JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING **33,** 107–121 (1996)
ARTICLE NO. 0030

A Simulation Study on Distributed Mutual Exclusion[1]

YE-IN CHANG[2]

Department of Applied Mathematics, National Sun Yat-Sen University, Kaohsiung, Taiwan, Republic of China

In the problem of mutual exclusion, concurrent access to a shared resource using a structural program abstraction called a *critical section* (CS) must be synchronized such that at any time only one process can enter the CS. In a distributed system, due to the lack of both a shared memory and a global clock, and due to unpredictable message delay, the design of a distributed mutual exclusion algorithm that is free from deadlock and starvation is much more complex than that in a centralized system. Based on different assumptions about communication topologies and a widely varying amount of information maintained by each site about other sites, several distributed mutual exclusion algorithms have been proposed. In this paper, we suvrey and analyze several well-known distributed mutual exclusion algorithms according to their related characteristics. We also compare the performance of these algorithms by a simulation study. Finally, we present a comparative analysis of these algorithms. © 1996 Academic Press, Inc.

1. INTRODUCTION

The mutual exclusion problem was originally considered in centralized systems for the synchronization of exclusive access to the shared resource. In the problem of mutual exclusion, concurrent access to a shared resource or the critical section (CS) must be synchronized such that at any time only one process can access the CS.

A distributed system consists of a collection of geographically dispersed autonomous sites connected by a communication network. The sites have no shared memory, no global clock, and communicate with one another by passing messages. Message propagation delay is finite but unpredictable. In a distributed system, due to the lack of both in shared memory and a global clock, and due to unpredictable message delay, the design of a distributed mutual exclusion algorithm that is free from deadlock and starvation is much more complex than that in a centralized system.

Over the past decade, many algorithms have been proposed to achieve mutual exclusion in distributed systems. These algorithm can be divided into two classes: *token-based* and *non-taken-based* [24] (or *permission-based* [17]).

[1] This research was supported by National Science Council of the Republic of China, NSC-81-0408-E-110-508.

[2] E-mail: changyi@math.nsysu.edu.tw.

In *token-based* algorithms [14–16, 20, 22, 25, 26], only the site holding the token can execute the CS and make the final decision on the next site to enter the CS. In *non-token-based* algorithms [3, 12, 13, 19, 21, 23], a requesting site can execute the CS only after it has received permission from each member of a subset of sites in the system, and every site receiving a CS request message participates in making the final decision. There is an orthogonal classification of mutual exclusion algorithms [24]: *static* and *dynamic*. A mutual exclusion algirithm is *static* if its actions do not depend upon the current system state (or history); otherwise, it is *dynamic*.

In token-based algorithms, a unique token is shared among the sites, and the possession of the token gives a site the authority to execute the CS. Singular existance of the token implies the existence of mutual exclusion in a distributed system. Depending on whether or not a logical configuration is imposed on sites, the token-based algorithms can be further classified into the following two approaches [24]: *broadcast-based* and *logical-structure-based*.

In *broadcast-based* algorithms, no structure is imposed on sites and a site sends token request messages to other sites in parallel. *Broadcast-based* algorithms are divided into two classes: static and dynamic. Static algorithms are memoryless because they do not remember the history of CS executions. In these algorithms [15, 20, 25], a requesting site sends token requests to all other sites. Dynamic algorithms [22], on the other hand, remember a recent history of the token locations and send token request messages only to a dynamically selected sites which are likely to have the token.

