An All-sharing Load-balancing Scheme on the CSMA/CD Network and Its Analysis[1]

YING HAO*, JYH-CHARN S. LIU† AND JUNGUK L. KIM‡

*Global Finance Architecture and Emerging Technology, Citicorp, 4 Campus Circle, Westlake, TX 76262, USA
†Department of Computer Science, Texas A&M University, College Station, TX 77843-3112, USA
‡Bellcore, 444 Hoes Lane, Piscataway, NJ 08854, USA

This paper analyzes a load-balancing scheme, called All-Sharing Load Balancing (ASLB), which evenly distributes the load of the system among all nodes on a CSMA/CD local area network in a collision-free manner for each load-balancing activity. A novel analytical model is presented to evaluate performance of the ASLB scheme and its effect on the normal communication message transmissions and vice versa. The evaluation also takes into account the tradeoff between system response time and message delay. Our analysis shows that ASLB can achieve good performance in both system response time and message transmission delay. Our analysis techniques are applicable to various distributed algorithms involved with interprocess coordination.

Received October 15 1993, revised September 14 1994

1. INTRODUCTION

Load balancing is a useful technique to improve the performance of a distributed system. The main objective of load balancing is to distribute workload among system nodes evenly to reduce the probability that some nodes are idle while others are heavily loaded (Livny and Melman, 1982; Zhou, 1988; Baumgarter and Wah, 1989; Rommel, 1991). Simulation studies show that a load balancing scheme can improve the response time 30–60% under moderate load (Zhou, 1988). A load-balancing scheme can be *static* or *dynamic*. In the static approach, job migration decisions are made based on *a priori* knowledge on the expected average workload of each node (Tantawi and Towsley, 1985; Bonomi and Kumar, 1988; Kurose and Simha, 1986) and hence it is more suitable for systems with known, steady workload. In the dynamic (state-dependent) approach, however, job migration decisions are made based on the current state of the system (Baumgarter and Wah, 1989; Eager *et al.*, 1986; Wang and Morris, 1985; Stankovic, 1985; Casavant and Kuhl, 1986; Lin and Keller, 1986; Pulidas *et al.*, 1988). Since the dynamic load balancing approach can capture the rapid system state changes, it is more suitable for systems with dynamic workload. For its effectiveness, we focus on dynamic load-balancing in this paper.

A dynamic load balancing scheme generally consists of several steps: state information collection, state/load information exchange and workload distribution. Load information of the system can be collected in a centralized or distributed manner. In the centralized approach, a designated node monitors the system workload to make decisions on job distribution (Bonomi and Kumar, 1988; Ni and Hwang, 1981; Chow and Kohler, 1979; Chow, 1982). In the distributed approach, each node collects the system load information and decides independently with which node it shares the workload (Baumgarter and Wah, 1989; Eager *et al.*, 1986a; Stankovic, 1985; Lin and Keller, 1986; Pulidas *et al.*, 1988). A load-balancing activity can be initiated by a node which will receive jobs (a receiver) or by a node which will send jobs (a sender) in the activity (Wang and Morris, 1985). It has been observed that the sender-initiated approach performs better when the system is lightly loaded, but the receiver-initiated approach performs better under heavily loaded situations (Eager *et al.*, 1986b; Hac, 1989). The load can be shared between two nodes (the *pair-sharing* approach) or by all nodes in a load-balancing activity. Most load sharing schemes in the literature employ the pair-sharing approach, though certain cooperations among nodes are utilized in some schemes to improve the system performance (Shirarakri and Krueger, 1990). A good methodological study on load sharing schemes is given by Kremien and Kramer (1992), who suggested that (1) node cooperation is necessary for efficient job transfers and (2) remote job execution should be restricted to a small proportion of the system for pair-sharing schemes. Most of the above discussions and their conclusions are for general distributed systems. When the distributed system is based on a CSMA/CD bus network, we need to pay special attention to the impact of network contention. Since a pair-sharing approach may need to send several probing messages for one job migration, the contention of message transmission could severely degrade the system performance. System performance degradation is further aggregated when load-balancing messages and normal communication messages interfere with each other.

[1] This work is done when the authors were with the Texas A&M University.

CSMA/CD is one of the most simple, reliable and efficient media access standards for local area networks. Although its current transmission speed, i.e. 10 Mbps, is relatively lower than fiber distributed data interface (FDDI) and asynchronous transfer mode (ATM) networks, a new generation of CSMA/CD standards at the speed of 100 Mbps are emerging (Roberts, 1993). The CSMA/CD protocol is most efficient for message broadcasting. By efficiently utilizing the broadcasting capability of the CSMA/CD bus, we can minimize the performance impact of communication delay on load-balancing activities. Based on this observation, we proposed the *All-Sharing Load Balancing (ASLB)* scheme to effectively utilize the broadcast capacity of the CSMA/CD bus network (Kim *et al.*, 1992). In this scheme, nodes compete to seize the communication media to initiate load-balancing activity based on the CSMA/CD protocol. Once an initiation message is successfully broadcast, all nodes exchange their state information and schedule job migration cooperatively. Communication collision during state information exchange and also during job migration is avoided by restricting nodes to transmit their messages in accordance with their assigned logical addresses. Logical addresses of the nodes are assigned based on a logical ring order at system initialization and they need to be re-negotiated whenever a node joins or departs from the system. We will only study job migration without considering process migration in ASLB, because it has been shown that the marginal performance gain of process migration does not justify its implementation cost (Eager *et al.*, 1988).

In ASLB, the system workload is distributed nearly evenly among all nodes after each load-balancing activity. Moreover, since load-balancing messages are transmitted in a contention-free manner, interference of the network traffic during a load-balancing activity is minimized. Since a portion of the network bandwidth is used by load-balancing activities, the normal message traffic may experience longer delay and hence it is possible that such delay may overshadow the system performance improvement brought about by load-balancing. In this paper, we study the effect of ASLB load balancing activities on the normal messages, and vice versa, through analysis and simulation. Assuming that the network is shared by both normal communication messages and load-balancing messages, the performance of the ASLB is evaluated for both system response time and message delay. Analytical and simulation results show that the system response time under ASLB is insensitive to network traffic loads, while the system response time under pair-sharing schemes increases sharply with the network utilization. We also show that the average communication delay under ASLB can be controlled without compromising the system response time.

The rest of this paper is organized as follows. Section 2 describes the ASLB in detail. In Section 3, the analytical model for the ASLB scheme is presented. We discuss the results of our performance analysis and simulations in Section 4, and Section 5 concludes this paper.

2. SYSTEM MODEL AND ASLB SCHEME

2.1. The system model

In our system model, m homogeneous nodes are connected through a bus network whose access is controlled by the CSMA/CD protocol. Jobs are assumed to be independent of each other and they can be executed either locally or migrated to other nodes for execution. Normal message communication is assumed to be asynchronous, i.e. job execution is not blocked when a node sends or receives messages. At any time instant, only one job can be executed in a node. For simplicity, we define the load of a node as the number of jobs being queued or executed, which is known to be a good workload metric for design of load balancing schemes (Ferrari and Zhou, 1986; Kunz, 1991).

We assume that failed components can be detected by low level fault detection mechanisms, such as watchdog timers, and the cyclic redundancy checking (CRC) codes in communication protocols. When faults are detected, any pending ASLB activities are aborted and failure recovery routines will be invoked to reconfigure non-fautly nodes, and normal activities will be resumed after the system reconfiguration is completed. During the system reconfiguration, a distributed diagnosis algorithm such as the well-known PMC algorithm (Preperata *et al.*, 1967) can be invoked to send the reconfiguration message to every fault-free node. During system reconfiguration, new logical addresses may be assigned to nodes so that their existing ordering relationship is reserved, but faulty nodes are excluded. For example, if the node with the logical address 5 becomes faulty, then nodes with logical addresses 4, 6 and 7 will be re-ordered as 4, 5 and 6, respectively. A faulty node cannot participate any future load-balancing activity until it recovers from the current fault. When the system is reconfigured, a recovered node will be assigned the largest logical address. On a CSMA/CD network, the performance impact of failed components and the associated recovery overheads can be reflected in our analytical model by adjustment of communication delays. Therefore, without loss of generality, we assume that all components are fault-free and the bus network is reliable in the subsequent discussion.

We assume that each node has one computing processor and a communication co-processor, where the computing processor is responsible for all the computation and job scheduling, and the communication co-processor handles sending and receiving of messages in response to the computing processor's commands. The computing and communication processors communicate with each other through a shared memory, which is used by the two processors to exchange messages, state variables, and communication commands. Pointers are passed between the processors to indicate locations of the different types of information. The communication commands are linked by the pointers and thus the computing processor can arbitrarily alter the execution sequence of outstanding commands by modifying the pointers (e.g. see Intel, 1990). The two processors can get each other's immediate attention through interrupt signals. For example, the communication co-processor can interrupt the computing processor when it receives a new message, so that the inbound message can be processed immediately. In the mean time, the computing processor can abort the current message transmission by altering points of commands and by interrupting the communication co-processor. With this system architecture, the computing processor need not be blocked from computation while its communication co-processor is transmitting or receiving messages.

2.2. The ASLB protocol

Each time a job is completed, the node executing the job checks its job-waiting queue: if the queue is empty, the node initiates load-balancing by broadcasting a load-balancing initiation message. The load-balancing initiation message may collide with other messages including other load-balancing initiation messages. However, after the first load-balancing initiation message goes through the bus, the computing processors interact with their communication processors to avoid any other message collisions until the load-balancing is completed. If a new job arrives at the initiator before the initiation message has been sent out, the initiator attempts to cancel its initiation message. If the message is cancelled, the node resumes its job execution.

Upon receiving the load-balancing initiation message, each node purges its *active message* (if any) which is the message in either backoff or transmission state. A purged active message will be placed back into the *waiting queue*, which holds messages in waiting state. After purging its active message, the node then waits for its turn to broadcast its load information. The node which has a logical address next to the initiator becomes the *load information exchange organizer*, i.e. this node is the first node to broadcast its load message. After the node with logical address i broadcasts it load, node $(i\, MOD\, m) + 1$ will broadcast its load information. After receiving load information of all nodes, a job migration plan is calculated by each node using algorithm FP (shown shortly) and migration is then done following the order calculated by FP.

When a node begins to purge its active message, at most one message may have been transmitted on the bus in the worst case. We show that only one such message can be transmitted in the worst case before all nodes can successfully purge their active messages, if any. Hence, collision-free load information exchange can begin after at most one message transmission.

Referring to Figure 1, we denote the time instant at which a load-balancing initiator finished transmission of an initiation message by t_b, the time instant at which all nodes receive the initiation message by t_o, the time instant at which a node N_i purges its active message by t_p and the propagation delay of the bus by τ. Let T_r be the time interval between t_o and t_p which is the time a node spends on the purge operation. Let M_i denote the message which becomes active in N_i between t_b and t_o, and T_{mesg} denote the time taken to transmit M_i. The worst case is that M_i seizes the bus immediately after t_0. When $T_r < T_{mesg}$, as shown in Figure 1(a), M_i can be successfully purged, and when $T_r \geqslant T_{mesg}$, as shown in Figure 1(b), M_i cannot be purged.

