JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING 7, 391–415, 1989

Load Balancing and Ordered Selections in a Computer System with Multiple Contention Buses

JIE-YONG JUANG

Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, Illinois 60208

AND

BENJAMIN W. WAH*

Department of Electrical and Computer Engineering and the Coordinated Science Laboratory, University of Illinois at Urbana–Champaign, Urbana, Illinois 61801

Received February 4, 1986

In this paper, we study an efficient scheme for disseminating status information in a distributed computer system connected by multiple contention buses. Such a scheme is critical in resource sharing and load balancing applications. The collection of status information in these systems usually incurs a large overhead, which may impede regular message traffic and degrade system performance. Moreover, the status information collected may be outdated due to network delays. We describe our scheme with respect to the load balancing problem, although the scheme developed applies to resource sharing applications in general. We first reduce the decision problem for job migration in a system with multiple contention buses to the ordered-selection problem. A heuristic multiwindow protocol that utilizes the collision-detection capability of these buses is proposed and 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 approximately $(0.8 \log_2 t + 0.2 \log_2 N + 1.2)$ contention steps.

1. INTRODUCTION

Advances in communication technology have resulted in the development of high-speed media, such as optical-fiber links and coaxial cables, for local computer networks. Although the potential bandwidth of these media is very

* Research supported by the National Science Foundation under Grant MIP 85-19649 and the National Aeronautics and Space Administration under Contract NCC 2-481.

0743-7315/89 $3.00

large, the current bandwidth used on most of these media is less than 50 million bits per second (Mbps). The scheduling schemes of current communication subsystems are built around networks of low bandwidth, such as the notable Ethernet and token-ring networks [15, 19, 20], and are not effective when higher bandwidth is available. Such a limitation is illustrated in the contention-resolution mechanism of Ethernet. When a bandwidth much higher than the currently popular 10 Mbps is used, the time taken for data transmission is shortened, while the time taken for contention resolution remains unchanged due to the speed-of-light limitation. To have better overall utilization of the channel for data transmission, it may be necessary to partition the single high-bandwidth channel into multiple low-bandwidth channels. A frequency-multiplexed configuration is becoming popular and practical with optical-fiber networks [6].

In this paper, we study an efficient protocol for disseminating status information for resource sharing in a local computer system connected by a set of contention buses, each behaving as an independent Ethernet. The protocol is described with respect to load balancing, although the same scheme applies to other resource sharing applications in general. The multiple buses can either be frequency multiplexed in a single high-bandwidth bus or exist as separate physical links. It is further assumed that job arrivals at processors are independent and may have different arrival rates and work demands. As a result, some of the processors may be heavily loaded while others may be idle.

There are three major issues that affect the design of a resource scheduling algorithm in a resource sharing computer system: network blockage (or packet congestion), request conflicts, and imbalance in workload. In such a system, a task may be assigned to one of a set of resources, and multiple requests may contend for the same resource. If the resource has no buffers, then all requests except one must be rescheduled again. This problem is called *resource conflicts.* On the other hand, if resources have local buffers, then the tasks can be queued at the resources regardless of conflicts, and load balanced when workload is not balanced. In a system with unbalanced workload, backlogged jobs waiting in heavily loaded processors can be migrated to lightly loaded processors and executed there. This has the effect 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 migrate from one processor to another. These remote executions of jobs and virtual terminal protocols [8, 22] are common in many local computer systems. This form of load balancing balances workload on a per-session basis but cannot effectively reduce the number of backlogged jobs, since a session is a mix of user think time and job-execution time. In contrast, *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 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. Note that a resource scheduling scheme that minimizes resource conflicts (or perfectly balances the workload) is not necessarily optimal, since paths leading from a requesting source to the designated resource may be blocked in a circuit-switched mode or may be heavily congested in a packet-switched mode.

Three issues are identified in making job-migration decisions in a user-transparent load balancing scheme: deciding the times to initiate job migrations, determining the jobs to be migrated, and finding the places to send the migrated jobs. Previous studies have provided various answers to these questions. In ECN [8], LOCUS [26], MACH [1], and V [4], 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 [3], a job is transferred to a neighboring processor whenever it is possible, and is rippled to a proper site later. In the approach using thresholds, job migrations are carried out whenever the workload in the local processor exceeds a threshold. The place to send a migrated job is based on a dynamic load table maintained at each processor [17]. For instance, jobs in Rediflow are migrated according to the difference between the local "internal pressure" and the neighboring "external pressure" of jobs [12]. In another approach using thresholds, a processor tries to bid for a job whenever it becomes idle [14]. Jobs can also be preempted and migrated to another processor when the workload in the current processor increases, as demonstrated in the University of Wisconsin system [13]. No previous work addresses the issue on the jobs to be transferred.

A load balancing scheme that supports migration by dynamic thresholds in multiple-contention-bus networks is presented in this paper. Status information and the overhead of collecting it are two critical factors in effective load balancing. In Section 2, the problems of global job bidding and individual job-migration decisions are formulated into an optimization problem and are shown to be equivalent to performing ordered selections. A multi-window protocol to perform ordered selections without explicit message transfers is discussed in Section 3. Section 4 presents the correctness proof and evaluates the performance of the protocol, and Section 5 draws conclusions.

2. Load Balancing in a Computer Network with Multiple Buses

In this section, we describe a load balancing scheme for a distributed system connected by t ($t \geq 1$) contention buses. We first discuss the general

problem of load balancing. Then we show that the collection of the necessary status information for load balancing in such a system can be reduced to ordered selections. Last, 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 includes processor workloads, network status, and the characteristics of the arrival processes [5]. There is a trade-off between the overhead spent on maintaining accurate status information in the processors and the increase in response time caused by using outdated status information. In this section, we describe a mechanism for processors to coordinate in making job-migration decisions, assuming that accurate status information is available. The method for obtaining this information is discussed later.

