GLOBAL STATE IDENTIFICATION FOR LOAD BALANCING IN A COMPUTER SYSTEM WITH MULTIPLE CONTENTION BUSSES

*Jie-Yong Juang and Benjamin W. Wah**

ABSTRACT

Load balancing is a job scheduling scheme that engages the communication facilities to support remote job execution in a user-transparent manner. By transferring jobs from heavily loaded processors to lightly loaded processors, the processor utilization can be increased and the average response time can be reduced. However, collecting the necessary status information to make the load balancing decisions usually incurs a large overhead, which may impede regular message traffic and degrade system performance. Moreover, network delays may outdate the status information collected and result in improper job migrations. In this paper, we study an efficient load balancing scheme for a computer network with multiple contention busses. The scheme minimizes the transfer of status information by reducing the job-migration decision problem to the ordered-selection problem. A heuristic multi-window protocol that utilizes the collision-detection capability of contention busses is analyzed. The proposed protocol does not require explicit message transfers and can identify the t smallest variates out of N distributed random variates in an average of $(0.8 \log_2 t + 0.2 \log_2 N + 1.2)$ contention steps.

1. INTRODUCTION

Advances in communication technology have resulted in the development of new media for local computer networks. Although the potential bandwidth of media such as optical-fiber links and coaxial cables is very large, the current bandwidth used on most of these links is less than 50 Mbps. The architectures of current communication subsystems are built around networks of low bandwidth, such as the notable Ethernet and token-ring networks [9, 14, 15] and is not effective when a higher bandwidth is allowed in the future. An example is the contention-resolution mechanism of Ethernet. When the bus has a much higher bandwidth than the currently popular 10-Mbps bus, the time taken for data transmission will be shortened, but the time taken for contention resolution remains unchanged due to the speed-of-light limitation. In this case, the network has to be broken down into multiple channels of lower bandwidth to have a better utilization of each channel for data transmission.

In this paper, we study load balancing schemes in local computer systems in which processors are connected by a set of contention busses similar to the Ethernet. The multiple busses can either be multiplexed in a single high-bandwidth bus or exist as separate Ethernet links. Each processor has an independent job stream with independent jobs. The job streams may have different arrival rates and work demand, hence some of the processors may be heavily loaded while others may be idle. Jobs waiting in heavily loaded processors, called *backlogged jobs*, can be executed by other lightly loaded processors. This has the effects of reducing the average response time and increasing the average processor utilization. A balanced workload can be achieved by either distributing users uniformly on the system or allowing them to switch from one processor to another. The remote execution of jobs and virtual terminal protocols [4, 17] are common in many local computer networks. This form of load balancing balances workload on a per-session basis. However, it cannot effectively reduce the number of backlogged jobs since a session is a mix of user think time and job-execution time. On the other hand, *user-transparent load balancing schemes* have been proposed to balance the workload on a per-job basis. They determine whether a job is to be either executed locally or migrated to another processor for execution. If a job is to be migrated, it is passed to the message-transfer subsystem to be sent. The message-transfer subsystem also allows the migrated job to access data from and return results to its originating site.

Three issues can be identified in making job-migration decisions in a user-transparent load balancing scheme. They are the times to initiate job migrations, the jobs to be migrated, and the places to send the migrated jobs. Previous studies have provided various answers to these questions. In ECN [4] and LOCUS [22], a migration decision is made upon the arrival of a new job, and the location to migrate the job is determined by polling other processors. In a second approach [1], a job is transferred to a neighbor whenever it is possible, and is rippled to a proper site later. In the threshold approach, a job migration is carried out when the workload of the local processor exceeds a threshold, and a load table is maintained in each processor to determine where the job should go [12]. In another approach using thresholds, a processor tries to bid for a job whenever it becomes idle [8]. No previous work addressed the issue on the jobs to be transferred.

A closer look at the problem reveals that load balancing enhances resource sharing by removing backlogged jobs and transferring them to more promising sites. Although a load balancing scheme is hard to analyze [2, 19], its capability to reduce backlogged jobs can be compared in terms of different queuing models. A system without load balancing is equivalent to a queuing system with independent servers. The probability of a job being backlogged may be high in such a system [8]. Load balancing schemes that determine job migrations upon every new arrival are equivalent to a multi-server queuing system with a job routing mechanism. *Probabilistic* and *deterministic* strategies have been studied [2]. In probabilistic routing, a new arrival is dispatched according to a set of branching probabilities that are optimized with respect to statistics gathered about the processing speeds and the job arrival history [11]. Only the average workload can be balanced, but instantaneous changes in workload cannot be accommodated. In contrast, a deterministic strategy is state-dependent and routes a job according to the current workload status. It can effectively smooth out instantaneous variations in workloads, and hence can further reduce possible backlogged jobs. Theoretical studies have shown that the performance of systems with a state-dependent routing strategy is nearly as good as that of a single fast computer.

Load balancing by job routing is suitable in multiprocessors with negligible routing delay and readily available workload information. In a local computer network, processors are physically separated, and routing delays may have adverse effects on load

* This research was supported by National Science Foundation Grant DMC 85-19649.

J. Y. Juang is with the Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL 60201.

B. W. Wah is with the Department of Electrical Engineering and Computer Engineering and the Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL 61801.

balancing. State-dependent routing is difficult to implement in such an environment because global status information is hard to maintain. Studies on using thresholds have found that jobs do not have to be migrated immediately when they arrive, but can be queued locally until load information becomes available. However, determining the appropriate thresholds is critical and must consider the overall system load and the time varying traffic levels of the communication network. The thresholds must be adjusted dynamically, which requires updated status information and induces further traffic in the system. Fixed thresholds have been used in most theoretical studies [8, 12].