All nodes can purge their active messages before $t_b + \tau + T_{mesg}$, i.e. $T_r < T_{mesg}$, based on the following reasoning. A typical 10 MHz local area network has the shortest message length of 0.057 ms (Schwartz, 1987); this is equivalent to the time to execute 570 instructions by a processor of 10 M *FLOPS*. To recognize a load-

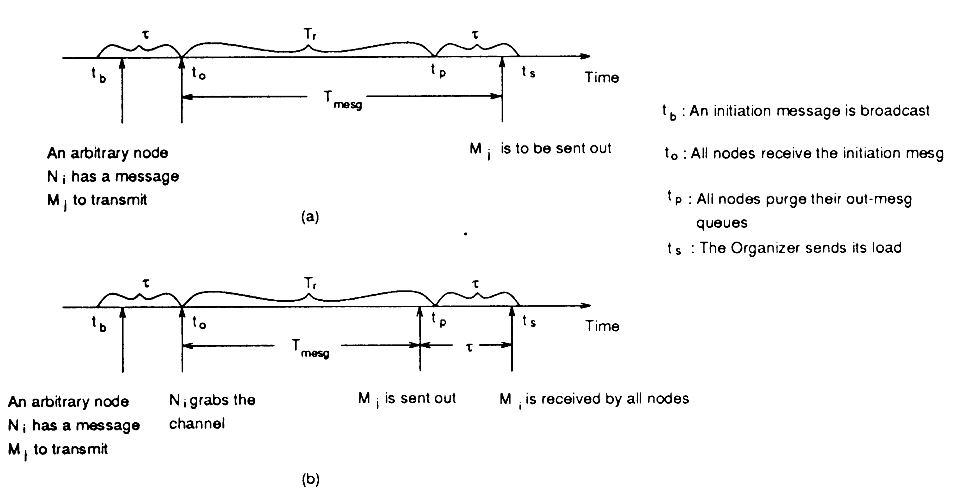
τ
T_r
τ
t_b
t_o
t_p
t_s
Time
T_mesg
An arbitrary node N_i has a message M_i to transmit
M_i is to be sent out
(a)
t_b : An initiation message is broadcast
t_o : All nodes receive the initiation mesg
t_p : All nodes purge their out-mesg queues
t_s : The Organizer sends its load
τ
T_r
τ
t_b
t_o
t_p
t_s
Time
T_mesg
τ
An arbitrary node N_i has a message M_i to transmit
N_i grabs the channel
M_i is sent out
M_i is received by all nodes
(b)

FIGURE 1. The purging process of an active message when load-balancing is initiated, where (a) message M_i can be purged and (b) M_i cannot be purged in time.

balancing initiation message, a node needs to transfer the message from its communication processor to main memory, and then let the computing processor interpret the message and purge an active message by altering the pointer to the active message. Hence T_r can be several micro-seconds long: $T_r < T_{mesg}$, and we can guarantee that all nodes can purge their active messages before $t_b + \tau + T_{mesg}$. To cope with the worst situation that one active message may get through, the load-information exchange organizer needs thus to wait for $T_u = 2\tau$ after it purges its own active message, before it passes its load message to its communication processor. The first τ guarantees that all nodes in the system receive the load-balancing message and the remaining τ guarantees that the unpurged message, if any, reaches its destination. Then, it is up to the communication processor to transmit the load message. The transmission sequence is known to all nodes since they are listening to the communication medium.

After the load information of the system is delivered to all nodes, each node computes the global load-balancing plan by using the FP algorithm given below. The algorithm produces a sequence of source-destination pairs and the number of jobs migrating between them. The nodes then send out their jobs based on their order to avoid communication collisions. To execute FP, each node maintains a *Load Table*, *TBL*, and each entry of the *TBL* has two entries representing the job number and the logical address of the node, respectively. The table has m entries, where m is the number of nodes. Let the number of jobs in node N_i be denoted as L_i $(i = 1, \ldots, m)$ and let $AVG = [\sum_{i=1}^{m} L_i/m]$, where AVG is an integer value. If N_i has extra jobs, i.e. $L_i - AVG > 0$, it will find N_j with load less than the average, i.e. $L_j - AVG < 0$, and share the jobs with N_j. If $L_i + L_j - 2 * AVG > 0$, then N_i looks for another node for further sharing; that is, N_i finds multiple nodes to share its jobs until $L_i \leqslant AVG$. Therefore, after a load-balancing activity, the number of jobs at each node is nearly the same as AVG. The FP algorithm is formally described as follows:

Algorithm FP:

```
$AVG = [(TBL(1).load + TBL(2).load + \cdots + TBL(m).load)/m]$
Sort(TBL) in the decreased sequence of load
if(|TBL(1) - TBL(m)| ≤ 1)
    return; /* The system is balanced* /
if(AVG = 0) AVG = 1 /* Avoid transmitting process being executed */
$TBL(i).load = TBL(i).load - AVG$, for $i = 1, \ldots, m$
i := 1;
j := m;
while(i < j) do begin
    cond := TBL(i) + TBL(j);
    if(cond = 0) then begin
        store migration (TBL(i).addr → TBL(j).addr,
            TBL(i).load); /*TBL(i).load jobs to be migrated
            from TBL(i).addr to TBL(j).addr */
        i := i + 1;
        j := j - 1
    end
    else if (cond > 0) then begin
        store migration (TBL(i).addr → TBL(j).addr,
        -TBL(j).load);
        TBL(i).load = TBL(i).load + TBL(j).load;
        j := j - 1
    end
    else if (cond < 0) then begin
        store migration (TBL(i).addr → TBL(j).addr,
        TBL(i).load);
        TBL(j).load := cond;
        i := i + 1
    end
end;
```

A pseudo-code of the ASLB scheme can be found in Appendix B.

From the description of FP, it is evident that the load difference between any two nodes is at most one after one round of load-balancing activity. Since a load-balancing message has to compete with other messages for the communication bus, the traffic intensity on the bus determines the delay of a load-balancing initiation message and hence may affect the performance of ASLB. Once a balancing initiation message gets through, the network is dedicated to the load-balancing activities and normal messages get blocked from transmission until the load-balancing activities complete. Hence, frequent load-balancing initiation may cause increase in normal message delays, and it is thus important to strike a balance between the system response time and the normal message delay. As will be shown in the Section 4, ASLB can achieve low system response time at a reasonable message delay by proper modification of load-balancing initiation conditions.

3. ANALYTICAL PERFORMANCE ANALYSIS

A commonly used performance criterion for evaluation of load-balancing schemes is the average system response time, which is the average time elapsed from the arrival of a job to its completion either locally or at a remote site. Since the load-balancing activities in a distributed system may consume a non-negligible portion of network bandwidth, the communication delay of normal messages may increase. Therefore, the effect of load-balancing activities on normal message communication should also be taken into account in evaluation of a load-balancing scheme. Therefore, we use the average normal message delay as well as the system response time as the performance criteria for evaluation of the ASLB scheme.

A common way to analyze the performance of a dynamic load-balancing scheme is through a multi-dimensional Markov chain. However, this approach is very inefficient, if not impossible, in the analysis of the ASLB scheme for the following reasons. First, since nodes under ASLB scheme balance their workload in a

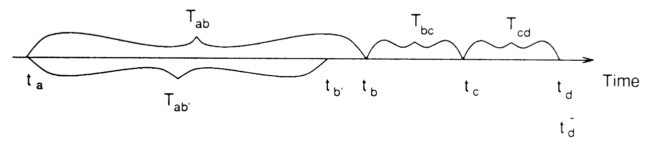

FIGURE 2. The cyclic behavior of nodes under the ASLB scheme.

fully cooperative manner, it is virtually impossible to enumerate all the possible system states to calculate transition rates. Secondly, in a Markov-chain-based model, job migration is approximated by state transitions, making it impossible to study the impact of communication delay on the system performance. In our approach, we first decompose the time domain into different phases and then analyze the behavior of the system in each of the phases. By proper integration of boundary conditions between phases, we obtain the overall system performance measurement. In our analysis, we assume that the interarrival time and the execution time of jobs at each node follow exponential distributions with job arrival rate λ and mean execution time $1/\mu$, respectively, where $\lambda < \mu$.

Under the ASLB scheme, the behavior of a node can be modeled as a renewal process, where one of the renewal cycles is shown in Figure 2. The cycle begins at time instant t_a, at which the previous load-balancing activity has just completed, and ends at instant t_d, at which the load-balancing activity initiated at t_a completes. It should hence be noted that t_d is the beginning instant of the next renewal cycle, i.e. the system state at t_d should be statistically equivalent to the system state at t_a. We denote the time instant immediately before t_d as t_{d^-}, meaning that the current load-balancing is about to complete. Starting from t_a, jobs are executed in each node independently until a load-balancing initiation message is received by all nodes at time instant t_b. $[t_a, t_b)$ is thus called the *independent job-execution phase*. More specifically, one of the nodes becomes idle at time $t_{b'}$, $t_{b'} \in [t_a, t_b)$, and broadcasts a load-balancing initiation message, which reaches all nodes at t_b after a network transmission delay of $T_{b'b} = t_b - t_{b'}$. The *load-balancing decision phase* starts at t_b and ends at t_c, during which the nodes exchange the load information, pack jobs which need to be migrated, and send packed jobs to communication processors. The *job-migration phase* starts at t_c and finishes at t_d, i.e. nodes begin to transmit their jobs which need to be migrated, if any, and all the migrated jobs will have reached their destinations at t_d, after which a new round of load-balancing activity begins.

By defining the state of a node as the number of jobs in that node, and the system state as the total number of jobs in the system, we derive the state probability distribution of an arbitrary node at any time instant in the three phases, from which we derive the average system response time and the average message delay. Let the probability distributions of the lengths of the three phases be denoted as $F_{T_{ab}}(t)$, $F_{T_{bc}}(t)$ and $F_{T_{cd^-}}(t)$, respectively. Also, the probability that n jobs are in the node within the three phases are denoted respectively as $p_n(t)$ $(t \in [t_a, t_b))$, $q_n(t)$ $(t \in [t_b, t_c))$, $\omega_n(t)$ $(t \in [t_c, t_{d^-}))$, respectively. The average number of jobs in the node can thus be expressed as:

$$\bar{L} = \frac{\sum_{n=0}^{\infty} n\left(\int_0^{\infty}\int_{t_a}^{t_a+t} p_n(\delta \mid t)\, d\delta\, dF_{T_{ab}}(t) + \int_0^{\infty}\int_{t_b}^{t_b+t} q_n(\delta \mid t)\, d\delta\, dF_{T_{bc}}(t) + \int_0^{\infty}\int_{t_c}^{t_c+t} w_n(\delta \mid t)\, d\delta\, dF_{T_{cd^-}}(t)\right)}{\int_0^{\infty} t\, dF_{T_{ab}}(t) + \int_0^{\infty} t\, dF_{T_{bc}}(t) + \int_0^{\infty} t\, dF_{T_{cd^-}}(t)} \tag{1}$$

Once we get $\bar{L}$, we can calculate the system response time using the Little's Law (Little, 1961): $\bar{R} = \bar{L}/\lambda$.