In *logical-structure-based* algorithms, sites are woven into a logical configuration. These algorithms can be static or dynamic. In static algorithms, the logical structure remains unchanged and only the direction of edges changes. For example, in a tree-based algorithm [16], sites are usually organized in a special configuration (e.g., tree). Requests sequentially propagate through the paths between the requesting site and the site holding the token, and so does the token. In dynamic algorithms [14, 26], a dynamic logical tree is maintained such that the root is always the site which will hold the token in the near future (i.e., the root is the last site to get the token among the current requesting sites) when no message is in transit. The token is directly sent to the next requesting site to execute the CS, but a request is sequentially forwarded along a virtual

0743-7315/96 $18.00

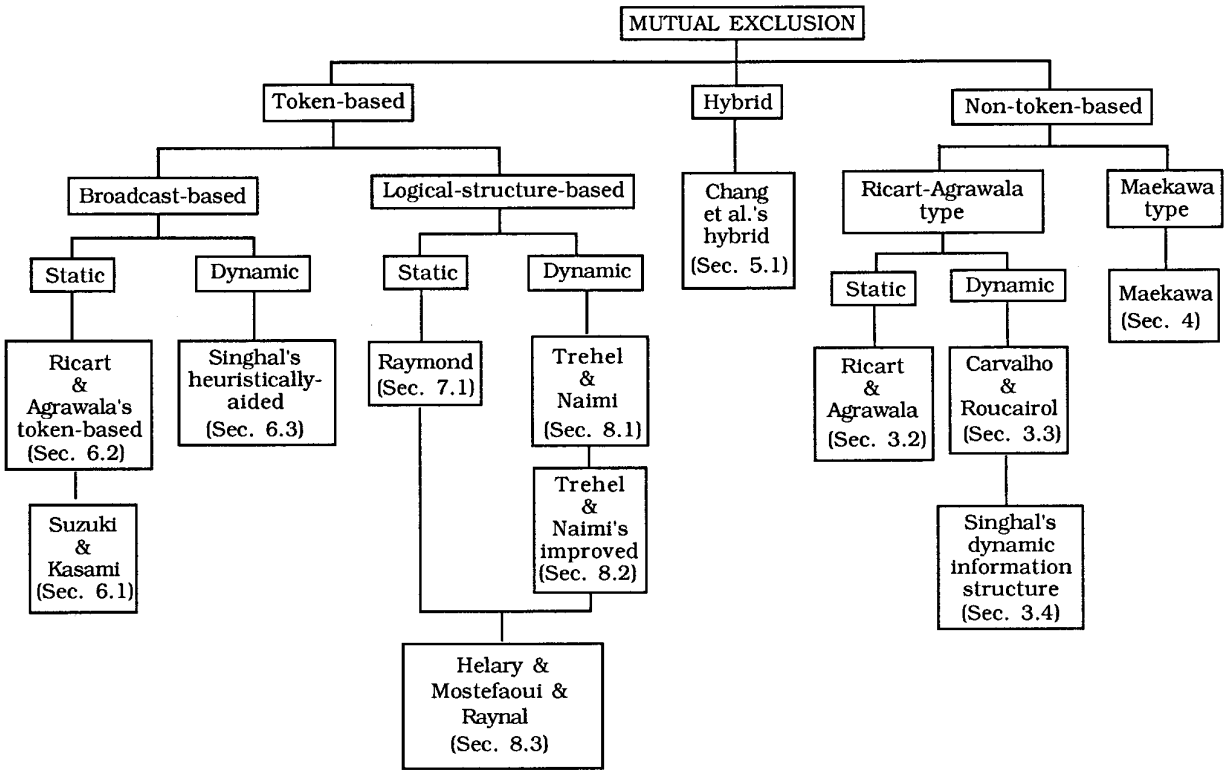

FIG. 1. Classification.

path to the root. Helary *et al.* have proposed a general scheme for token- and tree-based distributed mutual exclusion algorithms [11], which covers those algorithms [14, 16, 26] based on the static and dynamic *logical-structure-based* approaches.

In non-token-based algorithms, a *request set* at a site X (denoted as R_x) is used to record the identifiers of the sites to which site X sends CS request messages when requesting the CS. A site-invoking mutual exclusion can enter the CS only after it has received permission from all the sites whose identifiers are in its request set. To simplify the design of a non-token-based algorithm, the following two assumptions are usually made: (1) the network topology is logically fully connected; (2) between any pair of sites, messages are delivered in the order in which they are sent. To ensure that every site resolves conflicting requests in the same way, a unique *timestamp* is included in each CS request message to order the request. Depending on how a request set is formed, the non-token-based algorithms can be classified into the following two approaches [24]: *Ricart–Agrawala-type* and *Mackawa-type*.