The status of a processor can be characterized by two random variables: its current workload and the response time of a job when migrated to this processor (called *virtual load*). Let $\mathbf{L}$ be a set of observations, each of which represents the current workload of a processor, and $\mathbf{R}$ 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 $\mathbf{R}$ to $\mathbf{L}$. 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 response time of this job after migration; y_i, the load of the remote processor; and θ_i, the cost of such a migration. An optimal mapping can be obtained by maximizing the total utility over all possible mappings,

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

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

It is hard to obtain a general solution to the above optimization problem because it is difficult to estimate the costs of migrations and predict 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 performance of a processor to a set of virtual loads. The jth virtual load of Processor i is the predicted workload after Processor i has received j migrated jobs. The resulting formulation using this generalization is still very complicated because there may be network interference when

more than one job is sent to the same destination. Further, the costs of sending jobs to the same destination may not be a linear function of the number of jobs sent.

The mapping problem in Eq. (1) can be simplified in some special cases. In the next section, we discuss such a special case for a network with multiple buses. It is assumed in this network that there is no interference in sending jobs across the multiple buses, and that the costs of sending jobs are constant, independent of the source and destination.

2.2. *Migration Decision in a Computer Network with Multiple Buses*

The cost of migrating a job in 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.

To simplify the job-migration decisions while retaining good performance, the following assumptions are adopted. (a) The preference of a job to be migrated is characterized by a *bidding parameter* that measures the expected improvement in response time. It is a function of the local response times, the volume of data involved in the migration, and the average packet delay. There is also minimal interference of migrating multiple jobs to the same site using multiple buses, hence the bidding parameters are not affected in such a case. (b) Load balancing is carried out only when there are no regular messages 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 to those used for load balancing. It can be supported by CSMA/CD protocols that facilitate transmissions with priority [7, 9, 16, 18, 23, 25]. (c) When a job is migrated, the necessary data and control information is 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 [21], which is outside the scope of our study for supporting load balancing by low-level network protocols. (d) All jobs are stochastically identical, hence any job in the queue can be migrated. (e) The number of buses 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 buses simultaneously and that the interference of receiving jobs from multiple buses concurrently is minimal. (f) 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. Multi-

plexing packets of multiple migrated jobs simultaneously on a single bus delays the completion of all these jobs, hence is not effective. (g) The buses are reliable and information broadcast can be received by all processors simultaneously and correctly. (h) All processors in the network participate in identifying the sources and destinations of load balancing, hence the number of virtual loads and bidding parameters in the network is constant. (i) All bidding parameters and virtual loads are assumed to be uniformly distributed between L and U. If not, statistical distributions collected from information received are used to adjust the random variates according to the equation

$$x_{i,j} = F(x'_{i,j}) \cdot (U - L) + L, \tag{2}$$

where $x'_{i,j}$ is the jth local variate of Processor i, and F is the statistical distribution of the random variates. F is identical among the processors as they receive identical information from the buses.

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. Due to the assumption on minimal interference of migrating multiple jobs to the same site using multiple buses, 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

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

With t buses, 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. Equation (3) can be reduced to the form

$$\begin{aligned} S(\mathbf{R}, \mathbf{L}) &= \max_{\Omega \in \mathbf{R} \times \mathbf{L}} \sum_{i=1}^{t} \beta(x_i, \Omega(x_i)), \qquad \text{where} \\ \beta(x_i, \Omega(x_i)) &\leq \beta(x_j, \Omega(x_j)) \qquad \text{for} \qquad i < j. \end{aligned} \tag{4}$$

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

THEOREM 2.1. *To maximize the utility of load balancing in Eq.* (4), *t requests of the largest bidding parameters should be mapped to the t smallest virtual loads if β is a nondecreasing function of the job bidding parameter and a nonincreasing 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_is, the inequality

$$\beta(x_q, \Omega(x_p)) \geq \beta(x_p, \Omega(x_p)) \quad \text{for} \quad x_q > x_p \quad \text{and} \quad 1 \leq p \leq t < q \tag{5}$$

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

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

The above inequality indicates that 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 buses, 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 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, one solution is 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 Buses*

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., no, one, or more than one processor 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 algorithm proposed in this paper, 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 these subintervals 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 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 buses available.

The example in Fig. 1 illustrates the interval-resolution process. In this example, the order of 10 random variates generated by six processors is to be determined, and two contention buses 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, one of the numbers generated by Processor 3 is determined to be the minimum, and the second minimum is also identified. To determine the order of others, further contentions must be carried out.

Processors involved in ordered selections must 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 a message and synchronizing all the processors incur a significant overhead. Alternatively, if all processors are evaluating an identical algorithm with identical inputs, then the windows are synchronized

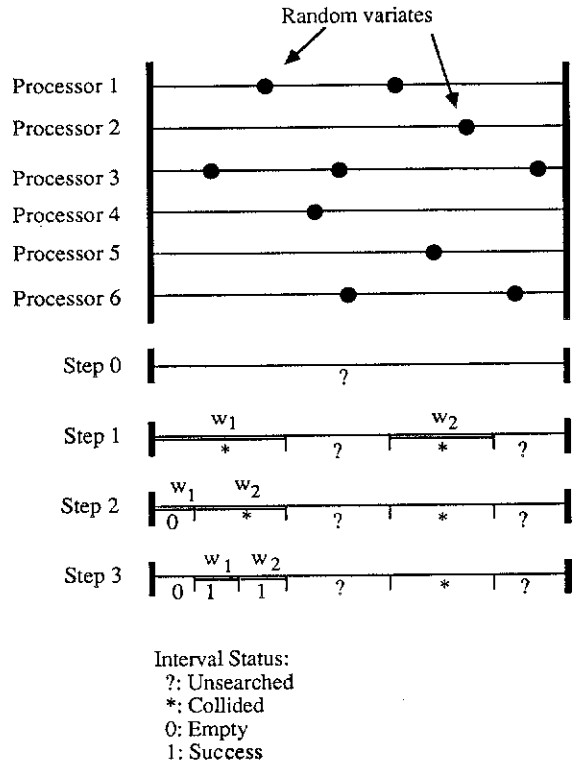

FIG. 1. An example illustrating an interval-resolution procedure for ordered selection (windows used in each step are labeled as w_1 and w_2).

automatically without any message transfer. The identical inputs needed are information readily available from the collision-detection mechanism.

