation* remains steady when we use the Up-Down algorithm When we use the Random and Round-Robin algorithms, the quality of service for *Lightstations* and *MediumStation* suffers as *HeavyStation* increases its load

Parameter	Value
servemean(i)	2 hours for all stations *i*
arrive(LightStation)	2000 minutes
arrive(MediumStation)	no such arrivals
arrive(HeavyStation)	no such arrivals
NumPermanent(LightStation)	0
Nudermanent(MediumStation)	2
NumPermanent(HeavyStation)	2-13
SchedInterval	10 minutes
JobTransferCost	1 minute
SimTime	2 years
NumWorkstations	13
BankSize	0 for first experiment, 5 for second experiment

Table 5 Simulation Parameter Settings

Up-Down Schedule Index Functions
$f(SI[i])=1$, for all i.
$h(SI[i])=1$, for all i.
$l(SI[i])=1$, for all i.
$g(SI[i]) = \begin{cases} 3, & SI[i] \geq 6 \\ 2, & 3 \leq SI[i] < 6 \\ 1, & SI[i] < 3 \end{cases}$

Table 6

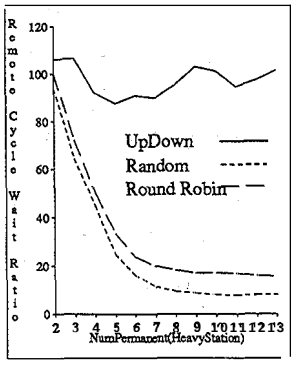


Figure 3
Remote Cycle Wait Ratio
(Lightstations)
No Processor Bank

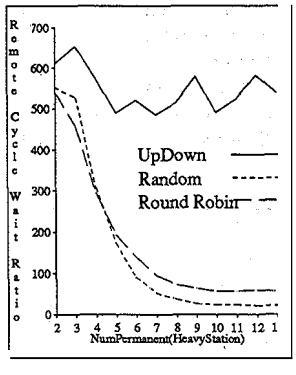


Figure 4
Remote Cycles Wait Ratio
(MediumStation)
No Processor Bank

5.3.1. Experiment 1: System Of Workstations Without Processor Bank

Heavy users want to separate their work into many jobs in order to take advantage of the huge amount of remote cycles available This activity is encouraged so that the system can be utilized as much as possible However, the heavy users can inhibit the light users' access to remote resources This is what occurs with the Random and Round-Robin algorithms The Up-Down algorithm enables light users to maintain fair access to remote resources while still allowing heavy users use an abundant supply of remote cycles

We used the outlined performance criteria to compare the Up-Down algorithm with the Random and Round-Robin algorithms We see in figures 3 through 7 that the Up-Down algorithm maintains an improved and stead.) quality of service for non-heavy users The quality does not depend on the usage pattern of heavy users Figure 3 presents the remote cycle wait ratio of *LightStations* as a function