A load balancing scheme that supports migration by dynamic thresholds in multiple-contention-bus networks is presented in this paper. In Section Two, the problems of global job bidding and individual job-migration decisions are formulated into an optimization problem. We show that the problem of collecting the necessary status information for a system with multiple busses is equivalent to performing ordered selections. A multi-window protocol to perform ordered selections without explicit message transfers and the corresponding hardware implementations are discussed in Section Three. Section Four presents the correctness proof and evaluates the performance of the protocol, and Five draws conclusions.

2. LOAD BALANCING IN A COMPUTER NETWORK WITH MULTIPLE BUSSES

In this section, we describe a load balancing scheme for a computer network with t ($t \geq 1$) contention busses. We first discuss the general problem of load balancing. Then we show that collecting the necessary status information for load balancing in a system with multiple busses can be reduced to ordered selections. Lastly, we describe a distributed interval-resolution scheme to support ordered selections.

2.1. Load Balancing in Computer Networks

Maintaining updated status information is essential in load balancing. Useful status information for load balancing include processor workloads, network status, and the characteristic of the arrival processes [2]. There is a trade-off between the overhead spent on maintaining updated status information in the processors and the increase in response time caused by using outdated status information. Moreover, processors should also coordinate in making job-migration decisions to avoid the problem of transferring jobs from all heavily-loaded processors to a single lightly-loaded processor. In this section, we describe a coordination mechanism, assuming that accurate status information is available. The method to obtain this information will be discussed later.

A processor status can be characterized by two random variables: its current workload (i.e., the unfinished work) and the response time of a job when migrated to this processor (called *virtual load*). Let **T** be a set of observations, each of which represents the current workload of a processor, and **R** be a set of virtual loads, one for each processor. A global job bidding decision can be made by defining a (possibly many-to-one) mapping Ω from **R** to **T**. Note that more than one job may be mapped to a single processor. For a given mapping, it is necessary to evaluate the associated cost. A utility function b of migrating Job i can be defined as a function of r_i, the job response time after migration, $y_i = \Omega(r_i)$, the load of the remote processor, and θ_i, the cost of migration. An optimal mapping can be obtained by maximizing the total utility over all possible mappings.

$$S(\mathbf{R},\mathbf{T}) = \max_{\Omega \in \mathbf{R} \times \mathbf{T}} \sum_{i=1}^{|\mathbf{R}|} b(r_i, y_i, \theta_i) \qquad (1)$$

where $|\mathbf{R}|$ is the cardinality of **R**, and $S(\mathbf{R},\mathbf{T})$ is the utility of the best mapping.

It is hard to obtain a general solution to the above optimization problem due to difficulties in estimating the costs of migrations and predicting the workloads of remote processors after the migrations. Multiple jobs may be mapped to the same processor, and a more complicated utility function may be needed to account for this effect. A possible way to solve this problem is to generalize the status of a processor to a set of virtual loads. The j'th virtual load of Processor i is the predicted workload after Processor i has received j migrated jobs. The resulting formulation using this generalization is very complicated because there may be network interference when more than one job are sent to the same destination. Further, the costs of sending jobs to the same destination may not be a linear function. The mapping problem can be simplified in some special cases. In the next section, we discuss a network with multiple busses in which there is no interference of sending jobs across the multiple busses, and the costs of sending jobs are constant, independent of the source and destination.

2.2. Migration Decision in a Computer Network with Multiple Busses

The cost to migrate a job between processors connected by a bus is independent of the source and destination processors, but depends on the current network traffic (in terms of communication delays) and the additional traffic incurred due to job migrations. It is not easy to characterize either one of them in general because they involve complicated interactions among the migrating jobs, the workload of the processors, and the interference on regular message traffic. In this paper, it is assumed that the preference of a job to be migrated is characterized by a *bidding parameter*.

To simplify the job-migration decisions while retaining good performance, the following assumptions are adopted. First, load balancing is only carried out when there is no regular message to transmit. As a result, the impact of job migrations on regular message transfers can be neglected. This policy can be implemented by assigning a higher priority to packets containing regular messages than that used for packets in load balancing. It can be supported by CSMA/CD protocols that facilitate transmissions with priority [18, 3, 13, 10, 5, 21]. Second, when a job is migrated, the necessary data and control information are sent to the remote processor at the same time. If this is not done, then the migration-decision problem becomes a task-allocation problem involving piecemeal communications in a dynamic environment [16], which is outside the scope of our study on supporting load balancing by low-level network protocols. Third, all jobs are stochastically identical, hence any job in the queue can be migrated. Fourth, the number of busses used to send or receive data simultaneously is a site-dependent parameter. It is assumed that special interface circuits are built to process status information from all busses simultaneously (Section 3.2) and that the interference of receiving jobs from multiple busses concurrently is minimal. Fifth, each bus is used to send one job at a time. A migrated job cannot start until all the necessary packets have been received by the remote processor. Multiplexing packets of multiple migrated jobs simultaneously on a single bus delays the completion of all these jobs, hence is not effective. Sixth, the busses are reliable and information broadcast can be received by all processors simultaneously and correctly. Seventh, all processors in the network participate to identify the sources and destinations of load balancing, hence the number of virtual loads and bidding parameters in the network is constant. Lastly, all bidding parameters and virtual loads are assumed to be uniformly distributed between L and U. Otherwise, statistical distributions collected from information received are used to adjust the random variates according to

$$x_j = F_i(x_j') \cdot (U - L) + L \qquad (2)$$

where x_j' is the j'th local variate of Processor i, and F_i is the statistical distribution of the random variates at processor i. F_i's are identical among the processors as they receive identical information from the busses.

With the above assumptions, the job-migration cost can be reduced to the delays of transferring the job to and returning the result from the remote site. The bidding parameter can be measured in terms of the expected improvement in response time, which is a function of the local response times, the volume of data involved in the migration, and the average packet delay. Due to the assumption on minimal interference of migrating multiple jobs to the same site using multiple busses, the total utility can be truly represented as a

summation of the utilities of individual migrated jobs.