A potential discrepancy happens when more than one variate are generated in a processor and fall in a single transmission window, while no other variates in other processors fall in this window. In this case, the potential collision is only known to the local processor, and other processors interpret that a successful contention has been made for this transmission window. A simple solution to this problem is for the processor concerned to transmit a special code to indicate self-collision. This code is received by all processors in the system, which refines the transmission window concerned in a way similar to that of collided ones.

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 Fig. 2. In this architec-

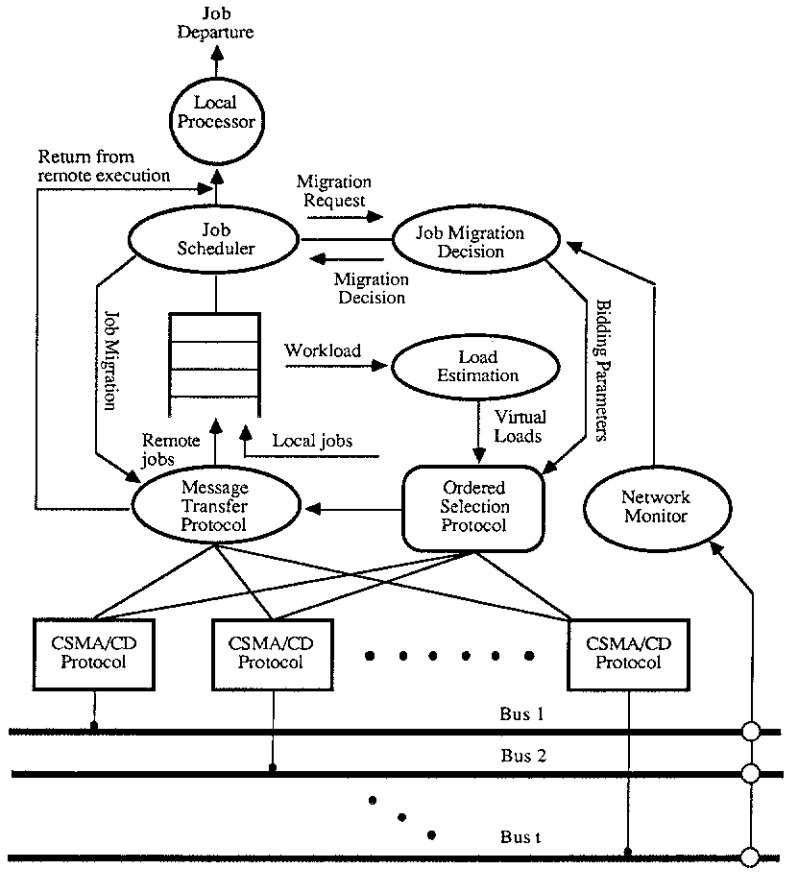

FIG. 2. Architecture of a processor to support load balancing in a computer network with multiple contention buses.

ture, 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 buses 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 in 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 of 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.

3. Multiwindow 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. Previously, we developed a window-control scheme for the degenerate case of a single contention bus [9, 25]. In this section, we describe a *multiwindow control scheme* for parallel interval resolutions on multiple buses.

In the multiwindow control scheme, window generations in different processors are synchronized by collision detections. A *contention step* consists of the generation of transmission windows, one for each bus; the contention of each bus; 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. The generation of transmission windows involves local processing and must be optimized.

3.1. *Optimal Multiwindow 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, unresolved subintervals are represented by vector $\mathbf{V}$. Each element of $\mathbf{V}$ is a subinterval represented by a triplet that consists of the lower and upper bounds and the status of the subinterval (empty, success, collided, or unsearched). Based on such a representation and since the Principle of Optimality is satisfied, the optimization of multiwindow control can 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 $\mathbf{V}$. Given $\mathbf{V}$, a contention step using window vector $\mathbf{W}$ will result in another set of unresolved subintervals $\mathbf{U}$. Denote the expected number of contention steps to complete the ordered selection by $C^t_{N,\mathbf{V}}$. The dynamic programming formulation for window generation may be expressed recursively as

$$C^t_{N,\mathbf{V}} = \min_{w} \{1 + \sum_{i=1}^{t} \sum_{\mathbf{U}} p_{N,t,i,\mathbf{V},\mathbf{U}}(\mathbf{W}) \cdot C^{t-i}_{N-i,\mathbf{U}}\}, \tag{7}$$

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

To evaluate Eq. (7), all $p_{N,t,i,\mathbf{V},\mathbf{U}}(\mathbf{W})$'s must be known. Since as many as t buses can be assigned to resolve a subinterval, there are K^t possible ways of assigning t buses 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 $p_{N,t,i,\mathbf{V},\mathbf{U}}(\mathbf{W})$. Consequently, Eq. (7) is too complex to be evaluated exhaustively. Suboptimal solutions to the window-generation problem are presented in the next section.