To derive $\bar{L}$, we ought to find probability distribution functions $F_{T_{ab}}(t)$, $F_{T_{bc}}(t)$, $F_{T_{cd^-}}(t)$ and the mass distribution functions $p_n(t)$ $(t \in [t_a, t_b))$, $q_n(t)$ $(t \in [t_b, t_c))$ and $w_n(t)$ $(t \in [t_c, t_{d^-}))$. However, it is difficult to derive $F_{T_{ab}}$, $F_{T_{bc}}$ and $F_{T_{cd^-}}$ directly for the following reasons. Firstly, the distribution of message delay $T_{b'b}$ on the CSMA/CD bus is known to be very difficult to derive. Secondly, the inter-dependency between T_{ab}, T_{bc} and T_{cd^-} makes derivation of their distributions mathematically intractable. For simplicity, we directly use mean values of T_{ab}, T_{bc} and T_{cd^-} instead of their distribution functions for our analysis, where $\bar{T}_{ab} = \int_0^{\infty} t dF_{T_{ab}}(t)$, $\bar{T}_{bc} = \int_0^{\infty} t dF_{T_{bc}}(t)$ and $\bar{T}_{cd\_} = \int_0^{\infty} t\, dF_{T_{cd^-}}(t)$, respectively. Equation (1) is thus simplified into

$$\bar{L} = \frac{\sum_{n=0}^{\infty} n\left(\int_{t_a}^{t_b} p_n(t \mid \bar{T}_{ab})\, dt + \int_{t_b}^{t_c} q_n(t \mid \bar{T}_{bc})\, dt + \int_{t_c}^{t_{d^-}} w_n(t \mid \bar{T}_{cd^-})\, dt\right)}{\bar{T}_{ab} + \bar{T}_{bc} + \bar{T}_{cd^-}} \tag{2}$$

after replacing δ by t, where $t_b = t_a + \bar{T}_{ab}$, $t_c = t_b + \bar{T}_{bc}$ and $t_d = t_c + \bar{T}_{cd}$.[2]

Since system states at t_a and t_b are statistically identical at the steady-state, we solve the problem recursively in the following manner. Assuming that the probability of the system t_a is P_ℓ, $\ell = 0, 1, \cdots$ (i.e. there are totally ℓ jobs in all the nodes), we derive $\bar{T}_{ab}(\mid \ell)$ and $p_n(t \mid \ell)$

[2] For typographical simplicity, $p_n(t \mid \bar{T}_{ab})$, $q_n(t \mid \bar{T}_{bc})$ and $w_n(t \mid \bar{T}_{cd^-})$ will be written as $p_n(t)$, $q_n(t)$ and $w_n(t)$, respectively.

$(t_a \leqslant t < t_b)$ for the given system state ℓ at t_a. Then, we derive $\bar{T}_{bc}(\,|\,\ell)$ and $q_n(t\,|\,\ell)$ $(t_b \leqslant t < t_c)$ for the given $p_n(t_b\,|\,\ell)$. $T_{cd^-}(\,|\,\ell)$ and $w_n(t\,|\,\ell)$ $(t_c \leqslant t < t_d^-)$ are consequently derived using $q_n(t_c\,|\,\ell)$. Once we obtain the state probability distribution of each node at t_d^-, $w_n(t_d^-\,|\,\ell)$, we can derive the probability of the system state at t_d, denoted as P_n $(n = 0, 1\cdots,)$. The steady-state probability P_ℓ can thus be obtained by utilizing $P_\ell \equiv P_n$, $\forall \ell = n$. After obtaining P_ℓ, we can thus derive $p_n(t)$ $(t_a \leqslant t < t_b)$, $q_n(t)$ $(t_b \leqslant t < t_c)$ and $w_n(t)$ $(t_c \leqslant t < t_d^-)$, and eventually the system response time.

In order to derive $p_n(t\,|\,\ell)$, we need to derive first the conditional probability distribution function of $T_{ab'}$ given that ℓ jobs are in the system at t_a, $F_{T_{ab'}}(t\,|\,\ell)$, and its mean value $\bar{T}_{ab'}(\,|\,\ell)$. Note that $T_{ab'}$ is the time elapse for one of the nodes, which is initially in state k, to become idle. Let T_i denote the time elapse for node N_i $(i = 1, 2, \cdots, m)$ to become idle starting at t_a, it is evident that $T_{ab'} = \min\{T_1, T_2, \cdots, T_m\}$. Assume that the system is in state ℓ at t_a, and let $r = \ell \, mod \, m$ and $n_I = \lfloor \ell/m \rfloor$. Since at t_a the load difference between nodes is at most one under ASLB, nodes can be divided into two groups. One group consists of r nodes, each of which has $n_I + 1$ jobs and the other group consists of $m - r$ nodes, each of which has n_I jobs. Since T_i's have the identical probability distributions if they have the same number of jobs at t_a, we denote $f_T(t\,|\,k)$ and $F_T(t\,|\,k)$ as the density function and probability distribution function of T_i, given that N_i has k jobs at t_a, where $k = n_I$ or $k = n_I + 1$. Therefore,

$$F_{T_{ab'}}(t\,|\,\ell) = 1 - (1 - F_T(t\,|\,n_I + 1))^r (1 - F_T(t\,|\,n_I))^{m-r}. \tag{3}$$

The mean value of the interval $T_{ab'}$ can thus be calculated as

$$\bar{T}_{ab'}(\,|\,\ell) = \int_0^\infty t \, dF_{T_{ab'}}(t\,|\,\ell). \tag{4}$$

We derive $f_T(t\,|\,k)$ in two cases: $k \neq 0$ and $k = 0$. In the first case, T is the time elapse for a node, which initially has k jobs, to become idle. In the second case, T is the sum of two time intervals T_A and T_B, that is, $T = T_A + T_B$. T_A is the time interval between t_a and a time instant at which a new job just arrives at the node, and the node becomes idle again right after the interval T_B. The state transition diagrams for calculating T_i corresponding to the two cases are plotted in Figure 3(a) and (b), respectively. Note that no transition occurs from S_0 to S_1 because the independent job-execution phase terminates when the node reaches state S_0. The dotted line indicates that a new job arrives at the node which is originally idle $(k = 0)$, and the state is depicted by S_{00} in the diagram.

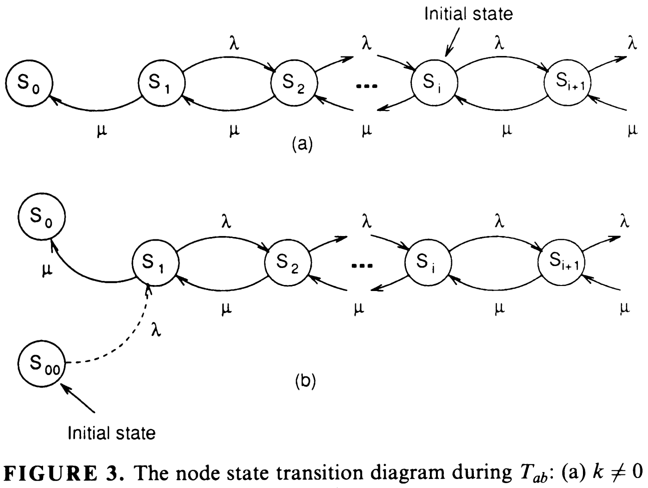

FIGURE 3. The node state transition diagram during T_{ab}: (a) $k \neq 0$ and (b) $k = 0$.

When $k \neq 0$, the differential-difference equations governing the node state can be derived from Figure 3 as

$$\begin{cases} p_0'(t) = \mu p_1(t) \\ p_1'(t) = -(\lambda + \mu) p_1(t) + \mu p_2(t) \\ p_n'(t) = \lambda p_{n-1}(t) - (\lambda + \mu) p_n(t) + \mu p_{n+1}(t) \quad (n > 1). \end{cases} \tag{5}$$

Solving the above equations with the initial condition $p_k(0) = 1$ and $p_j(0) = 0$ $(j \neq k)$, we get:

$$f_T(t\,|\,k) = p_0'(t) = \frac{k\left(\dfrac{\lambda}{\mu}\right)^{-k/2} I_k(2\sqrt{\lambda\mu t}) \mathrm{e}^{-(\lambda+\mu)t}}{t}, \tag{6}$$

where

$$I_n(y) = \sum_{i=0}^{\infty} \frac{(y/2)^{n+2i}}{i!(n+i)!} \quad (n \geqslant 0)$$

is a modified Bessel function with the property of $I_{-n}(y) = I_n(y)$. Detailed derivation of equation (6) can be found in Appendix C.

When $k = 0$, $f_T(t\,|\,0)$ is equal to the convolution of $f_{T_A}(t)$ and $f_{T_B}(t)$ because $T = T_A + T_B$. Since the density functions of T_A and T_B are $f_{T_A}(t) = \lambda e^{-\lambda t}$ and $f_{T_B}(t) = f_T(t\,|\,1)$, respectively, we get $f_T(t\,|\,0) = \lambda e^{-\lambda t} * f_T(t\,|\,1)$, where $*$ denotes the convolution operator.

By plugging $f_T(t\,|\,n_I)$ and $f_T(T\,|\,n_I + 1)$ into equations (3) and (4), we get $\bar{T}_{ab'}(\,|\,\ell)$. Now, we derive the state probability distribution of the node $p_n(t\,|\,k)$, where $n = 0, 1, \cdots, \infty$ and $k = n_I$ or $k = n_I + 1$, at any time instant t in the independent job-execution phase $[t_a, t_b)$. We need to use a transient analysis approach to derive $p_n(t)$, $\forall t \in [t_a, t_b)$, which cannot be obtained through the steady state solution of M/M/1 queueing system. Since no job migration occurs during the independent-job-execution phase, each node can thus be modeled as a Markovian process which initially has k jobs at t_a. Note that since $p_n(t\,|\,k)$ is defined only for the interval $[t_a, t_b)$, we are concerned with the probability of n jobs in the node at a time instant $t \in [t_a, t_b)$ which initially has k jobs at t_a. The differential-difference equations governing the state of a node during independent job execution phase are:

$$\begin{cases} p_0'(t\,|\,k) = -\lambda p_0(t\,|\,k) + \mu p_1(t\,|\,k) \\ p_n'(t\,|\,k) = -(\lambda+\mu)p_n(t\,|\,k) + \lambda p_{n-1}(t\,|\,k) \\ \qquad + \mu p_{n+1}(t\,|\,k) \quad (n>0). \end{cases} \tag{7}$$

From the solution in Gross and Harris (1985), with boundary conditions that $p_k(t_a\,|\,k) = 1$ and $p_j(t_a\,|\,k) = 0$, $\forall j \neq k$, and the unification condition that

$$\sum_{i=0}^{\infty} p_i(t) = 1,$$

we obtain:

$$p_n(t\,|\,k) = e^{-(\lambda+\mu)(t-t_a)}\left[\left(\frac{\mu}{\lambda}\right)^{(k-n)/2} I_{n-k}(2\sqrt{\lambda\mu}(t-t_a)) + \left(\frac{\mu}{\lambda}\right)^{(k-n+1)/2} I_{n+k+1}(2\sqrt{\lambda\mu}(t-t_a)) + \left(1-\frac{\lambda}{\mu}\right)\left(\frac{\lambda}{\mu}\right)^{n} \sum_{l=n+k+2}^{\infty} \left(\frac{\mu}{\lambda}\right)^{l/2} \times I_l(2\sqrt{\lambda\mu}(t-t_a))\right], \tag{8}$$

where $t_a \leqslant t < t_b$. Details on derivation of the above equations are given in Appendix C. For simplicity, we have assumed that all the nodes, including the load-balancing initiator, have identical behavior as described in equation (7). Although this is not an accurate description of the system, the incurred error by this approximation is quite reasonable, as found in our simulation.