Let the utility of migrating Job i in this class of load balancing schemes be $\beta(x_i, \Omega(x_i))$, where x_i is the bidding parameter of Job i, and Ω is a mapping. The search for the best mapping can be formulated as

$$S(R,T) = \max_{\Omega \in R \times L} \sum_{i=1}^{|R|} \beta(x_i, \Omega(x_i)) \quad (3)$$

With t busses, a processor can receive or distribute at most t jobs simultaneously and can make at most t migration requests. Let job-load pairs be ranked according to their utilities of migration. Eq. (3) can be reduced to the following form.

$$S(R,T) = \max_{\Omega \in R \times L} \sum_{i=1}^{t} \beta(x_i, \Omega(x_i)) \quad (4)$$

$$\beta(x_i, \Omega(x_i)) \leq \beta(x_j, \Omega(x_j)) \text{ for } i < j$$

Based on this formulation, the following theorem can be established.

Theorem 2.1: To maximize the utility of load balancing (Eq. (4)), t requests of the largest bidding parameters should be mapped to the t smallest virtual loads if β is a non-decreasing function of the job bidding parameter and a non-increasing function of the virtual load.

Proof: The theorem is proved by contradiction. Let $x_1, \ldots, x_t$ be the bidding parameters selected by Ω such that the corresponding utilities are among the smallest ones. Assume that the mapping Ω is optimal, but selects at least one processor whose bidding parameter x_p, $1 \leq p \leq t$, is not among the t largest ones. Therefore, there is a bidding parameter x_q, $q>t$, which is among the t largest ones (accordingly, $x_q > x_p$) but $\beta(x_q, \Omega(x_q))$ is not. Since β is an increasing function of the x_i's, the following inequality holds.

$$\beta(x_q, \Omega(x_p)) \geq \beta(x_p, \Omega(x_p)) \qquad x_q > x_p, \ 1 \leq p \leq t < q \quad (5)$$

Let Ω_{new} be another mapping obtained by exchanging x_p with x_q. Then,

$$\begin{aligned} \sum_{j=1}^{t} \beta(x_j, \Omega_{new}(x_j)) &= \sum_{j=1, j \neq p}^{t} \beta(x_j, \Omega_{new}(x_j)) + \beta(x_q, \Omega(x_p)) \quad (6) \\ &\geq \sum_{j=1, j \neq p}^{t} \beta(x_j, \Omega_{new}(x_j)) + \beta(x_p, \Omega(x_p)) \\ &= \sum_{j=1}^{t} \beta(x_j, \Omega(x_j)) \end{aligned}$$

The above inequality indicates that the mapping Ω_{new} has a higher utility than Ω, hence contradicting the assumption that Ω is optimal. To obtain the optimal mapping, jobs with the largest bidding parameters must be chosen. With a similar argument, processors with the smallest virtual loads should be selected. □

According to the above theorem, when load balancing is to be carried out on M processors using t busses, the t smallest virtual loads out of a maximum of $N = M \cdot t$ virtual loads and the t largest bidding parameters out of a maximum of $N = M \cdot t$ bidding parameters are to be selected. *N is constant* in load balancing because all processors will participate in identifying the sources and destinations. Identifying the smallest (or the largest) numbers among a set of random variates is the classical ordered-selection problem. Since all the numbers concerned are distributed physically, it may be necessary to collect them to a central site before sorting them. As this information is costly to collect and may be outdated by the time the selection is done, it calls for an efficient distributed ordered-selection scheme.

2.3. Distributed Ordered Selection on Multiple Contention Busses

An interval-resolution scheme for priority resolution is described here. It is a recursive scheme that partitions and tests the domain of random variates. An interval is resolved if it is empty or contains exactly one number (i.e., a *success*). An unresolved interval is partitioned recursively until it is resolved. To test whether an interval is resolved or not, a resolution scheme with binary questions and ternary responses is used. A processor is asked whether it generates a number in the interval [a,b) and will answer "yes" or "no" without further description. The same question is directed to all processors, and the aggregated response is of the ternary type, i.e., there is none, one, or more than one processor that responded positively. Such a question-answering session is isomorphic to the collision detection of a CSMA/CD network. In such a network, a processor is either transmitting or not transmitting during a contention slot. A transmission is equivalent to answering "yes," while no transmission is equivalent to answering "no." The capability of a collision-detection mechanism to detect whether there is none, one, or more than one processor transmitting is equivalent to obtaining the ternary response from the processors.

The above analogy suggests that interval resolution can be done by contentions on a bus. In the proposed algorithm, a subinterval to be resolved, called a *transmission window*, is assigned to a bus. A processor contends to transmit its random number on the bus if its random number falls in this subinterval. By interpreting the outcome of collision detection, a ternary status of the subinterval can be determined. As the interval-resolution process proceeds, the domain of random variates is partitioned into subintervals, each of which is in one of the four possible states: empty, success, collided, or unsearched. Assuming that all the random variates lie in the interval [L,U), the order of a random variate x_j can be determined if the interval between the L and x_j has been partitioned into subintervals, each of which has been resolved to be either success or empty. Since verifying the status of a subinterval is independent of verifying other subintervals, multiple subintervals can be resolved sequentially or in parallel depending on the number of contention busses available.

The example in Figure 1 illustrates an interval-resolution process. In this example, the order of ten random variates generated by six processors is to be determined, and two contention busses are available. The windows used in each step are labeled w_1 and w_2, and the status of each subinterval after a contention step is marked in the figure. After three contention steps, a number generated by Processor 3 can be determined to be the minimum, and the second minimum can also be identified. To determine the order of others, further contentions are necessary.

Processors involved in ordered selections have to know the windows used in each contention step. This may be done by assigning a processor to generate the appropriate windows and broadcast them to others. This approach is inefficient as broadcasting incurs a significant overhead. Alternatively, if all processors are evaluating an identical algorithm with identical inputs, then the windows can be synchronized without any message transfer. The identical inputs needed are information readily available from the collision-detection mechanism.