In *Ricart–Agrawala-type* algorithm, a site grants permission to a requesting site immediately if it is not requesting the CS or its own request has lower priority. Otherwise, it defers granting permission until its execution of the CS is over. That is, while granting a permission, a site looks into only its own conflict. A site can grant permission to many requesting sites simultaneously. This approach must ensure $Y \in R_x$, where site Y is executing the CS and site X is requesting the CS. The Ricart–Agrawala algorithm [19] is static because the contents of the request sets do not change as the algorithm executes. Carvalho–Roucairol proposed an improved variation of the Ricart–Agrawala algorithm where a site remembers the recent history of the CS executions and minimizes the number of messages exchanged [3]. (This was the informal start of dynamic Ricart–Agrawala-type algorithms.) Singhal for the first time has formally proposed a dynamic Ricart–Agrawala-type algorithm, proved its correctness, and analyzed its performance characteristics [23].

In *Maekawa-type* algorithms, on the contrary, a site can grant permission only to one site at a time [13, 21]. A site grants permission to a requesting site only if it has not currently granted permission to another site. (That is, site X is *locked* by site Y if site X grants permission to site Y.) Otherwise, it delays granting permission until the currently granted permission has been released. The request set of a site X contains the identifiers of the sites which are exclusively locked by site X when requesting the CS. This approach must ensure $\forall\ X, Y, R_X \cap R_Y \neq \varnothing$ such that two conflicting requests can be detected by a site Z ($Z \in R_X \cap R_Y$).

In general, there is a trade-off between synchronization delay and message complexity of distributed mutual exclusion algorithms. No single mutual exclusion algorithm can optimize both synchronization delay and the message complexity. The concept of hybrid mutual exclusion algorithms has been purposed to simultaneously minimize both synchronization delay and message complexity [4]. Sites are divided into groups, and different algorithms are used

to resolve local (intragroup) and global (intergroup) conflicts. By carefully controlling the interaction between the local and the global algorithms, one can minimize both message traffic and synchronization delay simultaneously.

In this paper, we survey and analyze several well-known distributed mutual exclusion algorithms according to their related characteristics as shown in Fig. 1. We also compare the performance of these algorithms by a simulation study. Finally, we present a comparative performance analysis of these algorithms.

The rest of the paper is organized as follows. Section 2 describes the system and the performance models. Sections 3 and 4 discuss non-token-based algorithms based on the *Ricart–Agrawala-type* approach and the *Mackawa-type* approach, respectively. Section 5 presents an algorithm based on the *hybrid* approach. Sections 6, 7, and 8 discuss token-based algorithms based on the *broadcast-based* approach, the *static logical-structure-based* approach, and the *dynamic logical-structure-based* approach, respectively. Section 9 presents a comparative analysis of these algorithms based on the performance. Section 10 contains concluding remarks.

2. SYSTEM AND PERFORMANCE MODELS

In this section, we describe the system and the performance models.

2.1. The System Model

A distributed system consists of N sites, uniquely numbered from 1 to N. Each site contains a process that makes a request to mutually exclusively access the CS. This request is communicated to other processes. Message propagation delay is finite but unpredictable. The communication network is assumed to be reliable (i.e., messages are neither lost nor duplicated and are transmitted error-free), and sites do not crash. There is only one CS in the system, and any process currently in the CS will exit in finite time. Moreover, a site cannot issue another request until the current request is granted, and the process itself exits the CS.