3.2. *A Multiwindow Protocol with Heuristic Window Control*

(i) *Heuristic Window Control*

To solve the window-generation problem heuristically, the following observations are made. First, random variates of the smallest values are to be selected, hence unresolved subintervals at the end with the smaller values are to 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 can be shown to have a size of u/N each (see the Appendix). It can also be shown that the average number of random variates falling in a subinterval of size u/N, given that collision is detected, is less than 2.4 (see the Appendix). In such a case, it is reasonable to divide the collided subinterval into two equal halves and search each independently.

Initially, the entire domain $[L, U)$ of size u $(=U - L)$ to be searched is taken as an unsearched interval. Given t buses, 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 part of the domain containing the smaller values are searched first. Note that there is only one unsearched subinterval $[L + tu/N, U)$ at the end of the domain containing the larger values. 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 and partitioning steps 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 un-

searched subinterval, Q, the number of random variates successfully isolated in $[L, \Phi)$, is used to estimate the upper bound of the number of random variates in $[\Phi, U)$. The unsearched subintervals are divided into smaller subintervals of size $(U - \Phi)/Q$ each.

It is shown in Section 4 that the window-control algorithm based on the above steps is correct and performs satisfactorily. In the Appendix, we discuss the rationale behind the proposed window-control rule.

(ii) *Sequencing and Termination*

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

To reduce the memory requirement, a constant-memory scheme is proposed here. For Processor i, only the order of the locally generated random variates $x_{i,j}, \ldots, x_{i,t}$ has to be known (assuming $x_{i,j} \leq x_{i,j+1}$, $1 \leq j < t$). 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 $x_{i,j}^{\text{order}}$. An estimate on the number of variates in collided subintervals between L and $x_{i,j}$ may be useful, but not essential, to improve $x_{i,j}^{\text{order}}$.

$x_{i,j}^{\text{order}}$ is initialized to be 1 and is updated as follows.[1] In each iteration, t subintervals as specified by $\mathbf{W}$ are assigned to the t available buses for contention resolution; $[l_j, u_j)$ is assigned to Bus j, where $l_j \geq u_{j-1}$, $u_j \leq l_{j+1}$, and $2 \leq j \leq t - 1$. The outcomes of collision detection are represented by two t-tuples, $\mathbf{S} = (s_1, \ldots, s_t)$ and $\mathbf{D} = (d_1, \ldots, d_t)$, where

$$s_j = \begin{cases} 1 & \text{if contention for the } j\text{th bus in interval } [l_j, u_j) \\ & \quad \text{results in successful transmission,} \\ 0 & \text{otherwise.} \end{cases} \tag{8a}$$

$$d_j = \begin{cases} 1 & \text{if contention for the } j\text{th bus in interval } [l_j, u_j) \\ & \quad \text{results in collision,} \\ 0 & \text{otherwise.} \end{cases} \tag{8b}$$

$x_{i,j}^{\text{order}}$ is updated by adding to it the number of successful subintervals between L and $x_{i,j}$ that have been resolved in the current iteration. Hence,

[1] $x_{i,j}^{\text{order}}$ could be initialized to be j initially. We did not do so in order to simplify the update rules for $x_{i,j}^{\text{order}}$ and $x_{i,j}^{\text{neworder}}$.

$$x_{i,j}^{\text{order}} = x_{i,j}^{\text{order}} + \sum_{\rho=1}^{k} S_\rho, \quad \text{where} \quad u_k \leq x_{i,j} < u_{k+1}, \quad 1 \leq k \leq t, \quad u_{t+1} = U. \tag{9}$$

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, $x_{i,j}^{\text{neworder}}$, an improved lower bound 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 $x_{i,j}^{\text{order}}$ computed in Eq. (9),

$$x_{i,j}^{\text{neworder}} = x_{i,j}^{\text{order}} + 2 \cdot \sum_{\rho=1}^{k} d_\rho, \qquad \text{where} \tag{10}$$

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

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

$$T = x_{i,\tau}, \qquad \text{where} \qquad x_{i,\tau}^{\text{neworder}} \leq t \leq x_{i,\tau+1}^{\text{neworder}}. \tag{11}$$

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

The sequencing and termination controls described above are correct only if all subintervals in $[L, T)$ have been searched. We show in Section 4 that this is true if the proposed heuristic multiwindow protocol is used.

A multiwindow protocol based on the above interval-resolution procedure with heuristic window control is outlined in Fig. 3. The function *observe*(**S**, **D**) in the protocol is based upon the collision-detection mechanism of the multiple CSMA/CD buses.

The following example illustrates the operations of the protocol. It is assumed that there are three processors ($M = 3$) and two buses ($t = 2$), and that it is necessary to select two minima from the $N (= M \cdot t = 6)$ 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 $x_{1,1} = 0.14$ and that $x_{1,2} = 0.65$. Initially, the window vector is $\mathbf{W} = \{[0, 0.167), [0.167, 0.333)\}$. The termination indicator in Processor 1 is $T_1 = 1.0$, and

```
procedure multiwindow-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         = {[l_1, u_1), . . . , [l_t, u_t)}: window vector to be searched, one for each of the t buses;
   Q:        total number of subintervals with one variate in each;
   S & D:    collison-detection vectors;
   Φ:        lower bound of the 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_{i,j}^order ← 1;  l_j ← u_{j-1};  u_j ← l_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, W);   /* Transmit to the kth channel if l_{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
      /* accumulate total number of subintervals with one variate in each */
      Q←Q+s_k;
      compute x_{i,j}^order using Eq. (9);
      compute x_{i,j}^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 multiwindow-protocol.site.i
```

FIG. 3a. Multiwindow protocol for ordered selections.