We now derive the mean value of T_{bc} and the probability that n jobs are in the node at any instant t, $q_n(t)$ when $t \in [t_b, t_c)$. The time interval T_{bc} consists of three sub-intervals. In the first sub-interval, nodes except the initiator exchange load information in a collision-free manner, which takes $(m-1)T_{load}$ time, where T_{load} denotes the transmission time of a load information message. In the second sub-interval, a node executes FP to find the load-sharing partners, and it takes T_{FP} time. The node then packs its jobs which need to be migrated to other nodes in the third sub-interval. Let $\bar{M}$ denote the average number of jobs in a node that need to be migrated to other nodes and T_{pack} denote the processing time for a node to pack one job, we get the duration of the third sub-interval as $\bar{M}T_{pack}$. Therefore,

$$\bar{T}_{bc}(\,|\,\ell) = (m-1)T_{load} + \bar{M}T_{pack} + T_{FP}. \tag{9}$$

We now calculate $\bar{M}$, which is determined by the job distribution at t_b, $p_n(t_b)$. The average number of jobs at t_b in each node can be calculated as

$$\bar{L}_b = \frac{r}{m}\sum_{n=0}^{\infty} np_n(t_b\,|\,n_I+1) + \frac{m-r}{m}\sum_{n=0}^{\infty} np_n(t_b\,|\,n+I)$$

since r nodes in the system initially have $n_I + 1$ jobs at t_a and the other $m-r$ nodes have n_I jobs. Since the node migrates its jobs only when it has more than $\bar{L}_b$ jobs, the average number of jobs migrated by the node is thus

$$\bar{M} = \sum_{n>\bar{L}_b} (n-\bar{L}_b)p_n(t_b\,|\,k),$$

when $\bar{L}_b > 1$. However, when the node has only one job, it cannot migrate this job since it is already being executed. Therefore, when $\bar{L}_b(t_b) < 1$, the average number of jobs migrated by a node is

$$\bar{M} = \sum_{n>\bar{L}_b, n\neq 1} (n-\bar{L}_b)p_n(T_b\,|\,k),$$

where $k = n_I$ or $k = n_I + 1$. Combining these two cases, we get

$$\bar{M} = \frac{r}{m}\sum_{n>\bar{L}_b, n\neq 1} (n-\bar{L}_b)p_n(t_b\,|\,n_I+1) + \frac{m-r}{m}\sum_{n>\bar{L}_b, n\neq 1} (n-\bar{L}_b)p_n(t_b\,|\,n_I). \tag{10}$$

The derived $\bar{M}$ can be plugged into equation (9) to get $\bar{T}_{bc}$.

In the time period T_{bc}, the node is dedicated to the load-balancing activity and thus only job arrivals can occur. Therefore, the number of jobs at instant t ($t \in [t_b, t_c)$), denoted as $N(t)$, is equal to the sum of the number of jobs t_b, $N(t_b)$, and any new arrivals during $[t_b, t)$, $A(t)$, i.e. $N(t) = N(t_b) + A(t)$. The probability that n jobs arrive during $[t_b, t]$, where $t \in [t_b, t_c)$, is simply $\alpha_n(t) = \lambda(t-t_a)e^{-\lambda(t-t_a)}/n!$ based on the assumption of the Poisson arrivals. Hence, the probability that n jobs are in the node at $t \in [t_b, t_c]$ is $q_n(t\,|\,k) = p_n(t_b\,|\,k) * \alpha_n(t)$, where $*$ denotes the convolution operator on the domain of n.

When the job-migration phase begins, the node resumes its job execution, however, no load-balancing can be triggered until after t_d. We now derive the average time elapse of this phase. Since each node has an average of $\bar{M}$ jobs to be migrated, the total job migration time is $m\bar{M}(T_{job} + \tau)$ in a collision-free environment, where T_{job} denotes the transmission time of a job. That is, $\bar{T}_{cd^-}(\,|\,\ell) = m\bar{M}(T_{job} + \tau)$. Since jobs are migrated one by one along the network in a collision-free manner, it is difficult to model the exact behavior of the node in this phase. For simplicity, we approximately model this phase as follows: nodes which have jobs to be migrated hold the jobs until the time instant t_{d^-} and then job migration is done instantaneously at t_d. This approximation is conservative in the sense that delaying the migration will actually increase the system response time. Errors caused by this approximation are expected to be fairly small in our analysis since T_{cd} is relatively shorter than T_{ad}. Based on this approximation, behavior of the node in $[t_c, t_{d^-})$ can be modeled in a similar way as in time interval $[t_a, t_b)$. Hence, given j jobs at t_c, the probability distribution on the number of jobs n at time $t \in [t_c, t_{d^-})$, $\beta_n(t\,|\,i)$, is derived in the same way as in the derivation of $p_n(t \mid k)$. That is,

$$\beta_n(t \mid i) = e^{-(\lambda+\mu)(t-t_c)} \Bigg[\left(\frac{\mu}{\lambda}\right)^{(i-n)/2} I_{n-i}(2\sqrt{\lambda\mu}(t-t_c)) + \left(\frac{\mu}{\lambda}\right)^{(n-i+1)/2} I_{n+i+1}(2\sqrt{\lambda\mu}(t-t_c)) + \left(1-\frac{\lambda}{\mu}\right)\left(\frac{\lambda}{\mu}\right)^n \sum_{l=n+i+2}^{\infty} \left(\frac{\mu}{\lambda}\right)^{l/2} \times I_l(2\sqrt{\lambda\mu}(t-t_c)) \Bigg], \quad (11)$$

and

$$w_n(t \mid k) = \sum_{i=0}^{\infty} \beta_i(t \mid i) p_i(t_b \mid k),$$

where $t \in [t_c, t_{d^-})$ and $k = n_I$ or $k = n_I + 1$.

Let the number of jobs at t_d^- in the nodes which initially have n_I and $n_I + 1$ jobs at t_a be $N(t_d^- \mid n_I)$ and $N(t_d^- \mid n_I + 1)$, respectively, the number of total jobs in the whole system at t_d is thus $N(t_d \mid \ell) = rN(t_d^- \mid n_I + 1) + (m-r)N(t_d^- \mid n_I)$. Let P_n denote the probability that n jobs are in the system at t_d, we get:

$$P_n = \sum_{\ell=0}^{\infty} P_\ell \overbrace{w_n(t_{d^-} \mid n_I) * \cdots, * w_n(t_{d^-} \mid n_I)}^{m-r} * \overbrace{w_n(t_{d^-} \mid n_I+1) * \cdots, * w_n(t_{d^-} \mid n_I+1)}^{r}. \quad (12)$$

Utilizing $P_\ell = P_n, \forall \ell = n$, we can obtain the steady-state probability P_ℓ, $\ell = 0, 1, \cdots$.

Once we get the steady-state probability distribution of system states at t_a, we can calculate the distribution $p_n(t)$ $(t \in [t_a, t_b))$, $q_n(t)$ $(t \in [t_b, t_c))$ and $w_n(t)$ $(t \in [t_c, t_d))$ as:

$$p_n(t) = \sum_{\ell=0}^{\infty} P_\ell \left(\frac{r}{m} p_n(t \mid n_I + 1) + \frac{m-r}{m} p_n(t \mid n_I) \right), \quad (13)$$

$$q_n(t) = \sum_{\ell=0}^{\infty} P_\ell \left(\frac{r}{m} p_n(t \mid n_I + 1) + \frac{m-r}{m} p_n(t \mid n_I) \right), \quad (14)$$

and

$$w_n(t) = \sum_{\ell=0}^{\infty} P_\ell \left(\frac{r}{m} p_n(t \mid n_I + 1) + \frac{m-r}{m} p_n(t \mid n_I) \right), \quad (15)$$

where $t = \ell \bmod m$, $n_I = \lfloor \ell/m \rfloor$ and

$$\bar{T}_{ab} = \sum_{\ell=0}^{\infty} P_\ell \bar{T}_{ab}(\mid \ell), \quad (16)$$

$$\bar{T}_{bc} = \sum_{\ell=0}^{\infty} P_\ell \bar{T}_{bc}(\mid \ell), \quad (17)$$

and

$$\bar{T}_{cd} = \sum_{\ell=0}^{\infty} P_\ell \bar{T}_{cd}(\mid \ell). \quad (18)$$

By plugging equations (16), (17) and (18) into equation (2), we can derive the average system response time.

We now calculate $T_{b'b}$ through analyzing the message delay on the CSMA/CD bus in the presence of load-balancing activities. $T_{b'b}$ consists of two sub-intervals, one of which is the time spent on bus contention and the other is the transmission time of initiation message, denoted as T_{int}. It can be seen from Figure 2 that the network is fully dedicated to load-balancing activity once a load-balancing initiation message (broadcast at $t_{b'}$) grasps the bus.

During $[t_b - T_{int}, t_d)$, the network is dedicated to load balancing activities and no normal message can be transmitted. Hence, the network can be modeled as having two classes of messages, one of which contains normal communication messages and the other, load-balancing messages which simulate the total transmission activities in $[t_b - T_{int}, t_d)$. It is evident that the load-balancing messages have an average interarrival time $\bar{T}_{ad}$ and average message length $\bar{T}_{bd} + T_{int}$. Let λ_{mesg} denote the normal message generation rate from all nodes and T_{mesg} denote the average message length, the offered load to the network from normal messages is thus $\rho_{mesg} = \lambda_{mesg} T_{mesg}$. Let $\lambda_{bal} = 1/\bar{T}_{ad}$, $T_{bal} = \bar{T}_{bd} + T_{int}$, the offered load from the load-balancing activity to the network is thus $\rho_{bal} = \lambda_{bal} T_{bal}$.

For simplicity, we assume that generation of load-balancing messages still follows a Poisson process and the length of each message follows an exponential distribution. Since it is very difficult to model the CSMA/CD behavior with multiple classes of messages, we extend the well-known result in Lam (1980) to approximate the average message delay with two classes of messages, based on an approach similar to the one in Schaar *et al.* (1991). Knowing that the total network utilization is $\rho = \rho_{mesg} + \rho_{bal} = \lambda_1 T_{mesg} + \lambda_2 T_{bal}$, we have the average message length as $\bar{T} = (\rho_{mesg} + \rho_{bal})/(\lambda_{mesg} + \lambda_{bal})$, and the average message delay becomes (Lam, 1980):

$$D = \left(\rho \frac{2 + (4e+2)a + 5a^2 + 4e(2e-1)a^2}{2(1 - \rho(1 + (2e+1)a))} + 1 + 2ea - \frac{(1 - e^{-2a\rho})2/\rho + 2ea^{-1} - 6a}{2(1/(1+\rho)e^{-\rho a - 1} - 1 + e^{-2\rho a})} + \frac{a}{2} \right) T, \quad (19)$$

where $a = \tau/\bar{T}$. Thus, the normal message delay is $D_{mesg} = D - \bar{T} + T_{mesg}$ and the contention time for the load-balancing initiation message $D - \bar{T}$ and $T_{b'b} = D - \bar{T} + T_{int}$.