2.2. The Performance Model

The operation of mutual exclusion algorithm is very complex and is quite difficult to analyze mathematically. Analytic performance study of mutual exclusion algorithms is intractable due to the rapid growth of the cardinality of the state space with the number of the sites in the system. Therefore, we have studied the performance of the distributed mutual exclusion algorithms using simulation techniques. The performance model used in this paper is similar to the one used in [18, 22]. We assume that requests for CS execution arrive at a site according to the Poisson distribution with parameter λ. Message propagation delay between any two sites is a constant (T) times a random number (between 0 and 1) with the uniform distribution. The time taken by a site to execute the CS is constant (E). A site processes the requests for the CS one by one, and there is only one CS in the system. In this simulation study, the following two performance measures are considered: (1) message traffic: the average number of messages exchanged among the sites per CS execution; (2) time delay: The average time delay in granting the CS, which is the period of time between the instant a site involves mutual exclusion and the instant when the site enters the CS.

Simulation experiments were carried out for a homogeneous system of 21 sites ($N = 21$) for various values of the traffic of CS requests (λ). (The reason that we choose $N = 21$ is because this number satisfies the required properties of some special algorithms [4, 13].) Message propagation delay (T) between the sites was taken as 0.1, and CS execution time (E) was taken as 0.01. The values of the parameters chosen here are consistent with those in [18, 22]. Collected performance measures, namely, the number of messages per CS execution and the delay in granting the CS, are probabilistic in nature. For these measures, we collect the values of these variables for 5000 CS executions. Moreover, the following two cases are specially concerned in the performance study: (1) light traffic: In this case, most of the time only one or no request for the CS is present in the system; (2) heavy traffic: In this case, all the sites will always have a pending request for the CS.

3. ALGORITHMS BASED ON THE RICART–AGRAWALA-TYPE APPROACH

In algorithms based on the *Ricart–Agrawala-type* approach, a requesting site contains the identifiers of those sits which are possibly inside the CS. The simplest initialization of R_X at a site X is to place every site identifier to R_X. A *timestamp* is included in each request to resolve conflicting requests in a logically fully connected network. In this section, we first present Lamport's algorithm [12] since it is the first distributed mutual exclusion algorithm. We then present three algorithms based on the *Ricart–Agrawala-type* approach [3, 19, 23]. These three algorithms have successively reduced message traffic in Lamport's algorithm by deferring reply messages, reducing the size of request sets, and minimizing the initial values of request sets, respectively.

3.1. Lamport's Algorithm

In Lamport's mutual exclusion algorithm, every request message includes a timestamp T. Moreover, every site maintains a *priority queue* Q in which requests are ordered by the timestamps. (Note that this algorithm requires messages to be delivered in the order they were sent.) When site X invokes mutual exclusion, it adds its request to Q_X and sends a Request(T_X, X) message to every site (i.e., $|R_X| = N - 1$). When site Y receives the Request message, it returns a timestamped Reply message and adds site X's request to Q_Y. Site X can enter the CS when the following two conditions are satisfied: (1) site X's own request is in

front of Q_x; (2) site X has received a message from every site with a timestamp larger than (T_X, X).

Since Lamport's algorithm assumes that between any pair of sites, messages are delivered in the order in which they are sent, the second condition guarantees that site X has learned about all the requests that preceded its current request. Based on the total order defined by the timestamps, the first condition will permit one and only one site (i.e., the site with the smallest timestamp among current requesting sites) to enter the CS.

To release the resource, site X removes its request from Q_X and sends timestamped Release messages to all other sites. When site Y receives site X's Release message, it removes site X's request from Q_Y. Lamport's algorithm requires $3 * (N - 1)$ messages per CS execution.

3.2. Ricart and Agrawala's Algorithm

Ricart and Agrawala's algorithm [19] reduces the message traffic over Lamport's algorithm [12] by using an *implicit Release message* strategy. In this algorithm, when site X wants to enter the CS, it sends a Request mesage to all the sites. Site X can defer its reply to any other site Y whose request has a lower priority than site X's request until site X finishes execution of the CS. Site X enters the CS after it has received Reply messages from all the sites. When site X releases the CS, it sends Reply messages to all the deferred requests. Therefore, when site Y receives a Reply message from site X, the Reply message implies that site X has finished execution of the CS. In this algorithm, R_X at each site X contains the identifier of every site; therefore, this algorithm requires $2 * (N - 1)$ messages per CS execution.