$x_{1,1}^{\text{order}} = x_{1,2}^{\text{order}} = 1$. After the first contention step, $\mathbf{S} = (0, 0)$, $\mathbf{D} = (1, 0)$. Hence, we set $x_{1,1}^{\text{order}} = x_{1,2}^{\text{order}} = x_{1,1}^{\text{neworder}} = 1$, $x_{1,2}^{\text{neworder}} = 3$, and $T_1 = 0.14$. $x_{1,2}$ is eliminated because it cannot be among the two smallest variates in the system. In the second contention step, $\mathbf{W} = \{[0.0, 0.083), [0.083, 0.167)\}$, and we obtain $\mathbf{S} = (0, 0)$ and $\mathbf{D} = (0, 1)$. $x_{1,1}^{\text{order}}$, $x_{1,1}^{\text{neworder}}$, and T_1 remain unchanged. In the third contention step, $\mathbf{W} = \{[0.083, 0.125), [0.125, 0.167)\}$, and we obtain $\mathbf{S} = (1, 1)$ and $\mathbf{D} = (0, 0)$. Hence, $x_{1,1}^{\text{order}} = 2$, and

```
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       = {[l1, u1), . . . , [lt, ut)}, window vector defined for the t buses;
   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 the unsearched subinterval;
   T:      termination marker */

begin
  i ← 1;  j ← 1;  /* i is the index for buses, j is the index for windows */
  while (i≤t and u_j≤T) do
    begin
    while (j≤t and d_j=0) do j ← j+1;  /* search for a collided window */
    if (j ≤ t) then  /* still has unresolved collided subintervals */
      begin          /* allocate two buses to resolve a collided interval */
      l_i^new ← l_j;      u_i^new ← (l_j+u_j)/2;
      l_{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 to allocate to the t buses, search the
         unsearched subinterval */
      increment = (U−Φ)/(N−Q−2 · Σ_{k=1}^{t} d_j)
      l_i^new = Φ;
      for k=i to t do
        begin
        u_k^new = l_k^new + increment;
        if k<t then l_{k+1}^new=u_k^new
        end;
      Φ = u_t^new;  i ← t+1;
      end;
    end;
  W ← W^new;
end.
```

FIG. 3b. A heuristic window-control procedure.

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 amount of time must have elapsed between the completion of collision detection and the initiation of the next contention step. To shorten this elapsed time, it may be necessary to implement the proposed protocol in hardware.

In systems in which modification to existing hardware is impossible, the multiwindow 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 defined for a bus 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 has variates inside the window. (This is equivalent to a network with a two-state collision-detection capability.) There are three alternatives here. First, all contending processors proceed to reduce the windows after the variate has been broadcast (the one-broadcast approach). A disadvantage of this approach is that it may take a substantial amount of time to subdivide the interval containing no variate into smaller subintervals until the boundary condition is reached. Second, a verification phase can follow the broadcast to assert that the broadcast variate is the only variate in this window (the two-broadcast approach). 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. If no processor broadcast in this timeout period, then this subinterval does not have to be further partitioned. In each iteration of the multiwindow protocol, the channel must be contended twice, and two broadcasts of variates must be made. Third, a judicious selection between the one-broadcast and two-broadcast approaches can be made in each iteration (the combined approach). We have studied the above three alternatives for the single-bus case [2] and have found that the one-broadcast approach is the simplest and performs almost as well as the other two approaches. The one-broadcast approach is also the best approach to apply in the multibus case, as each bus operates independently.

4. Correctness and Performance Evaluation

In this section, we prove that the proposed multiwindow protocol correctly identifies the t smallest variates. The performance of the protocol is also discussed.

4.1. *Correctness*

First, we show that the protocol terminates in a finite number of steps.

Lemma 4.1. *The multiwindow 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 $1/(N(y_i - y_j))$ is finite.*

Proof. The procedure terminates when all disjoint subintervals below the termination indicator have been resolved. It is necessary to show that there are a finite number of such subintervals and that these subintervals were obtained from partitioning the search range in a finite number of steps. To

search an unresolved subinterval, the window control determines subintervals of nonzero size as transmission windows and partitions the search range into a finite number of subintervals. These subintervals may be resolved or remain unresolved after a contention step. If a subinterval remains unresolved, it is split into two halves of finite sizes. The maximum number of steps to separate any two random numbers is

$$k = \left\lceil \log_2 \frac{1}{N \cdot \delta} \right\rceil , \tag{12}$$

where $\delta = \min\{|y_i - y_j|, 1 \leq i, j \leq t, \text{ and } i \neq j\}$. Since δ is finite, so is k. Thus the procedure terminates in a finite number of steps. ■

If two numbers to be selected are not separable, then they always fall in the same subinterval and result in infinite collisions. Theoretically, the probability for two continuous random numbers to be unseparable is zero. In practice, two random numbers generated may be identical because they are represented in a limited number of bits. To prevent running into infinite collisions, each random variate may be augmented by either another site-dependent constant, such as the site identifier, or a distinct sequence of small random numbers such that the original order of random variates is not disturbed.

The following lemma proves the correctness of the termination condition.

LEMMA 4.2. *All subintervals in* $[L, T)$ *are resolved when the multiwindow protocol terminates, where* T *is the termination indicator.*

Proof. In providing this lemma, it is necessary to show that the value of the termination indicator becomes smaller as the resolution process proceeds. Without loss of generality, assume that Processor i has generated a random variate $x_{i,j}$ to be sent on the tth bus. After the first contention step, there are two possible outcomes.

$$\left.\begin{aligned} x_{i,j}^{\text{neworder}} &\leq t \\ x_{i,j}^{\text{neworder}} &> t \end{aligned}\right\}, \quad \text{where} \quad x_{i,j}^{\text{neworder}} = x_{i,j}^{\text{order}} + 2 \cdot \sum_{\rho=1}^{t-1} d_\rho . \tag{13}$$

If $x_{i,j}^{\text{neworder}} \leq t$ and since $x_{i,j}^{\text{order}} \geq 1$, it implies that $2 \cdot \sum_{\rho=1}^{t-1} d_j < t$, and that the number of unresolved subintervals in $[L, T)$ is less than the number of available buses. Hence, the resolution process must be extended into the unsearched range in the next contention step. The positions of the termination indicators in all processors remain unchanged.