We solve P_n and $T_{b'b}$ by a numerical, iterative method. First, we calculate P_n for a given $T_{b'b}$. Then, based on the derived distribution, we recalculate $T_{b'b}$ through calculating bandwidth utilized by load-balancing activities. The above process iterates until stable results for both P_n and $T_{b'b}$ are obtained.

4. DISCUSSION

In this section, we present both analytical and simulation results to evaluate performance of the ASLB scheme. We compare performance of the ASLB scheme with a well-known pair-sharing approach, called the Threshold scheme, through the Ribyl implementation suggested in Schaar *et al.* (1991). The Threshold scheme is a simple yet efficient load balancing algorithm proposed for a general distributed system (Eager *et al.*, 1986a). Threshold scheme in the CSMA/CD environment. The Ribyl algorithm proposed in Schaar *et al.* (1991) is essentially a server initiated Threshold scheme on CSMA/CD bus. It has been shown in Schaar *et al.* (1991) that Ribyl performed better in all conditions than the direct implementation of the Threshold algorithm (Eager *et al.*, 1986a). In the Ribyl, a node broadcasts the load-balancing initiation message once it becomes idle, and all other nodes with load greater than a threshold value T_l contend to send a job to the initiator following the CSMA/CD protocol. Once a node wins the contention, it migrates its job(s) to the initiator, and all other nodes withdraw from the contention once they recognize that a successful job migration has been completed.

In both analytical model and simulation, $m = 20$ nodes are connected through a CSMA/CD bus, which has the bandwidth of $B = 10$ Mbps and the propagation delay $\tau = 0.0225$ ms. The normal message size is assumed to be exponentially distributed with mean $T_{mesg} = 0.8$ ms. The length of a load message T_{load} and load-balancing initiation message T_{int} are assumed to be fixed to the shortest allowable message length in a CSMA/CD network: $T_{load} = T_{int} = 0.057$ ms. The sizes of jobs are assumed to follow an exponential distribution with a mean of 10 000 bits, or the mean transmission time $T_{job} = 1$ ms, equivalently. Jobs arriving at each node

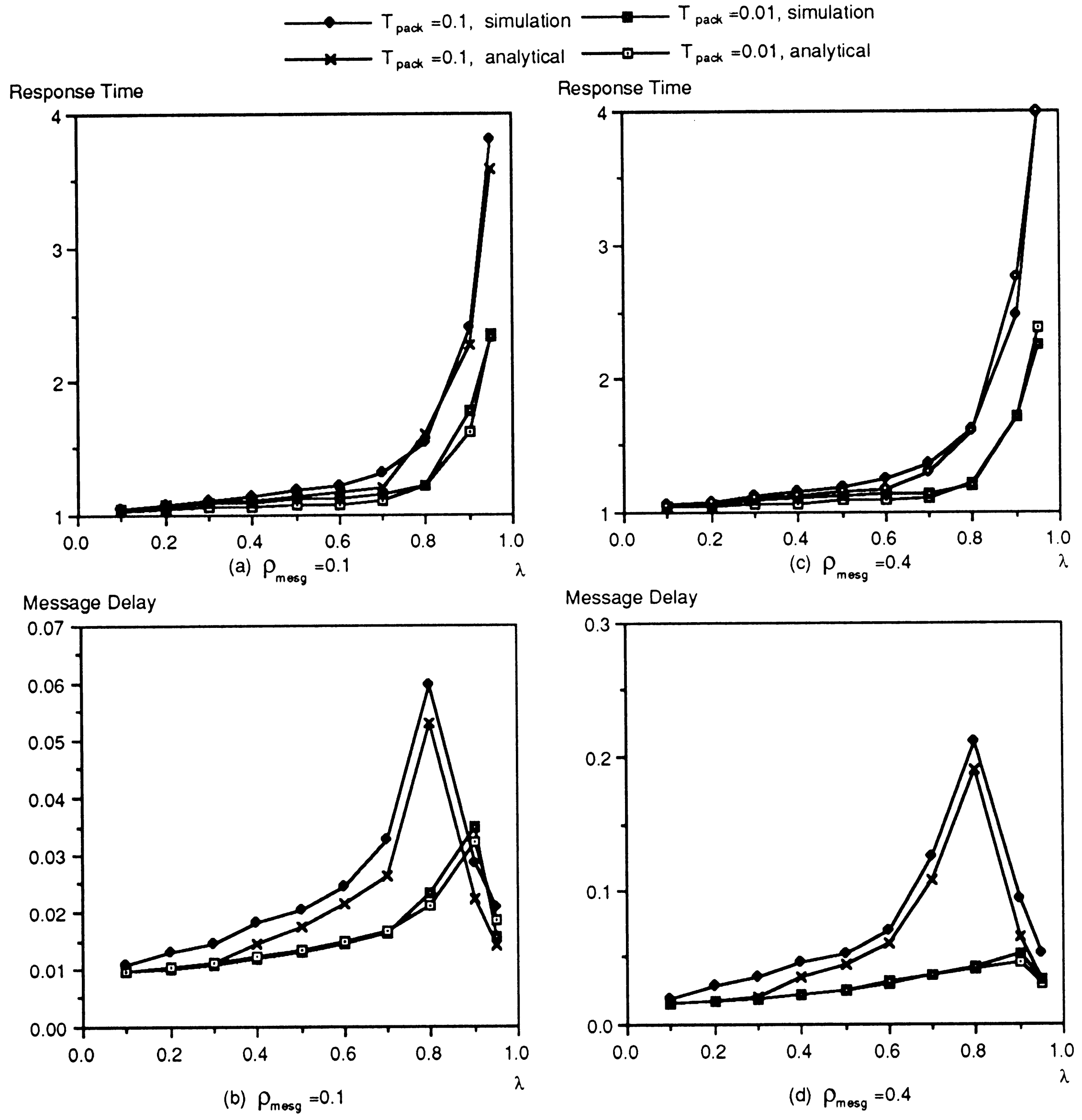

FIGURE 4. Analytical and simulation results on average system response time and average normal message delay for different packing costs, system loads and network loads.

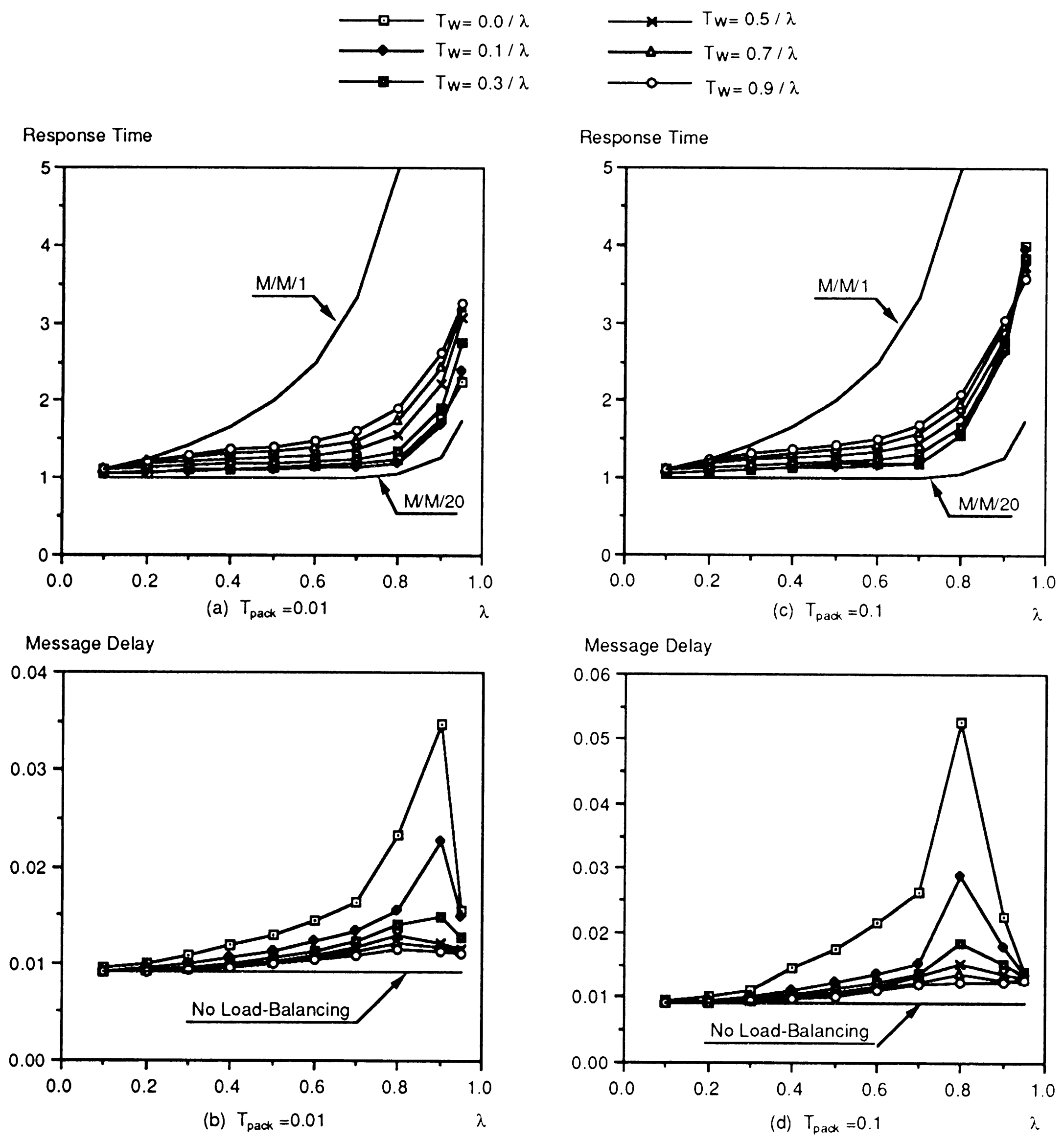

FIGURE 5. The system performance under different initiation-waiting time T_w and packing cost when the offered network load is 0.1.

follow a Poisson process with arrival rate λ. Service times of jobs are assumed to be exponentially distributed with mean $1/\mu = 100\,\text{ms}$. Execution of the FP algorithm in ASLB is set at $T_{FP} = 0.1\,\text{ms}$. We further normalize all parameters with respect to job service time $1/\mu$, i.e. $\tau = 0.000\,225$, $T_{mesg} = 0.008$, $T_{load} = 0.000\,57$; $T_{int} = 0.000\,57$, $T_{job} = 0.01$ and $T_{FP} = 0.001$. Performance of both ASLB and Ribyl schemes is evaluated for different values of two system parameters: the network load offered by normal communication messages, denoted as ρ_{mesg}, and the processing time of packing a job before the job is migrated, denoted as T_{pack}. The confidential interval of the simulation is 95%.