3.3. Carvalho and Roucairol's Algorithm

Carvalho and Roucairol's algorithm [3] avoids some unnecessary Request and Reply messages in Ricart and Agrawala's algorithm [19] by using an *implicit no Request message* strategy. In this algorithm, if site X has not received a Request message from site Y since site X executed the CS last time, it implies that site Y has given its implicit permission to site X. Therefore, site X does not have to ask for permission from all such sites, and message traffic is reduced. This is achieved by keeping a boolean array A that is dynamically updated at each site such that it records the identifiers of those sites which are not requesting the CS. The number of the messages exchanged per CS execution in this algorithm is between 0 and $2 * (N - 1)$.

3.4. Singhal's Dynamic Information Structure Algorithm

Singhal's dynamic information structure algorithm [23] reduces message traffic by cleverly initializing the information structure and updating it as the algorithm evolves. The information structure at site S includes a request set R_S (as defined before) and an inform set I_S that records the identifiers of those sites to which Reply messages are to be sent after the execution of the CS. In this algorithm, the request set is set to $R_S = \{1, 2, ..., (S - 1)\ S\}$, and the inform set is set to $I_S = \{S\}$, for $1 \leq S \leq N$.

When site S invokes mutual exclusion, it sends Request messages to sites whose identifiers are in a dynamically changing request set R_S as opposed to all the sites as in Ricart and Agrawala's algorithm [19]. Therefore, the algorithm is *dynamic* because the request set is dynamically changed such that the request set records only the identifiers of those sites which are possibly inside the CS.

Upon receiving the Request message, site Y takes different actions depending on its current state. (1) If it is not requesting or its request has a lower priority than site S's request, it responds with a Reply message immediately and updates $R_Y = R_Y \cup \{S\}$. Moreover, in the latter case, if site Y has not requested permission from site S, it sends a Request message to site S. (2) If its request has a higher priority than site S's requests, it defers its reply and updates $I_Y = I_Y \cup \{S\}$. When site S receives a Reply message from site Y, it updates $R_S = R_S - \{Y\}$. Site S enters the CS when $R_S = \phi$. After exiting the CS, site S sends a Reply message to every site Y whose identifier is in I_S, removes site identifier Y from I_S and adds site identifier Y to R_S. As the algorithm executes, R_S is dynamically changed; however, the condition ($S \in R_Y$ or $Y \in R_S$) is always satisfied. The number of the messages exchanged per CS execution in this algorithm is also between 0 and $2 * (N - 1)$, and the average number of the messages exchanged in light traffic is $(N - 1)$.

3.5. A Comparison of Performance

Since Lamport's algorithm requires more messages than the other three algorithms, and Carvalho and Roucairol's algorithm has applied a similar strategy to reduce message traffic as Singhal's algorithm, we only do the simulation study of Ricart and Agrawala's algorithm [19] and Singhal's algorithm [23]. (Note that the boolean array **A** used in Carvalho and Roucairol's algorithm has the similar function as the request set R in Singhal's algorithm.) Figure 2a shows a comparison of message traffic between these two algorithms [19, 23]. In light traffic ($0 \leq \lambda \leq 0.3$), Singhal's algorithm needs about $(N - 1)$ messages per CS execution. As the traffic is increased, the number of the sites possibly requesting the CS is increased in Singhal's algorithm. In heavy traffic, both algorithms need $2 * (N - 1)$ messages. Figure 2b shows a comparison of time delay between these two algorithms. In heavy traffic, Singhal's algorithm has longer time delay than Ricart and Agrawala's algorithm, since all of the needed Request messages may not be sent out simultaneously.