Since subintervals are repeatedly partitioned as contention proceeds, $x_{i,j}^{\text{neworder}}$ will increase, and eventually $x_{i,j}^{\text{neworder}} > t$. In this case, there are more subintervals to be resolved than the number of available buses and more than

t variates with values smaller than the termination indicator. There exists an index k, $1 \leq k \leq t$, such that

$$x_{i,k}^{\text{order}} + 2 \cdot \sum_{j=1}^{\tau-1} d_j = \begin{cases} t, & \text{success in the } \tau\text{th bus,} \\ t+1, & \text{collision in the } \tau\text{th bus,} \end{cases} \tag{14}$$

where $x_{i,k}$ is another random variate generated to contend for the τth bus. As a result, the termination indicators in all processors are moved to a point smaller than the upper bound of window used in the τth bus. The same argument can be repeated, and the value of the termination indicator is monotonically nonincreasing. Eventually, all subintervals in $[L, T)$ will be resolved when the process terminates. ■

The correctness of the proposed multiwindow protocol can be summarized in the following theorem.

THEOREM 4.1. *The multiwindow protocol with the proposed window-control heuristic 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 multiwindow 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 I and II. 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 a 5% error. This approximation was obtained under the assumption that t buses are employed when t numbers are to be identified.

Hardware implementation of the multiwindow protocol has been discussed elsewhere and will not be shown here [10]. Such a hardware implementation does not rely on message exchanges and requires the time of one

TABLE I
AVERAGE NUMBER OF CONTENTION STEPS ($\bar{C}$) FOR IDENTIFYING THE t SMALLEST NUMBERS AMONG N DISTRIBUTED RANDOM VARIATES USING THE PROPOSED ORDERED-SELECTION PROTOCOL

					$N = 20$					
t	2	4	6	8	10	12	14	16	18	20
$\bar{C}$	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
$\bar{C}$	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
$\bar{C}$	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

Note. N is fixed; $\alpha(N, t) = 0.2 \log_2 N + 0.8 \log_2 t + 1.2$ is given here for comparisons.

contention slot for each iteration. In contrast, in previous systems, messages are transmitted through the message passing subsystem and many layers of protocol conversions are required. Hence, the proposed protocol can be several orders of magnitude more efficient than previous protocols. Likewise, in a software implementation of the proposed protocol, the number of iterations required is the same as that of a hardware implementation, except that

TABLE II
AVERAGE NUMBER OF CONTENTION STEPS ($\bar{C}$) FOR IDENTIFYING THE t SMALLEST NUMBERS AMONG N DISTRIBUTED RANDOM VARIATES USING THE PROPOSED ORDERED-SELECTION PROTOCOL

					$t = 10$					
N	30	60	90	120	150	180	210	240	270	300
$\bar{C}$	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
$\bar{C}$	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

Note. t is fixed; $\alpha(N, t) = 0.2 \log_2 N + 0.8 \log_2 t + 1.2$ is given here for comparisons.

each iteration is the time to contend for the bus and broadcast a random variate. This is still more efficient than conventional protocols that collect all variates to a centralized site and perform the scheduling there.

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 tradeoff 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 contention-bus networks can be reduced to the problem of ordered selections. An efficient multiwindow protocol for supporting priority-based channel allocation using the primitive collision-detection mechanism of CSMA/CD networks was studied. It can identify the t smallest (or the largest) random variates among N spatially distributed contenders in approximately $0.8 \log_2 t + 0.2 \log_2 N + 1.2$ contention steps. An architecture to implement the proposed protocol was also presented.

The proposed protocol can be applied to other resource sharing applications. The capability to perform ordered selection in a distributed environment allows various scheduling strategies, such as the first-come-first-serve, priority, and shortest-job-first disciplines, and the sharing of a pool of resources among a set of contending processors. Techniques developed earlier to estimate the channel load of single contention buses can be adapted to the case of multiple contention buses [10, 11, 25]. Due to the complexity of the resulting dynamic programming formulation, an optimal solution similar to that developed for a single contention bus [25] cannot be used. Greedy heuristics using a good initial partitioning of the interval of random variates usually perform satisfactorily.

APPENDIX: Analysis of Heuristic Window Control

First, we show that partitioning a collided interval into two equal halves is a good heuristic rule. It has been proved that the binary-divide strategy is optimal for resolving a collided interval if there are two variates uniformly distributed in this interval [24]. Suppose that the size of the unsearched interval is u and that the number of random variates falling in this interval is N. Let the size of the window to be searched initially be u/N, and Z be the random variable representing the number of y_i's in such a window, assuming that a collision has been detected. Z has a binomial distribution since the y_i's are uniformly distributed, and the probability that a given variate falls in the

window is $1/N$. The expected value of Z conditioned on detection of a collision is

$$E(Z \mid \text{collision detected}) = \frac{\sum_{p=2}^{N} p\binom{N}{p}\left(\frac{1}{N}\right)^p\left(1-\frac{1}{N}\right)^{N-p}}{1-\left(1-\frac{1}{N}\right)^N-\left(1-\frac{1}{N}\right)^{N-1}} = \frac{1-\left(1-\frac{1}{N}\right)^{N-1}}{1-\left(2-\frac{1}{N}\right)\left(1-\frac{1}{N}\right)^{N-1}}, \quad \text{(A1)}$$

where the denominator is the probability that there is a collision in the window. When N is 2, $E(Z \mid \text{collision detected})$ is 2, while in the limiting case,

$$\lim_{N\to\infty} E(Z \mid \text{collision detected}) = \frac{1-e^{-1}}{1-2\cdot e^{-1}} = 2.392. \quad \text{(A2)}$$

A collided window contains between 2 and 2.392 variates on the average. The arguments above indicate that a collided window often contains slightly over two variates, and that the binary-divide rule is a good approximation to the optimal window control.