The average system response time and the normal message delay of ASLB obtained through both analytical model and simulation are presented in Figure 4 with respect to varying ρ_{mesg}, λ and T_{pack}. The system response time and average normal message delay obtained from our model and from simulation closely match each other under most conditions. It should be noted that the reason for the 'convex' shape of the message delay curve is that when the system load gets high enough, load-balancing activities decline since the chance that nodes become idle decreases and, therefore, the bandwidth used by load-balancing activities reduces.

Comparing Figure 4(b) and 4(d), it should be noticed that ρ_{mesg} has a significant impact on the normal message delay of ASLB, especially when the system load (λ/μ) is in the range of 0.7–0.9, for which load-balancing activity is frequent. To reduce the normal message delay, ASLB can be slightly modified as follows. When a node becomes idle, it will broadcast the load-balancing initiation message after delaying a time period T_w, if no new job nor load-balancing initiation message arrive in the waiting period. By adjusting T_w, we can reduce the frequency of load balancing activities. An analytical

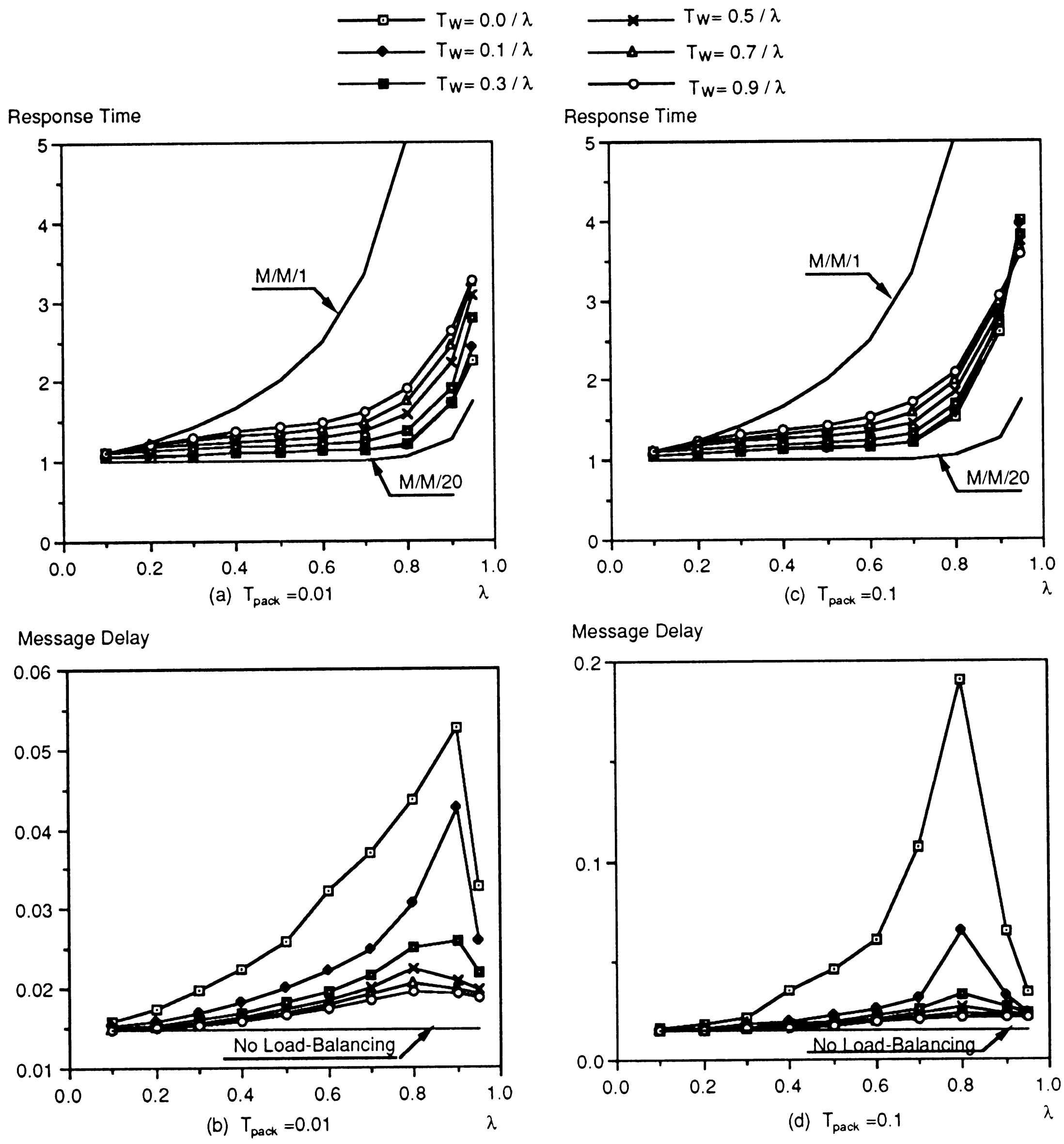

FIGURE 6. System performance on under different initiation-waiting time T_w and packing cost when network offered load is 0.4.

model, which takes T_w into account to derive the time elapse of the independent job-execution phase is given in Appendix D. The analytical results under different waiting time T_w are plotted in Figures 5 and 6, when the offered network loads are $\rho_{mesg} = 0.1$ and $\rho_{mesg} = 0.4$, respectively. The system response times of the $M/M/1$ and $M/M/20$ queueing systems are upper and lower bounds for any load-balancing scheme. It can be seen that the system response time of ASLB is close to that of the $M/M/20$ queueing system when the system is not extremely heavily loaded. When $T_w < 1/2\lambda$, it has little impact on the system response time; however, the normal message delay decreases significantly. The normal message delay improves little with further increase of T_w, and the system response time begins to deteriorate. When $T_w = 1/2\lambda$, a good trade-off between system response time and the normal message delay can be found in which the normal message delay is nearly as low as the case with no load-balancing and the system response time deteriorate only slightly.

The above observation can be explained as follows. As it has been explained in Section 3, the average bandwidth consumed by the load-balancing activities is

$$\rho_{bal} = \frac{T_{int} + (m-1)\,T_{load} + T_{FP} + \bar{M}T_{pack} + \bar{T}_{cd}}{\bar{T}_{ab} + \bar{T}_{bd}}.$$

When T_w increases, $\bar{T}_{ab}$ increases and thus ρ_{bal} may reduce. Since $\bar{M}$ is determined by the load distribution at the beginning of load-balancing decision phase, increasing $\bar{T}_{ab}$ will inevitably increase the load difference between nodes at the end of $\bar{T}_{ab}$ and thus will likely increase $\bar{M}$. Hence, when T_w increases beyond a certain value, further increase in T_w will not reduce ρ_{bal} and the normal message delay. It is also interesting to observe from Figure 6 that when the system load is extremely high (ρ is around 0.95) and the packing cost for job

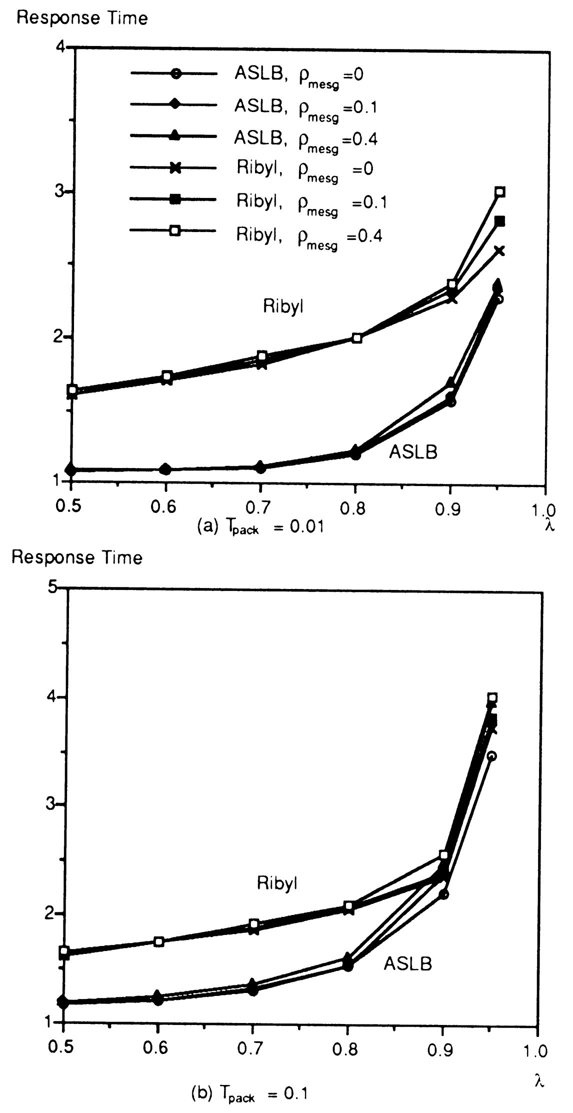

FIGURE 7. Performance comparison of the ASLB scheme and the Ribyl scheme under different packing costs and network utilization.

migration is also high ($T_{pack} = 0.1$), the system response time decreases with load-balancing frequency (i.e. increase of T_w). This indicates that when both the system load and job migration cost are high, frequent load-balancing does not necessarily improve the system performance.

We now analyze the impact of the network load on the system response time. In Figure 7, we compare the simulated average system response time of both ASLB and Ribyl schemes with varying ρ_{mesg}. It is shown that the network load has little impact on the system response time of ASLB scheme, given that the network is not overloaded. The main reasons that the network load has little effect on the system response time of ASLB are (1) the job migration and load information collection are done in a collision-free manner and (2) only the initiation message needs to contend for the bus. The network load, however, has larger impact on the system response time of the Ribyl scheme since both job migration and load-balancing initiation messages need to compete for the bus.

The average system response time of the ASLB is much shorter than that of the Ribyl scheme in most cases. We believe that ASLB performs better than Ribyl mainly because (1) the system load under ASLB is nearly equally distributed after each load-balancing activity, and (2) the chance that jobs are migrated more than once is quite low in ASLB, as shown in Figure 8. A significant number of jobs (around 70%) are executed locally, and when jobs are migrated, most of them are migrated only once.

Finally, we compare the performance of ASLB with that of the Ribyl scheme when nodes have four different job arrival rates:

1. Five nodes have low job arrival rate ($\lambda = 0.1$) and the remaining nodes have a high arrival rate ($\lambda = 0.9$).
2. Reverse of case 1.
3. Ten nodes have job arrival rate $\lambda = 0.1$ and the other 10 have $\lambda = 0.9$.
4. Five nodes have job arrival rate $\lambda = 0.1$, five nodes have $\lambda = 0.4$, five nodes have $\lambda = 0.7$ and the other five have $\lambda = 0.9$.

Results on these four cases given in Figure 9 show that the performance of ASLB is very stable and is much better than that of Ribyl.