4. AN ALGORITHM BASED ON THE MAEKAWA-TYPE APPROACH

In algorithms based on the *Maekawa-type* approach, the request set of a site X contains the identifiers of the sites which are exclusively locked by site X when requesting

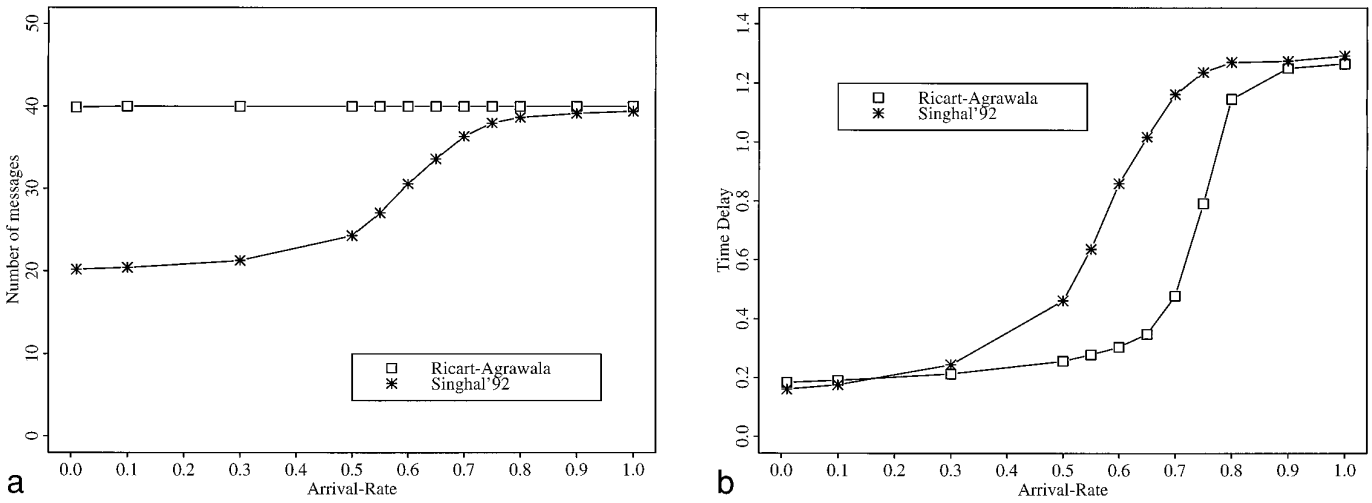

FIG. 2. Comparison of algorithms based on the *Ricart–Agrawala-type* approach: (a) message traffic; (b) time delay.

the CS [13]. To ensure that such an algorithm works correctly, the conditions $\forall\ X, Y, R_X \cap R_Y \neq \emptyset$ must be satisfied such that two conflicting requests can be detected by a site Z ($Z \in R_X \cap R_Y$). A centralized control algorithm [2] is one of the special case of *Maekawa-type* algorithms as pointed out in [24]. In this section, we discuss Maekawa's algorithm [13].

In Maekawa's algorithm [13], a site S must exclusively lock every site whose identifier is in its request set before site S executes the CS. In this algorithm, the request set R_S of each site S has the following properties (where $N = K * (K - 1) + 1$ and $K \in I^+$): (1) $R_S \cap R_Y \neq \emptyset, S \neq Y, 1 \leq S, Y \leq N$; (2) $S \in R_S, 1 \leq S \leq N$; (3) $|R_S| = K, 1 \leq S \leq N$; (4) Any $Y, 1 \leq Y \leq N$, is contained in K of R_S's, $1 \leq S \leq N$.

Consequently, every pair of sites has a common site which mediates conflicts between the pair. When $N = 7$ ($K = 3$), an example of the request sets is as follows: $R_1 = \{1, 2, 3\}$, $R_2 = \{2, 4, 6\}$, $R_3 = \{3, 5, 6\}$, $R_4 = \{1, 4, 5\}$, $R_5 = \{2, 5, 7\}$, $R_6 = \{1, 6, 7\}$, and $R_7 = \{3, 4, 7\}$.