The following lemma and theorem show that the heuristic rule of determining the interval size in the unsearched range is also an efficient strategy.

LEMMA A1. *Let p be a natural number, and $w_i > 0$. If $\sum_{i=1}^{p} w_i = c$, then $\prod_{i=1}^{p} w_i$ is maximized when the w_i's are equal. That is, $w_i = c/p$, $i = 1, \ldots, p$.*

Proof. The lemma is proved by mathematical induction. The induction basis ($p = 1$) is trivial. Consider the case in which p is greater than one. Assume that

$$\sum_{i=1}^{p-1} w_i = \beta c, \qquad \text{where} \qquad 0 < \beta < 1. \quad \text{(A3)}$$

Accordingly, $w_p = (1-\beta)c$. If p is equal to 2, then

$$w_1 \cdot w_2 = c^2\beta(1-\beta). \quad \text{(A4)}$$

The RHS of Eq. (A4) is maximized when $\beta = 0.5$, so the lemma is true for $p = 2$.

Assume that the lemma is true for $p = m$, and consider $p = m + 1$. Let $w_{m+1} = (1-\beta)c$ and $\sum_{i=1}^{m} w_i = \beta c$. Then $\prod_{i=1}^{m} w_i$ is maximized at $w_i = \beta c/m$. Hence, $\prod_{i=1}^{m+1} w_i$ can be rewritten as

$$\prod_{i=1}^{m+1} w_i = \left(\prod_{i=1}^{m} w_i\right) \cdot w_{m+1} = \left(\frac{\beta c}{m}\right)^m (1-\beta)c. \tag{A5}$$

The RHS of Eq. (A5) is maximized at $\beta = m/(m+1)$, which yields $w_i = c/(m+1)$ for $i = 1, \ldots, m+1$. Therefore, the lemma is also true for $p = m+1$. ■

THEOREM A1. *Let the total number of y_i's uniformly distributed in the unsearched range $[0, u)$ be n, and the boundaries of the ith window be $[w_{i-1}, w_i)$, $i = 1, \ldots, t$. Then the probability that all t of the y_i's are isolated in one contention step is maximized if $w_i - w_{i-1} = u/n$, $i = 1, \ldots, t$.*

Proof. In general, the joint probability density function of the ordered statistics, $y_1, \ldots, y_{t+1}$, is

$$f_{y_1 \cdots y_{t+1}}(x_1, \ldots, x_{t+1}) = \frac{n!}{(n-t-1)!} g(x_1) \cdots g(x_{t+1})(1 - G(x_{t+1}))^{n-t-1} \tag{A6}$$

where $g(x)$ and $G(x)$ are the probability density function and distribution function of the parent distribution, respectively. For a uniform distribution, we have $g(x) = 1$ and $G(x) = x$. So

$$f_{y_1 \cdots y_{t+1}}(x_1, \ldots, x_{t+1}) = \frac{n!}{(n-t-1)!}(1 - x_{t+1})^{n-t-1}. \tag{A7}$$

A selection is successful when $y_i \in [w_{i-1}, w_i)$, $i = 1, \ldots, t$, and $y_{t+1} \in [w_t, 1)$, implying that $y_{t+2}, \ldots, y_n \in [w_t, 1)$. The success probability can be expressed as

$$\begin{aligned}
&(\Pr\{y_i \in [w_{i-1}, w_i),\ i = 1, \ldots, t, \text{ and } y_{t+1} \in [w_t, 1)\} \\
&\quad = \int_0^{w_1} \cdots \int_{w_{t-1}}^{w_t} \frac{n!}{(n-t-1)!}(1 - x_{t+1})^{n-t-1} dx_{t+1} \cdots dx_1 \qquad \text{(A8)} \\
&\quad = \frac{n!}{(n-t)!}(1 - w_t)^{n-t} \prod_{i=1}^{t} (w_i - w_{i-1}).
\end{aligned}$$

Note that the value of $\prod_{i=1}^{t} (w_i - w_{i-1})$ depends on w_i and the partitioning of the interval $[0, w_t)$. It follows from Lemma A1 that $\prod_{i=1}^{t} (w_i - w_{i-1})$ is maximized when $w_i - w_{i-1} = w_t/t$. Substituting this result into Eq. (A8) yields

$$\Pr\{y_i \in [w_{i-1}, w_i),\ i = 1, \ldots, t, \text{ and } y_{t+1} \in [w_t, 1)\}$$
$$= \frac{n!}{(n-t)!}(1 - w_t)^{n-t}\left(\frac{w_t}{t}\right)^t. \quad \text{(A9)}$$

The RHS of Eq. (A9) is maximized at $w_t = t/n$, hence $w_i = i/n$. Last, the w_i's must be adjusted by a factor of u if the random variates are distributed in $[0, u)$, and $w_i - w_{i-1} = u/n$. ■