2.4. Organization of the Load Balancing Mechanism

Based on the distributed ordered-selection algorithm described above, an architecture that supports load balancing is shown in Figure 2. In this architecture, the job scheduler requests the migration-decision mechanism to compute a bidding parameter for each job concerned. A collection of bidding parameters and the processor's virtual loads are sent to the ordered-selection protocol. Given that there are t busses available, two ordered-selection operations are then invoked in sequence. One identifies the t smallest virtual loads in the network, and the other identifies the t largest bidding parameters. This information will be passed to the migration-decision mechanism at which a mapping is generated. The local job scheduler will be notified if any migration request is accepted. The accepted job is then scheduled to migrate through the message-transfer subsystem. The message-transfer subsystem will also be notified to prepare buffers to receive migrating jobs from other processors.

The average packet delay in the network may be obtained by piggybacking timing information on packets or by monitoring bus activities. However, an accurate estimation of the delays is very complex and is a subject for future studies. In the remaining part of this paper, we focus on the design of an ordered-selection protocol.

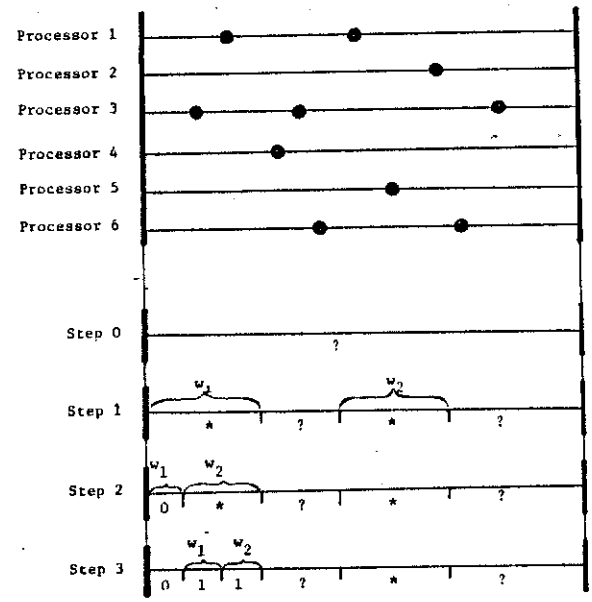

Figure 1. An example illustrating an interval-resolution procedure for ordered selection (windows used in each step are labeled by w_1 and w_2).

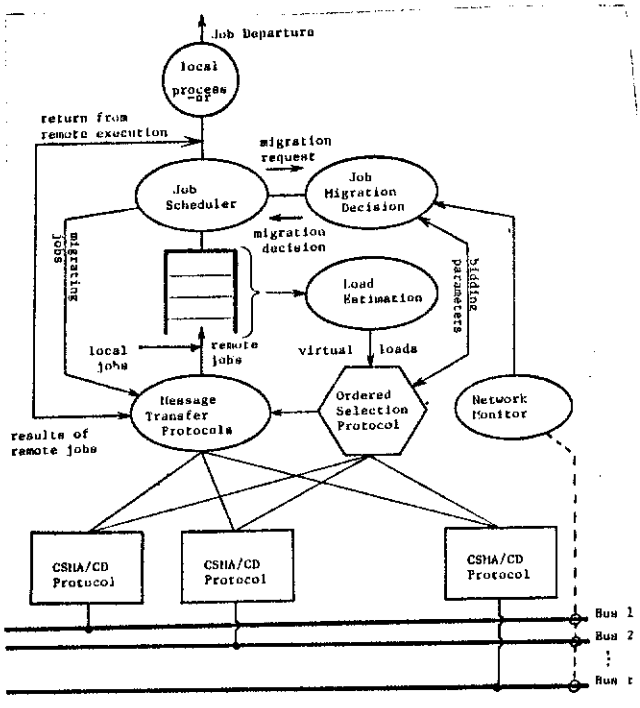

Figure 2. The architecture of a processor to support load balancing in a computer network with multiple contention busses.

3. MULTI-WINDOW PROTOCOL FOR ORDERED SELECTIONS

A key issue in an interval-resolution scheme for ordered selection is to determine a proper transmission window for each bus. We have developed a window-control scheme for the degenerate case of a single contention bus [5, 21]. In this section, we describe a *multi-window control scheme* for parallel interval resolutions on multiple busses.

In the multi-window control scheme, window generations in different processors are synchronized by collision detections. A *contention step* consists of the generation of transmission windows, the contention for interval resolution, and the acquisition of interval status by collision detection. Transmission and collision detection can be done in one *contention slot*, which is a fixed system parameter of a CSMA/CD network. On the other hand, the generation of transmission windows involves local processing and must be optimized.

3.1. Optimal Multi-Window Control

The set of windows used in a contention step is abbreviated as the *window vector* in the sequel. A window vector is chosen from unresolved subintervals, including collided and unsearched ones. For convenience, subintervals are represented by vector $\vec{V}$ in which each element is a subinterval represented by a triplet consisting of the lower and upper bounds and the status of the subinterval (empty, success, collided, or unsearched). Based on such a representation, the optimization of multi-window control may be formulated in dynamic programming.

Consider the case in which the t smallest numbers are to be selected from N distributed random numbers, and the current unresolved subintervals are represented by $\vec{V}$. Given $\vec{V}$, a contention step using window vector $\vec{W}$ will result in another set of unresolved subintervals $\vec{U}$. Denote the expected number of contention steps to complete an ordered selection by $C^t_{N,\vec{V}}$. The dynamic programming formulation for window generation may be expressed recursively as follows.

$$C^t_{N,\vec{V}} = \min_{\vec{W}} \left\{ 1 + \sum_{i=1}^{t} \sum_{\vec{U}} p_{N,i,t,\vec{V},\vec{U}}(\vec{W}) \cdot C^{t-i}_{N-i,\vec{U}} \right\} \qquad (7)$$

where $p_{N,i,t,\vec{V},\vec{U}}(\vec{W})$ is the probability to successfully isolate the i smallest numbers with windows $\vec{W}$, and the set of unresolved subintervals changes from $\vec{V}$ to $\vec{U}$ in this process. The optimal window vector is the one that minimizes Eq. (1).