When site X invokes mutual exclusion, it sends Request messages to every site whose identifier is in R_X. Upon receiving a Request message from site S takes actions depending on the following three cases: (1) if site S has not granted its permission (i.e., a Reply message) to some other site, site S grants its permission to site X; (2) if site S has granted its permission to some other site Y and site X's request has lower priority than any other request received at site S, site S sends a Failed message to site X so that site X can relinquish any received permission; (3) it site S has granted its permission to some other site Y and site X's request has higher priority than all other requests received at site S, site S sends an Inquire message to site Y, provided site S has not sent this message to site Y before. Moreover, in cases (2) and (3), site S adds site X's request to Q_S. In case (3), site S also adds site X's request to another set NextQ_S to avoid deadlock [5].

Upon receiving an Inquire message from site S, site Y takes actions depending on the following three cases: (1) if site Y has received a Failed message, site Y sends a Relinquish message to yield the permission; (2) if site Y has received all needed Reply messages, site Y ignores the Inquire message; (3) otherwise, site Y adds S to an InquireSite set. Upon receiving a Failed message from site W, site Y adds W to a FailedFrom set, and sends a Relinquish message to every site whose identifier is in the InquireSite set. After finishing execution of the CS, site X sends Release messages to unlock sites whose identifiers are in R_X. Upon receiving a Relinquish message or a Release message, site S grants its permission to the site P whose request has the highest priority in Q_S. Moreover, site S sends a Failed message to every site in (NextQ_S − $\{P\}$) to avoid deadlock [5]. In the case that a Relinquish message from site Y is received, site S adds site Y's request to Q_S.

Maekawa's algorithm is prone to deadlock because a site is exclusively locked by one requesting site at a time, and requests can arrive in any order. (Note that site X is *locked* by site Y if site X grants permission to site Y.) A possible deadlock occurs in the case when site X's request arrives at site Y ($Y \in R_X$) and site X's request has a priority higher than site Y's current locking site Z's request. In this case, site Y will send an Inquire message to the current locking site Z and waits for a Release or a Relinquish message. (Note that site Z will yield the permission if site Z has received any Failed message.) Consequently, deadlock is resolved by requiring a site to yield if the timestamp of its request is larger (i.e., younger) than the timestamp of any other request. When no deadlock occurs, K Request-Reply-Release messages are needed. When a deadlock occurs, extra messages (Failed, Inquire, and Relinquish) are needed to resolve the deadlock. This algorithm requires $O(\sqrt{N})$ messages per CS execution because the size of a request set is $\sqrt{N}$. The performance study of Maekawa's algorithm will be discussed in Section 5.2, since we will compare it with Chang *et al.*'s hybrid algorithm [4]. A generalized *Maekawa-type* algorithm with several

variants of initialization of request sets has been discussed by Sanders [21].

5. AN ALGORITHM BASED ON THE HYBRID APPROACH

Although several distributed mutual exclusion algorithms have been proposed to minimize either message traffic or time delay, none of them can minimize both at the same time. Most algorithms have reduced message traffic by increasing time delay. For example, Maekawa's algorithm [13] reduces message traffic to $O(\sqrt{N})$; however, it has longer time delay in successive executions of CS as compared to Ricart–Agrawala's algorithm [19] (whose message traffic is $O(N)$). To minimize both message traffic and time delay at the same time, we look for a hybrid approach to mutual exclusion in which two algorithms are combined such that one minimizes message traffic and the other minimizes time delay. In this section, we discuss Chang *et al.*'s hybrid algorithm [4].

5.1. Chang et al.'s Hybrid Algorithm

In Chang *et al.*'s hybrid approach [4], sits are divided into groups, and different algorithms are used to resolve local (intragroup) and global (intergroup) conflicts. That is, sites use one local algorithm to resolve conflicts with sites in the same group and use a different global algorithm to resolve conflicts with sites in other groups. A request set R_S consists of two sets: a local set L_S and a global set G_S. A local set L_S is used to enforce mutual exclusion locally (i.e., within the group) and a global set G_S is used to enforce mutual exclusion globally (i.e., among the groups). The choice of L_S and G_S must satisfy the following conditions: (1) $L_S \cap L_Y \neq \emptyset$, $\forall\ S, Y \in \text{Group}_K$; (2) $L_S \cap L_Y = \emptyset$, $\forall\ S \in \text{Group}_K$, $\forall\ Y \in \text{Group}_P$, $(K \neq P)$; (3) $G_S = G_Y$, $\forall\ S, Y \in \text{Group}_K$; (4) $G_S \cap G_Y \neq \emptyset$, $\forall\ S \in \text{Group}_K$, $\forall\ Y \in \text{Group}_P$, $(K \neq P)$.