5. CONCLUSIONS

In this paper, we analyzed a dynamic load-balancing scheme, ASLB, for a distributed system on the CSMA/CD bus network. In ASLB, the workload is evenly distributed throughout the system in a collision-free manner in each load-balancing activity. The system performance is significantly improved by the ASLB under different computation and communication workloads. We can make a tradeoff between the system response time and message delay through simple adjustment of the load-balancing initiation condition. A comprehensive transient analysis technique to capture dynamic behaviors of the system under ASLB load-balancing activities is presented. Our technique can also be applied to other distributed algorithms, provided that the initiating phase of the algorithm is not excessively long, because we did not single out the behavior of the initiator in the analysis. In ASLB, any node can trigger the load balancing activity once it becomes idle. Note that, however, we do not assume that the load balancing initiator remains idle in the load balancing phase, since jobs may arrive at any node during a load balancing activity and jobs are migrated between all nodes. Therefore, the initiator is not treated differently from any other nodes, in terms of job arrivals, job distribution, and job execution. The only possible difference in the behavior of nodes is when the initiator is trying to initiate a load balancing activity. At this period, a potential initiator can only accept new jobs, but cannot have any

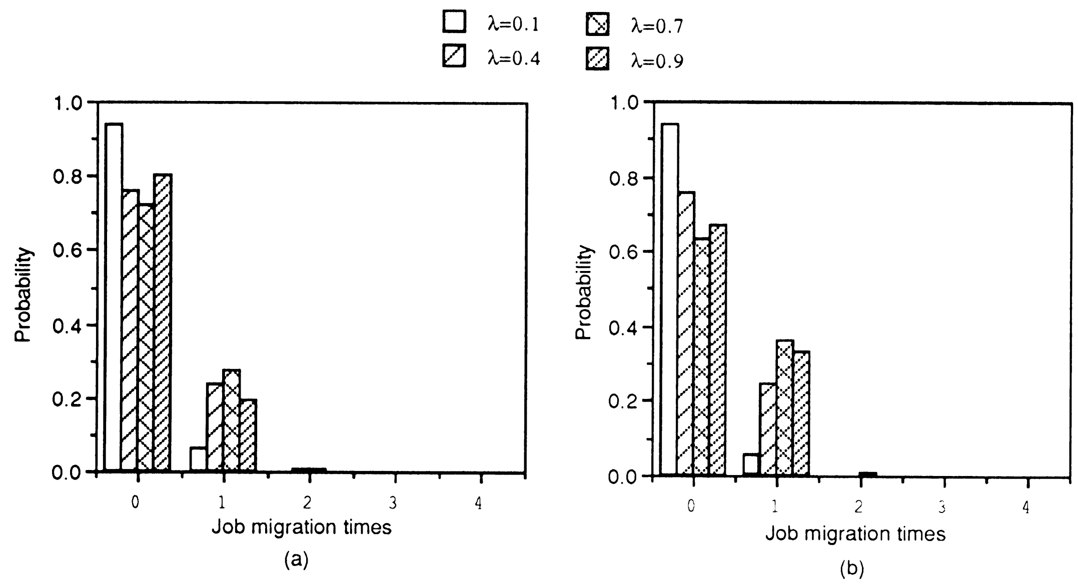

FIGURE 8. The probability distribution of job migration frequency under ASLB for difference job packing costs when network utilization $\rho_{mesg} = 0.1$: (a)$T_{pack} = 0.01$ and (b) $T_{pack} = 0.1$.

jobs for execution (because it is idle). Since the relative duration of (t_b', t_b) is much shorter than that of other relevant events, the analysis error introduced by the approximation that all nodes are identical is negligible, as evidenced by our simulation results. For distributed algorithms which have a long initiation time, it may be important to consider the different behavior of an initiator.

ACKNOWLEDGMENT

The authors wish to thank the anonymous reviewers for their constructive comments which led to substantial improvement of this paper.

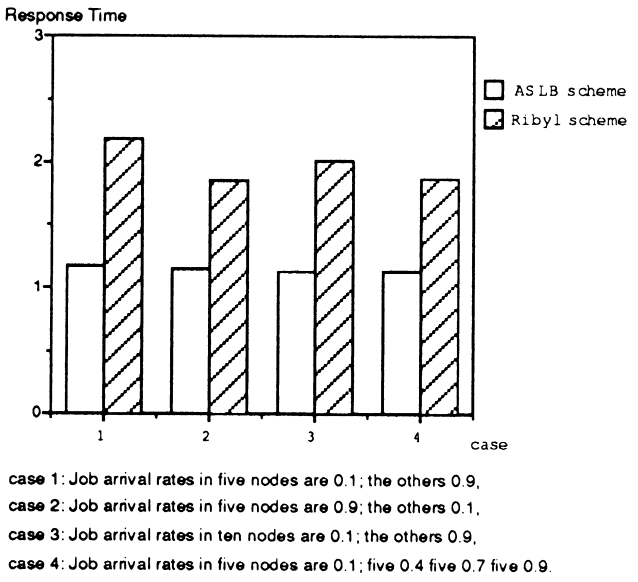

FIGURE 9. Performance comparison of ASLB and Ribyl schemes when the job arrival rates are non-uniform in different nodes, and $\rho_{mesg} = 0.1$, $T_{pack} = 0.01$.

REFERENCES

Baumgarter, K. M. and Wah, B. W. (1989) GAMMON: a load balancing strategy for local computer systems with multiaccess networks. *IEEE Trans. Comput.*, **C-38**, 1098–1109.

Bonomi, F. and Kumar, A. (1988) Adaptive optimal load balancing in a heterogeneous multiserver system with a central job scheduler. In *Proc. 1988 IEEE Int. Conf. on Distributed Computing Systems*, pp. 500–508.

Casavant, T. J. and Kuhl, J. G. (1986) A formal model of distributed decision making and its application to distributed load balancing. In *Proc. 1986 IEEE Int. Conf. on Parallel Processing*, pp. 232–239.

Chow, Y. C. (1982) Load balancing in distributed systems. *IEEE Trans. Software Eng.*, **8,** 401–412.

Chow, Y. C. and Kohler, W. (1979) Models for dynamic load balancing in a heterogeneous multiple processor system. *IEEE Trans. Comput.*, **C-28**, 334–361.

Cox, D. R. and Smith, W. L. (1961) *Queues*. Chapman & Hall, London.

Eager, D. L., Lazowska, E. D. and Zahorjan, J. (1986a) Adaptive load sharing in homogeneous distributed systems. *IEEE Trans. Software Eng.*, **12**, 662–675.

Eager, D. L., Lazowska, E. D. and Zahorjan, J. (1986b) A comparison of receiver-initiated and sender-initiated adaptive load sharing. *Performance Evaluation*, **6**, 53–68.

Eager, D. L., Lazowska, E. D. and Zahorjan, J. (1988) The limited performance benefits of migrating processes for load sharing. *Performance Evaluation Rev.*, **16**(1), 63–72.

Ferrari, D. and Zhou, S. (1986) A load index for dynamic load balancing. In *Proc. ACM–IEEE Fall Joint Computer Conf.*, pp. 684–690.

Gross, D. and Harris, C. M. (1985) *Fundamentals of Queueing Theory*, 2nd edn. John Wiley, New York.

Hac, A. (1989) Load balancing in distributed systems: a summary. *Performance Evaluation Rev.*, **6**(2–4), 17–19.

Intel (1990) *Microcommunications.* Intel Corporation, Santa Clara, USA.

Kim, J. L., Liu, J. C. and Hao, Y. (1992) An all sharing load balancing protocol in distributed systems based on CSMA/CD networks. In *Proc. 12th IEEE Int. Conf. on Distributed Computing Systems*, pp. 82–89.

Kremien, O. and Kramer, J. (1992) Methodological analysis of adaptive load sharing algorithms. *IEEE Trans. Parallel and Distributed Systems*, **3**, 747–760.

Kunz, K. (1991) The influence of different work load descriptions on a heuristic load balancing scheme. *IEEE Trans. Software Eng.*, **17**, 725–730.

Kurose, J. F. and Simha, R. (1988) A distributed algorithm for optimal static load balancing in distributed computer systems. In *Proc. INFOCOM'86*.

Lam, S. S. (1980) A carrier sense multiple access protocols for local networks. *Comp. Networks*, **21**, 21–32.

Lin, F. C. H. and Keller, R. M. (1986) Gradient method: a demand-driven load balancing scheme. In *Proc. 1986 IEEE Int. Conf. on Distributed Computing Systems*, pp. 329–336.

Little, J. D. C. (1961) A proof for the queueing formula $l = \lambda w$. *Operations Res.*, **9**, 383–387.

Livny, M. and Melman, M. (1982) Load balancing in homogeneous broadcast distributed systems. In *Proc. 1982 ACM Computer Network Performance Symp.*, pp. 47–55, Publisher?

Ni, L. M. Hwang, K. (1981) Optimal load balancing strategies for a multiple processor systems. In *Proc. IEEE 10th Int. Conf. Parallel Processing*, pp. 352–357.

Preparata, F. P., Metze, G. and Chien, R. T. (1967) On the connection assignment problem of diagnosable systems. *IEEE Trans. Electron. Comput.*, **16**, 848–854.

Pulidas, S., Towsley, D. and Stankovic, J. A. (1988) Imbedding gradient estimators in load balancing algorithms. In *Proc. 1988 IEEE Int. Conf. on Distributed Computing Systems*, pp. 482–490.

Robert, E. (1993) Fast results from the fast ethernet alliance. *LAN Times*, **10**, no. 25.

Rommel, C. G. (1991) The probability of load balancing success in a homogeneous network. *IEEE Trans. Software Eng.*, **17**, 922–933.

Schaar, M., Efe, K., Delcambre, L. and Bhuyan, L. N. (1991) Load balancing with network cooperation. *Proc. Int. Conf. on Distributed Computing Systems*, pp. 000–000, Publisher.

Schwartz, M. (1987) *Telecommunication Networks: Protocols, Modeling and Analysis*. Addison-Wesley, Reading, MA.

Shivaratri, N. G. and Krueger, P. (1990) Two adaptive policies for global scheduling algorithms. In *Proc. 10th Int. IEEE Conf. on Distributed Computing Systems*, pp. 502–509.

Stankovic, J. A. (1985) An application of Bayesian decision theory to decentralized control of job scheduling. *IEEE Trans. Comput.*, **C-34**, 117–130.

Tantawi, A. N. and Towsley, D. (1985) Optimal static load balancing in distributed computer systems. *J. ACM*, **32**, 445–465.

Wang, Y. T. and Morris, R. J. T. (1985) Load sharing in distributed systems. *IEEE Trans. Comput.*, **C-34**, 204–217.

Zhou, S. (1988) A trace-driven simulation study of dynamic load balancing. *IEEE Trans. Software Eng.*, **14**, 1327–1341.