To evaluate Eq. (1), all $p_{N,i,t,\vec{V},\vec{U}}(\vec{W})$'s must be known. Since as many as t busses can be assigned to resolve a subinterval, there are K^t possible ways of assigning t busses to resolve K subintervals. For each assignment, there exists a large number of combinations of window sizes to be determined. This leads to an exponential number of combinations, each with a different $p_{N,i,t,\vec{V},\vec{U}}(\vec{W})$. Hence Eq. (1) is too complex to be evaluated exhaustively, and suboptimal solutions to the window-generation problem will be presented in the next section.

3.2. A Multi-Window Protocol with Heuristic Window Control

(i) *Heuristic Window Control*

To solve the window-generation problem heuristically, the following observations were made. First, random variates of the smallest values are to be selected, hence unresolved subintervals at the lower end (those containing smaller values) should be searched first. Second, given an interval of size u with N uniformly distributed random variates, the proper windows to maximize the probability of having exactly one random variate in each window can be shown to be u/N. It also can be shown that the average number of random variates falling in a subinterval of size u/N is less than 2.4. Hence if two busses are available and collision is detected in a subinterval of size u/N, then the subinterval should be divided into two equal partitions and searched individually.

Initially, the entire domain [L,U) of size u = U–L to be searched is taken as an unsearched interval. Given that there are t busses, this interval is partitioned into (t+1) subintervals

[L, L+u/N), [L+u/N, L+2u/N), ..., [L+(t-1)u/N, L+tu/N), [L+tu/N, U), where N is the number of random variates in [L,U). The size of the first t subintervals is u/N each. Unresolved subintervals at the lower end of the domain (those having smaller values) are searched first. Note that there is only one unsearched subinterval [L+tu/N, U) at the upper end of the domain. A processor is allowed to contend for a bus if it has a random variate falling in the subinterval assigned to that bus. After a contention step, a collided subinterval is partitioned into two equal halves, each of which is considered as an unresolved subinterval. The contention steps and partitioning are repeated until either the ordered selection is done, or all subintervals except [L+tu/N, U) have been resolved. In the latter case, the search is extended into the unsearched subinterval [L+tu/N, U) and the search procedure is repeated. A variable Φ is kept in each processor to indicate the lower bound of the unsearched subinterval. When the search is extended into the unsearched subinterval, Q, the number of random variates successfully isolated in [L,Φ), is used to estimate the upper bound of the number of random variates in [Φ,U). The unsearched subintervals is divided into smaller subintervals of size (U-Φ)/Q each.

It will be shown in Section Four that the window-control algorithm based on the above steps is correct and performs satisfactorily.

(ii) *Sequencing and Termination*

If the status of all subintervals are known, then the order of the selected numbers and the termination condition of the selection process can be determined. As unresolved subintervals are partitioned iteratively, and the number of subintervals grows linearly as the resolution proceeds, it is hard for a processor to keep track of the status of all subintervals with a limited amount of memory space.

To reduce the memory requirement, a constant-memory scheme is proposed here. For Processor i, only the order of the locally generated random variates $x_{i,1}$, ..., $x_{i,t}$ has to be known. For $x_{i,j}$, it is necessary to know the number of subintervals between L and $x_{i,j}$ that have been identified to contain exactly one variate each. This gives a lower bound on the order of $x_{i,j}$ in the N random variates and is kept in $x_{i,j}^{order}$. An estimate on the number of variates in collided subintervals between L and $x_{i,j}$ may be useful, but not essential, to improve $x_{i,j}^{order}$.

$x_{i,j}^{order}$ is initialized to one and updated as follows. In each iteration, t subintervals as specified by $\vec{W}$ are assigned to the t available busses for contention resolution such that $[\ell_j, u_j)$ is assigned to Bus j and $\ell_j \geq u_{j-1}$ and $u_j \leq \ell_{j+1}$, $2 \leq j \leq t-1$. The outcomes of collision detection are represented by two t-tuples, $\vec{S} = (s_1, ..., s_t)$ and $\vec{D} = (d_1, ..., d_t)$, where

$$s_j = \begin{cases} 1 & \text{if contention in the j'th bus on } [\ell_j, u_j) \text{ succeeds,} \\ 0 & \text{otherwise.} \end{cases} \quad (8a)$$

$$d_j = \begin{cases} 1 & \text{if contention in the j'th bus on } [\ell_j, u_j) \text{ collides,} \\ 0 & \text{otherwise.} \end{cases} \quad (8b)$$

$x_{i,j}^{order}$ is updated by adding to it the number of successful subintervals between L and $x_{i,j}$ currently being resolved. Hence

$$x_{i,j}^{order} = x_{i,j}^{order} + \sum_{\rho=1}^{k} s_\rho \quad (9)$$

where $u_k \leq x_{i,j} < u_{k+1}$, $1 \leq k \leq t$, $u_{t+1} = U$

A selection process may be terminated when all numbers to be selected are identified. The termination point in Processor i is set with respect to the locally generated random variates. The search terminates in Processor i when either all locally generated variates are not among the t smallest variates in the network, or all subintervals less than the termination point have been resolved. To set the termination point, an improved lower bound $x_{i,j}^{neworder}$ on the order of $x_{i,j}$ in the N random variates is computed. In computing this lower bound, it is assumed that only two random variates are contained in a collided subinterval. Using $x_{i,j}^{order}$ computed in Eq. (9),

$$x_{i,j}^{neworder} = x_{i,j}^{order} + 2 \cdot \sum_{\rho=1}^{k} d_i \quad (10)$$

where $u_k \leq x_{i,j} < u_{k+1}$ $1 \leq k \leq t$, $u_{t+1} = U$

The improved lower bound cannot replace $x_{i,j}^{order}$ because the state of a collided subinterval may be changed in subsequent contention steps, and a more complicated adjustment to $x_{i,j}^{neworder}$ than that of Eq. (9) is needed when this happens. T, the termination point, is set as

$$T = x_{i,r} \quad \text{where } x_{i,r}^{neworder} \leq t < x_{i,r+1}^{neworder} \quad (11)$$

A boundary condition that $x_{i,t+1}^{order} = t+1$ is assumed. The search process in Processor i terminates when either $x_{i,1}^{neworder} > t$ or all subintervals in [L,T) have been resolved.