REFERENCES

1. Baron, R. V., *et al.* MACH kernel interface manual. Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, Jan. 1987.
2. Baumgartner, K. M., and Wah, B. W. A global load balancing strategy for a distributed computer system. *Proc. Workshop on Future Trends in Distributed Computer Systems in the 1990s,* IEEE, Hong Kong, Sept. 1988, pp. 93–102.
3. Bryant, R. M., and Finkel, R. A. A stable distributed scheduling algorithm. *Proc. 1st International Conference on Distributed Computing Systems,* 1981, pp. 314–323.
4. Cheriton, D. R. The V distributed system. Department of Computer Science, Stanford University, Stanford, CA, Mar. 1987.
5. Chow, Y. C., and Kohler, W. Models for dynamic load balancing in a heterogeneous multiple processor system. *IEEE Trans. Comput.* **C-28,** 5 (May 1979), 334–361.
6. Fioretti, A., and Rocchini, R. A. Design aspects of concurrent optical networks. *Proc. Workshop on Future Trends in Distributed Computing Systems in the 1990s,* IEEE, Hong Kong, Sept. 1988, pp. 57–64.
7. Gold, Y. I., and Franta, W. R. An efficient collision-free protocol for prioritized access-control of cable radio channels. *Comput. Networks* **7** (1983), 83–98.
8. Hwang, K., *et al.* A UNIX-based local computer network with load balancing. *IEEE Trans. Comput.* (Apr. 1982), 55–64.
9. Juang, J. Y., and Wah, B. W. Unified window protocols for local multiaccess networks. *Proc. INFOCOM,* IEEE, Apr. 1984, pp. 97–104.
10. Juang, J. Y., Resource allocation in computer networks. Ph.D. thesis, School of Electrical Engineering, Purdue University, West Lafayette, IN, Aug. 1985.
11. Juang, J. Y., and Wah, B. W. Global state identification for load balancing in a computer system with multiple contention busses. *Proc. Computer Software and Applications Conference,* IEEE, Oct. 1986, pp. 36–42.
12. Keller, R. M., Lin, F. C. H., and Tanaka, J. Rediflow multiprocessing. *Proc. COMPCON Spring,* IEEE, 1984, pp. 410–417.
13. Krueger, P., and Livny, M. Load balancing, load sharing and performance in distributed systems. Tech. Rep. No. 700, Department of Computer Science, University of Wisconsin, Madison, Aug. 1987.
14. Livny, M., and Melman, M. Load balancing in homogeneous broadcast distributed systems. *Proc. Modeling and Performance Evaluation of Computer Systems,* ACM SIGMETRICS, 1982, pp. 47–55.
15. Metcalfe, R., and Boggs, D. Ethernet: Distributed packet switching for local computer networks. *Comm. ACM* **19,** 7 (1976), 395–404.
16. Ni, L. M., and Li, X. Prioritizing packet transmission in local multiaccess networks. *Proc. 8th Data Communications Symposium,* IEEE, 1983.

17. Ni, L. M., Xu, C.-M., and Gendreau, T. B. A distributed drafting algorithm for load balancing. *IEEE Trans. Software Engrg.* **SE-11,** 10 (Oct. 1985), 1153–1161.
18. Shacham, N., A protocol for preferred access in packet-switching radio networks. *IEEE Trans. Commun.* **COM-31,** 2 (Feb. 1983), 253–264.
19. Shock, J. F., *et al.* Evolution of the Ethernet local computer network. *IEEE Trans. Comput.* **15,** 8 (Aug. 1982), 10–27.
20. Stallings, W. *Local Networks: An Introduction.* Macmillan, New York, 1984.
21. Stone, H. S. Critical load factors in two-processor distributed systems. *IEEE Trans. Software Engrg.* **SE-4,** (May 1978), 254–259.
22. Tanenbaum, A. S. *Computer Networks.* Prentice–Hall, Englewood Cliffs, NJ, 1981.
23. Tobagi, F. A. Carrier sense multiple access with message-based priority functions. *IEEE Trans. Commun.* **COM-30,** 1 (Jan. 1982).
24. Wah, B. W., and Juang, J. Y. An efficient protocol for load balancing on CSMA/CD networks. *Proc. 8th Conference on Local Computer Networks,* IEEE, Oct. 1983, pp. 55–61.
25. Wah, B. W., and Juang, J. Y. Resource scheduling for local computer systems with a multiaccess network. *IEEE Trans. Comput.* **C-34,** 12 (Dec. 1985), 1144–1157.
26. Walker, B., Popek, G., English, R., Kline, C., and Thiel, G. The LOCUS distributed operating system. *Proc. Ninth Symposium on Operating System Principles,* ACM, 1983, pp. 49–70.

JIE-YONG JUANG received the B.S. degree in electrical engineering from National Taiwan University, Taipei, Taiwan, in 1976, the M.S. degree in computer science from the University of Nebraska, Lincoln, in 1981, and the Ph.D. degree in electrical engineering from Purdue University, West Lafayette, Indiana, in 1985. He is an assistant professor in the Department of Electrical Engineering and Computer Science at Northwestern University, Evanston, Illinois. His areas of research include computer architecture, parallel processing, distributed processing, fault-tolerant computing, and logic programming.

BENJAMIN W. WAH received the Ph.D. degree in computer science from the University of California, Berkeley, in 1979. He was on the faculty of the School of Electrical Engineering at Purdue University, West Lafayette, Indiana, between 1979 and 1985. He is now a professor in the Department of Electrical and Computer Engineering and the Coordinated Science Laboratory of the University of Illinois at Urbana–Champaign, Urbana. Between 1988 and 1989, he served as a program director of the Microelectronic Systems Architecture Program, National Science Foundation. He is selected as a University Scholar of the University of Illinois in 1989. His areas of research include computer architecture, parallel processing, artificial intelligence, distributed database, and computer networks. Dr. Wah is associate editor-in-chief of the *IEEE Transactions on Knowledge and Data Engineering,* an area editor of the *Journal of Parallel and Distributed Computing,* and editor of *Information Science.* He serves as a member of the Governing Board of the IEEE Computer Society, a program evaluator for ABET (computer engineering) and CSAC (computer science), and the program chairman of the 1990 International Conference on Parallel Processing.