APPENDIX A. LIST OF SYMBOLS

t_a	The time instant at which a load-balancing activity just completes. It is the beginning of a job execution phase.
t_b	The time instant at which one of the node in the system becomes idle. It is the beginning of the load-balancing decision phase.
t_c	The time instant at which job migration begins.
t_d	Job migration completes and another cycle of load-balancing activity begins. It is equivalent to time instant t_a.
T_{ab}	The time elapse of independent job execution phase.
T_{bc}	The time elapse of load-balancing decision phase.
T_{cd}	The time elapse of job migration phase.
$T_{b'b}$	The transmission time of the load-balancing initiation message.
T_w	The time elapse between the instant at which a node becomes idle and the time it begins to broadcast the load-balancing initiation message.
$p_n(t)$	The probability that n jobs are in a node at a time instant within the independent job execution phase.
$q_n(t)$	The probability that n jobs are in a node at a time instant within load-balancing decision phase.
$w_n(t)$	The probability that n jobs are in a node within the job migration phase.
P_ℓ	The steady state probability that ℓ jobs are in the system.
$F_{T_{ab'}}(t), f_{T_{ab'}}$	The probability distribution and density function of the time elapse from the beginning of a cycle to the time instant at which one of the node becomes idle.
T_{load}	The length of the message carrying load message (in time unit).
T_{mesg}	The length of normal communication message (in time unit).
T_{FP}	The time for executing algorithm FP.
T_{job}	The length of message carrying a migrating job.
T_{pack}	The processing time of a node for packing a job.
T_{int}	The length of load-balancing initiation message (in time unit).
τ	The propagation delay of the bus.
λ	The interarrival rate of jobs.
μ	The service rate of a node.
λ_{mesg}	The generation rate of normal communication messages.
ρ_{mesg}	The offered load of the normal communication message to the network.
L	Average number of jobs in a node.
R	Average system response time.
D	Average delay of normal messages.

APPENDIX B. PSEUDO-CODE OF THE ASLB PROTOCOL

```
BEGIN
if(JOB_COMPLETION = TRUE)
  Dequeue(Job_queue, job); /* Get the another job in the
    job queue*/
  if((job=NULL) and (NODE_STATE=
    NORMAL)) /* No job in job queue*/
    Set(NODE_STATE, INITIATION);
    message.id=LB_INITIATION;
    message.source = my_address;
    message.destination = all nodes;
    Send(message); /* Initiate load-balancing by sending
      the initiation message to its communication
      processor*/
  else
    Execute(job);
if(JOB_ARRIVE = TRUE)
  Queue(job, Job_queue);
  if((NODE_STATE = INITIATION) and
    (Abort(message) = SUCCESS))
    Set(NODE_STATE, NORMAL);
if(Receive(message) = TRUE)
  case message.id = LB_INITIATION:
    if(NODE_STATE = NORMAL)
      Set(NODE_STATE, INITIATION);
      if(my_address = (message.source + 1) mod N)
        /*This node is the organizer*/
        Set_Timer(2r + T_r);
        Abort(message);
        Backup(message, out_message_queue);
          /*Purge the out-message queue*/
        TBL(message.source) = 0;
    break;
  case message.id = LOAD:
    TBL(message.source) = message.load;
    if(Status(TBL) = FULL) /*All node broadcast
      their loads*/
      FP; /* Find load balancing partner*/
      Pop(partner(partner.source, partner.destination,
        job.number));
      if(partner.source = my_address)
        message.id = JOB;
        message.source = partner.source;
        message.destination = partner.destination;
        Dequeue(Job_queue, job, job.number);
        message.content = job;
        Send(message); /* Send the job to
          communication processor for migration*/
    else
        if(my_address = (message.source + 1) mod N)
          message.id = LOAD;
          message.source = my_address;
          message.destination = all_nodes;
          message.content = my_load;
          Send(message);
  break;
case message.id = JOB:
  if(message.destination = my_address)
    job = message.content;
    Enqueue(job);
    if(Pop(partner(partner.source, partner.destination,
      job.number)) = NULL)
        /* No job to be migrated*/
      Set(NODE_STATE, NORMAL);
    else
      if(partner.source = my_address)
        message.id = JOB;
        message.source = partner.source;
        message.destination = partner.destination;
        Dequeue(Job_queue, job, partner.number);
        message.content = job;
        Send(message);
  end case;
if(EXPIRE(Timer))
  message.id = LOAD;
  message.content = my_load;
  message.destination = all_nodes;
  Send(message);
END
```

APPENDIX C. DERIVATION OF $f_T(t)$ AND $p_n(t \mid k)$

Derivation of $f(T = t \mid k), k \neq 0$

The derivation of $f(T = t \mid k)$ is briefly described here. More comprehensive discussion can be found in Cox and Smith (1961).

By applying z-transform to both sides of equations (5), we have

$$\frac{\partial P(z,t)}{\partial t} = \frac{\mu - (\lambda + \mu)z + \lambda z^2)}{z}(P(z,t) - p_0(t)), \quad (20)$$

where $P(z,t)$ is the z-transform of $p_n(t)$. Then we apply the Laplace transform to both sides of equation (20), noting that $\partial P(z,t)/\partial t = sP(z,s) - P(z,0)$ and $P(z,0) = z^k$ initially, and we get

$$P(z,s) = \frac{z^2 - (1-z)(\mu - \lambda z)(z_1/s)}{\lambda(z - z_1)(z_2 - z)}, \quad (21)$$

where $P(z,s)$ is the Laplace transform of $P(z,t)$ and

$$\begin{cases} z_1 = \dfrac{\lambda + \mu + s - \sqrt{(\lambda + \mu + s)^2 - 4\lambda\mu}}{2\lambda} \\ z_2 = \dfrac{\lambda + \mu + s + \sqrt{\lambda + \mu + s)^2 - 4\lambda\mu}}{2\lambda} \end{cases} \quad (22)$$

Thus,

$$p_0(s) = P(0,s) = \frac{2\mu}{s(\lambda + \mu + s + \sqrt{(\lambda + \mu + s)^2 - 4\lambda\mu})} \quad (23)$$

Since $\mathscr{L}\{p_0'(t)\} = sp_0(s) - p_0(0) = sp_0(s)$, after applying the reverse Laplace transform, we get,

$$p_0'(t) = \frac{\left(\frac{\mu}{\lambda}\right)^{k/2} e^{-(\lambda+\mu)t} I_k(2\sqrt{\lambda\mu}t)}{t}. \quad (24)$$

That is, we get $f_T(t \mid k)$, which is the same as $p_0'(t)$.

Derivation of $p_n(t \mid k)$, $t \in [t_a, t_b)$

In this analysis, we use $p_n(t)$ $t \in [0, t_b - t_a)$ to represent $p_n(t \mid k)$, $t \in [t_a, t_b)$ for notational convenience. Since a node in the independent job-execution phase can be modeled as an $M/M/1$ queuing system, the differential-difference equations governing system states are

$$\begin{cases} p_0'(t) = -\lambda p_0(t) + \mu p_1(t) \\ p_n'(t) = -(\lambda+\mu)p_n(t) + \lambda p_{n-1}(t) + \mu p_{n+1}(t) \quad (n>0) \end{cases} \tag{25}$$

By applying the z-transform to both sides of equations (25), we get

$$\frac{\partial P(z,t)}{\partial t} = \frac{1-z}{z}[(\mu - \lambda z)P(z,t) - \mu p_0(t)], \tag{26}$$

where $P(z,t)$ is the z transform of $p_n(t)$. By applying the Laplace transform to equation (26), we get

$$P(z,s) = \frac{z^{i+1} - \mu(1-z)p_0(s)}{(\lambda+\mu+s)z - \mu - \lambda z^2}, \tag{27}$$

where $P(z,s)$ is the Laplace transform of $P(z,t)$.

Since the denominator has two zeros in equation (22), the equation can thus be rewritten as

$$P(z,s) = \frac{z^{k+1} - \mu(1-z)p_0(s)}{\lambda(z-z_1)(z_2-z)}. \tag{28}$$

From Rouchè's theorem (Gross and Harris, 1985), it is known that z_1 must also be a zero of the nominators, that is $z_1^{k+1} - \mu(1-z_1)p_0(s) = 0$, or equivalently, $p_0(s) = z_1^{i+1}/\mu(1-z_1)$. Using the properties $z_1 + z_2 = \lambda + \mu + s/\lambda$, and $z_1 z_2 = \mu/\lambda$, we can rewrite equation (28) as:

$$P(z,s) = \frac{1}{\lambda z_2}(z^k + z_1 z^{k-1} + \cdots + z_1^k) \times \sum_{i=0}^{\infty}\left(\frac{z}{z_2}\right)^i + \frac{z_1^{k+1}}{\lambda z_2(1-z_1)}\sum_{i=0}^{\infty}\left(\frac{z}{z_2}\right)^i. \tag{29}$$

Since

$$\mathscr{L}\{P(z,t)\} = \sum_{n=0}^{\infty} p_n(s) z^n,$$

so the coefficient of z^n is $p_n(s)$, i.e.

$$p_n(s) = \frac{1}{\lambda}\left[\frac{1}{z_2^{n-k+1}} + \frac{\mu/\lambda}{z_2^{n-k+1}} + \cdots + \frac{(\mu/\lambda)^k}{z_2^{n-k+1}} + \left(\frac{\lambda}{\mu}\right)^{n+1} \sum_{l=n+k+2}^{\infty}\left(\frac{\mu}{\lambda z_2}\right)^l\right]. \tag{30}$$

Since

$$\left(\frac{s + \sqrt{s^2 - 4\lambda\mu t}}{2\lambda}\right)^{-i} = \mathscr{L}\left\{i\left(\frac{\lambda}{\mu}\right)^{i/2} t^{-1} I_n(2\sqrt{\lambda\mu t})\right\},$$

we get

$$p_n(t\,|\,k) = e^{-(\lambda+\mu)t}\left[\left(\frac{\mu}{\lambda}\right)^{(k-n)/2} I_{n-k}(2\sqrt{\lambda\mu\delta}) + \left(\frac{\mu}{\lambda}\right)^{(k-n+1)/2} I_{n+k+1}(2\sqrt{\lambda\mu t}) + \left(1-\frac{\lambda}{\mu}\right)\left(\frac{\lambda}{\mu}\right)^n \sum_{l=n+k+2}^{\infty}\left(\frac{\mu}{\lambda}\right)^{l/2} I_l(2\sqrt{\lambda\mu t})\right], \tag{31}$$

where $0 \leqslant t < t_b$. Recall that we use $p_n(t)$ $(0 \leqslant t < t_b)$ to substitute for $p_n(t\,|\,k)$ $(t_a \leqslant t < t_b)$, we get equation (8) in Section 3.

APPENDIX D. DERIVATION OF $f_T(t)$ CONSIDERING WAITING TIME T_w

Since a node will not initiate load-balancing immediately after it becomes idle, we need to modify derivation of $f_T(t\,|\,k)$, where $k = n_I$ or $n_I + 1$ following the notation in Section 3. Note that when a node becomes idle, it has to wait for a period of T_w before it can broadcast a load-balancing initiation message. The probability that no new job arrives during the time period T_w is $1 - \lambda e^{-\lambda T_w}$. Thus, the density function of T is modified as

$$f_T(t\,|\,k) = f_T^{org}(t\,|\,k)(1 - \lambda e^{-\lambda T_w}), \tag{32}$$

where $f_T^{org}(t\,|\,k)$ is the density function of T derived in Section 3. Therefore, we can derive $f_T(t\,|\,k)$ by first deriving $f_T^{org}(t\,|\,k)$ using the method in Section 3 and then simply plugging it into equation (32).