The sequencing and termination controls described above will be correct only if all subintervals in [L,T) are searched. We will show in Section Four that this is true if the proposed heuristic multi-window protocol is used.

A multi-window protocol based on the above interval-resolution procedure with heuristic window control is outlined in Figures 3 and 4. The function *observe*($\vec{S}$, $\vec{D}$) in the protocol is built upon the collision-detection mechanism of the multiple CSMA/CD busses.

The following example illustrates the operations of the protocol. Suppose there are three processors (M=3) and two busses (t=2). It is necessary to select two minima from the N (= M•t = 6) random variates uniformly distributed in [0,1). Assume the set of random variates to be {0.11, 0.14, 0.35, 0.65, 0.78, 0.96}. For Processor 1, suppose that it had generated $x_{1,1} = 0.14$ and $x_{1,2} = 0.65$. Initially, the window vector is $\vec{W} = \{[0,0.2), [0.2,0.4)\}$, the termination indicator in Processor 1 is $T_1 = 1.0$, and $x_{1,1}^{order} = x_{1,2}^{order} = 1$. After the first contention step, $\vec{S} = (0, 0)$, $\vec{D} = (0, 1)$. Hence we set $x_{1,1}^{order} = x_{1,2}^{order} = x_{1,1}^{neworder} = 1$, $x_{1,2}^{neworder} = 3$, and $T = 0.14$. $x_{1,2}$ is eliminated because it cannot be amongst the two smallest variates in the system. In the second contention step, $\vec{W} = \{[0.1,0.15), [0.15,0.2)\}$, and we obtain $\vec{S} = (0,0)$ and $\vec{D} = (1,0)$. $x_{1,1}^{order}$, $x_{1,1}^{neworder}$ and T remain unchanged. In the third contention step, $\vec{W} = \{[0.1,0.125), [0.125,0.15)\}$, and we obtain $\vec{S} = (1,1)$ and $\vec{D} = (0,0)$. Hence $x_{1,1}^{order} = 2$, and the contention process in Processor 1 stops because all subintervals between L and T have been resolved.

Since the windows are updated in each contention step and must be generated before the next contention step begins, a reasonable elapsed time must be set between the completion of collision detection and the initiation of the next contention step. To shorten this elapsed time, this protocol must be implemented in hardware. In systems where modification to existing hardware is impossible, the multi-window protocol can be implemented in software. Suppose that each bus is an existing Ethernet link with the broadcast capability and that processors can contend and communicate simultaneously and independently on each bus. Processors with variates inside the window of a bus will contend for the bus. The processor that is granted the bus will broadcast its variate. However, in this case, it is not clear whether exactly one processor or more than one processor have variates inside the window. (This is equivalent to a network with a two-state collision-detection capability.) Hence a verification phase must follow to assert that the broadcast variate is the only variate in this window. This verification phase can be implemented as a timeout period, so other processors with variates in this window can continue to contend and broadcast inside this timeout period. In each iteration of the multi-window protocol, the channel has to be contended twice, and two broadcasts of variates have to be made.

4. CORRECTNESS AND PERFORMANCE EVALUATION

In this section, we will prove that the proposed multi-window protocol correctly identifies the t smallest variates. Lemmas are stated without proof due to space limitation. The performance of the protocol is also discussed.

```
procedure multi-window-protocol-site-i (N, t, X_i, X_i^order);
/* N:      total number of distributed random variates;
   t:      number of random variates to be selected;
   L & U:  lower & upper bounds of the domain of random variates;
   X_i:    = (x_i,1, ..., x_i,t), the set of random variates generated by Processor i;
   X_i^order: = (x_i,1^order, ..., x_i,t^order) where x_i,j^order is the lower-bound order
           of x_i,j among the N distributed random variates (Eq. (9));
   X_i^neworder: = (x_i,1^neworder, ..., x_i,t^neworder) where x_i,j^neworder is the improved lower-bound
           order of x_i,j among the N distributed random variates (Eq. (10));
   W:      = {[0_1, u_1), ..., [0_t, u_t)}, window vector to be searched by the t busses;
   Q:      total number of subintervals with one variate in each;
   S & D:  collision-detection vectors;
   Φ:      lower bound of unsearched subinterval;
   T_i:    termination indicator of Processor i */

begin
    u_0 = L;
    for j=1 to t do
        begin
        s_j ← 0;    d_j ← 0;    x_ij^order ← 1;    0_j ← u_j-1;    u_j ← 0_j+(U-L)/N;
        end;
    x_i,t+1^neworder ← t+1;    Q ← 0;    T_i ← U;    done ← false;    Φ ← L+(U-L)t/N;
    while ((not done) and (not all x_i^order > t)) do
        begin
        transmit(X_i, W);    /* Transmit to the k'th channel if 0_i,k ≤ x_i,j < u_i,k */
        observe(S, D);    /* Detect outcome of contention. */
        for k=1 to t do    /* Update X_i^order and X_i^neworder */
            begin
            Q ← Q + s_k;    /* accumulate total number of subintervals with one variate in each */
            compute x_ij^order using Eq. (9);
            compute x_ij^neworder using Eq. (10);
            end
        update T_i using Eq. (11);
        if (all subintervals in [L,T) have been resolved) then
            done ← true
        else
            window (W, Q, D, T_i, Φ);    /* Determine the transmission windows */
        end;
end multi-window-protocol-site-i
```

Figure 3. Multi-window protocol for ordered selections.

```
procedure window (W, Q, D, T, Φ):
/* N:      total number of distributed random variates;
   t:      number of random numbers to be selected;
   L & U:  lower & upper bounds of the domain of random variates;
   W:      = {[0_1, u_1), ..., [0_t, u_t)}, window vector to be searched by the t busses;
   W^new:  temporary storage for new search windows;
   Q:      accumulated total number of subintervals with one variate in each;
   D:      collision indicator;
   Φ:      lower bound of unsearched subinterval;
   T:      termination marker */

begin
    i ← 1;  j ← 1;    /* i is index for busses, j is index for windows */
    while (i ≤ t and u_j ≤ T) do
        begin
        while (d_j=0) do j ← j+1;    /* search for a collided window */
        if (j≤t) then    /* still has unresolved collided subintervals */
            begin    /* allocate two busses to resolve a collided interval */
            0_i^new ← 0_j;    u_i^new ← (0_j+u_j)/2;
            0_i+1^new ← u_i^new;    u_i+1 ← u_j;
            i ← i+2;    j ← j+1;
            end;
        else begin  /* insufficient number of collided subintervals, search into unsearched subinterval */
            increment = (U − Φ) / (N − Q − 2^(t−1) Σ_{i=1}^{t} d_i);
            0_i^new ← Φ;
            for k=i to t do
                begin
                u_k^new = 0_k^new + increment;
                if k<t then 0_k+1^new = u_k^new
                end;
            Φ = u_t^new;    i ← t+1;
            end;
        end;
    W ← W^new;
end.
```

Figure 4. A heuristic window-control procedure.

4.1. Correctness

The following lemma shows that an ordered selection terminates in a finite number of steps.

Lemma 4.1: The multi-window protocol terminates in a finite number of steps if the random numbers to be selected are separable. Two random numbers, y_i and y_j, are separable if $\frac{1}{N(y_i - y_j)}$ is finite.

The following lemma proves the correctness of the termination condition.

Lemma 4.2: All subintervals in [L,T) are resolved when the multi-window protocol terminates, where T is the termination indicator.

The correctness of proposed multi-window protocol can be summarized in the following theorem.

Theorem 4.1: The multi-window protocol with the proposed heuristic window control performs an ordered selection correctly.

Proof: In Lemma 4.1, we have shown that the protocol terminates in a finite number of steps. According to Lemma 4.2, all subintervals in [L,T) are resolved when the process terminates. >From the way the termination indicators are set, it is easy to show that there are at least t numbers being isolated in these subintervals. Since resolved subintervals are disjoint and follow a linear ordering relation, the numbers isolated in these subintervals can be ordered correctly. □

4.2. Performance

Simulations have been conducted to evaluate the performance of the multi-window protocol. The simulator was coded in F77 and run on a VAX 11/780 computer. In the simulator, each processor generates a random number uniformly distributed in [0,1). A collision-detection mechanism is modeled by a counter that counts the number of random variates in a given subinterval. Different combinations of N and t were evaluated, each of which was run a number of times with different seeds until a 95% confidence interval of less than 0.2 was obtained. The simulation results are shown in Tables 1 and 2. They show that the average number of contention steps to identify the t smallest variates out of N random variates can be approximated by $\alpha(N,t)$ (=$0.8 \cdot \log_2 t + 0.2 \cdot \log_2 N + 1.2$) with less than 5% error. This approximation was obtained under the assumption that t busses are employed when t numbers are to be identified. Note that virtual loads and job response time are not uniformly distributed, they have to be transformed using Eq. 2.

The multi-window protocol does not rely on message exchanges and requires the time of one contention slot for each iteration. In previous systems, messages are transmitted through the message-passing subsystem and requires many layers of protocol conversions. Moreover, bus contentions are needed before the messages are sent. Hence the proposed protocol can be several orders of magnitude more efficient than previous protocols. On the other hand, the number of iterations required in a software implementation is the same, except that each iteration requires the time to contend and broadcast two random variates. This is still more efficient than conventional protocols that collect all variates to a centralized site and perform the scheduling there.

N=20										
t	2	4	6	8	10	12	14	16	18	20
$\overline{C}$	2.97	3.65	4.20	4.49	4.78	4.99	5.20	5.33	5.51	5.53
$\alpha(N,t)$	2.86	3.66	4.13	4.46	4.72	4.93	5.11	5.26	5.40	5.52
N=100										
t	10	20	30	40	50	60	70	80	90	100
$\overline{C}$	5.19	5.77	6.46	6.65	7.00	7.15	7.37	7.56	7.67	7.75
$\alpha(N,t)$	5.19	5.96	6.46	6.79	7.04	7.26	7.43	7.59	7.72	7.84
N=500										
t	50	100	150	200	250	300	350	400	450	500
$\overline{C}$	7.39	8.12	8.42	8.85	8.96	9.14	9.51	9.87	9.89	10.0
$\alpha(N,t)$	7.51	8.31	8.77	9.11	9.36	9.57	9.75	9.91	10.0	10.2

Table 1. Average number of contention steps ($\overline{C}$) to identify the t smallest numbers among N distributed random variates using the proposed ordered-selection protocol (N is fixed; $\alpha(N,t) = 0.2 \log_2 N + 0.8 \log_2 t + 1.2$ is given here for comparisons).

t=10										
N	30	60	90	120	150	180	210	240	270	300
$\overline{C}$	4.88	4.93	5.09	5.13	5.18	5.20	5.40	5.50	5.54	5.58
$\alpha(N,t)$	4.84	5.04	5.16	5.24	5.31	5.36	5.40	5.44	5.48	5.51
t=50										
N	50	100	150	200	250	300	350	400	450	500
$\overline{C}$	6.77	7.04	7.22	7.26	7.28	7.40	7.37	7.47	7.60	7.60
$\alpha(N,t)$	6.85	7.05	7.17	7.25	7.31	7.37	7.41	7.45	7.48	7.51

Table 2. Average number of contention steps ($\overline{C}$) to identify the t smallest numbers among N distributed random variates using the proposed ordered-selection protocol (t is fixed; $\alpha(N,t) = 0.2 \log_2 N + 0.8 \log_2 t + 1.2$ is given here for comparisons).

5. CONCLUDING REMARKS

Load balancing can effectively enhance resource sharing and reduce response times of jobs in a computer network. Accurate system status information is crucial to job-migration decisions in load balancing. There is a trade-off between the overhead spent in collecting updated status information and the benefit in making accurate migration decisions. In this paper, we have shown that finding the necessary status information for a class of load balancing strategies in multiple-bus networks can be reduced to the problem of ordered selections. An efficient multi-window protocol for supporting priority-based channel allocation using the primitive collision-detection mechanism of CSMA/CD networks is studied. It can identify the t smallest (or the largest) random variates among N spatially distributed contenders in about $0.8 \log_2 t + 0.2 \log_2 N + 1.2$ contention steps. An architecture to implement the proposed protocol is also presented.

The proposed protocol can be applied to other resource sharing applications. The capability to perform ordered selection in a distributed environment allows first-come-first-serve, priority, and shortest-job-first scheduling of message transmissions, and scheduling of a pool of resources among a set of contending processors. Techniques developed earlier to estimate channel load of single contention busses can be adapted to the case of multiple contention busses [20, 6, 7]. Due to the complexity of the resulting dynamic programming formulation, optimal solution similar to that developed for a single contention bus cannot be found [20]. Current research lies in statically evaluating the dynamic programming formulation into tables, which can be looked up in real time.

REFERENCES

[1] R. M. Bryant and R. A. Finkel, "A Stable Distributed Scheduling Algorithm," *Proc. 1st International Conference on Distributed Computing Systems*, pp. 314-323, 1981.

[2] Y. C. Chow and W. Kohler, "Models for Dynamic Load Balancing in a Heterogeneous Multiple Processor System," *Trans. on Computers*, vol. C-28, no. 5, pp. 334-361, IEEE, May 1979.

[3] Y. I. Gold and W. R. Franta, "An Efficient Collision-Free Protocol for Prioritized Access-Control of Cable Radio Channels," *Computer Networks*, vol. 7, pp. 83-98, North-Holland, 1983.

[4] K. Hwang, et al., "A UNIX-Based Local Computer Network with Load Balancing," *Computer*, pp. 55-64, IEEE, April 1982.

[5] J. Y. Juang and B. W. Wah, "Unified Window Protocol for Local Multiaccess Networks," *Proc. Third Annual Joint Conference of the IEEE Computer and Communications Societies*, pp. 97-104, IEEE, April 1984.

[6] J. Y. Juang, *Resource Allocation in Computer Networks*, Ph.D. Thesis, Purdue University, West Lafayette, IN, Aug. 1985.

[7] J. Y. Juang and B. W. Wah, "Channel Allocation in Multiple Contention Bus Networks," *Proc. Fifth Annual Joint Conference of the IEEE Computer and Communication Societies*, IEEE, April 1986.

[8] M. Livny and M. Melman, "Load Balancing in Homogeneous Broadcast Distributed Systems," *Proc. Modeling and Performance Evaluation of Computer Systems*, pp. 47-55, ACM SIGMETRICS, 1982.

[9] R. M. Metcalfe and D. R. Boggs, "Ethernet: Distributed Packet Switching for Local Computer Networks," *Comm. of the ACM*, vol. 19, no. 7, pp. 395-404, July 1976.

[10] L. M. Ni and X. Li, "Prioritizing Packet Transmission in Local Multiaccess Networks," *Proc. Eighth Data Communications Symposium*, IEEE, 1983.

[11] L. M. Ni and K. Hwang, "Optimal Load Balancing in a Multiple Processor System with Many Job Classes," *Trans. on Software Engineering*, vol. SE-11, no. 5, pp. 491-496, IEEE, May 1985.

[12] L. M. Ni, C.-M. Xu, and T. B. Gendreau, "A Distributed Drafting Algorithm for Load Balancing," *Trans. on Software Engineering*, vol. SE-11, no. 10, pp. 1153-1161, IEEE, Oct. 1985.

[13] N. Shacham, "A Protocol for Preferred Access in Packet-Switching Radio Networks," *Trans. on Communications*, vol. COM-31, no. 2, pp. 253-264, IEEE, Feb. 1983.

[14] J. F. Shock, et al., "Evolution of the Ethernet Local Computer Network," *Computer*, vol. 15, no. 8, pp. 10-27, IEEE, Aug. 1982.

[15] W. Stallings, *Local Networks: An Introduction*, Macmillan, New York, 1984.

[16] H. S. Stone, "Critical Load Factors in Two-processor Distributed Systems," *Trans. on Software Engineering*, vol. SE-4, pp. 254-259, IEEE, May 1978.

[17] A. S. Tanenbaum, *Computer Networks*, Prentice Hall, New Jersey, 1981.

[18] F. A. Tobagi, "Carrier Sense Multiple Access with Message-Based Priority Functions," *Trans. on Communications*, vol. COM-30, no. 1, IEEE, Jan. 1982.

[19] D. Towsley, "Queuing Network Models with State-Dependent Routing," *Journal of ACM*, vol. 27, no. 2, pp. 323-337, ACM, April 1980.

[20] B. W. Wah and J. Y. Juang, "Resource Scheduling for Local Computer Systems with a Multiaccess Network," *Trans. on Computers*, vol. C-34, no. 12, pp. 1144-1157, IEEE, Dec. 1985.

[21] B. Walker, et al., "The LOCUS Distributed Operating System," *Proc. Nineth ACM Symp. on Operating System Principles*, pp. 49-70, ACM, 1983.