To satisfy conditions (1) and (2), the local sets in each Group_K of $|\text{Group}_K|$ sites are initialized according to the local algorithm. To satisfy conditions (3) and (4), sites in the a local representative set are determined first. A local representative set (LRS) of Group_K is a subset of sites in Group_K. These are the sites which grant global permission to sites in other groups as far as Group_K is concerned. After sites in LRS_K have been determined in each Group_K, the global set G_S at site S can be determined. Note that as far as a global algorithm is concerned, it treats each group as a site and the LRS acts as the site identifier for a group. The global set G_S contains a subset of LRS_j's, $1 \leq j \leq g$, and the number of LRS's in each G_S is determined according to the global algorithm.

Several design issues must be addressed to control interaction between the local and global algorithms. The design issues includes modes of release, semantics of global messages and modes of request. The *release local sites first* mode means that requesting sites in the same group as the site finishing execution of CS have a higher priority in getting global permission than requesting sites in other groups. The *requesting group* semantics means that the same reply messages (granted or rejected) are sent to all requesting sites in the same group (assuming that all requesting sites in the same group have the same priority). A hybrid mutual exclusion algorithm using the *release local sites first* mode, the *requesting group* semantics, and the requesting sequence in which local competition is followed by global competition has been found to be an efficient way to control the interaction. Since the global permission can be used for the successive execution of CS in the same group before it is released to other groups, message traffic and time delay can be reduced at the same time. Chang *et al.*'s hybrid algorithm uses Singhal's algorithm [23] as the local algorithm and Maekawa's algorithm [13] as the global algorithm under the above combination of these design issues.

In order to avoid starvation resulting from too many successive executions of CS in the same group, a local counter LC_S at site S keeps track of the number of successive executions of CS in its group. LC_S is set to 0 initially and is increased by 1 when site S finishes execution of the CS. This value is passed around in a local Reply message and is used as a flag: (1) the condition $(LC_S \bmod M) \neq 0$, where M is the upper bound of successive executions of CS in the same group, indicates implicit global permission; (2) the condition $(LC_S \bmod M) = 0$ indicates the absence of implicit global permission. When site S releases global permission, LC_S is increased by t such that $((LC_S + t) \bmod M) = 0$, $0 \leq t < M$. A site is granted all global permission when any of the following two cases satisfies: (1) when the requesting site S has received a global Reply message from every site whose identifies is in G_S; (2) when the requesting site S receives a local Reply message which denotes implicit global permission.

5.2. A Comparison of Performance

We first discuss the performance of Maekawa's algorithm as shown in Fig. 3. In light traffic, Maekawa's algorithm needs 12 messages (i.e., $3 * (K - 1)$, where $N = K * (K - 1) + 1$), $N = 21$ and $K = 5$). That is, one Request message, one Reply message and one Release message are exchanged between a requesting site X and each of members in its request set R_X. As the traffic is increased, many conflicting requests can occur; the algorithm uses Inquire and Relinguish messages to resolve possible deadlock. Therefore, as the traffic is increased, the number of messages exchanged is increased. While in heavy traffic, since the logical clock at each site will be updated frequently and quickly, requests with a higher priority arrive before requests with a lower priority at a site most of the time. Therefore, many Failed messages respond to the incoming requests. This algorithm requires 16 messages (i.e., $4 * (K - 1)$) in heavy traffic. Moreover, as the traffic is increased, longer time delay is needed for a site to enter

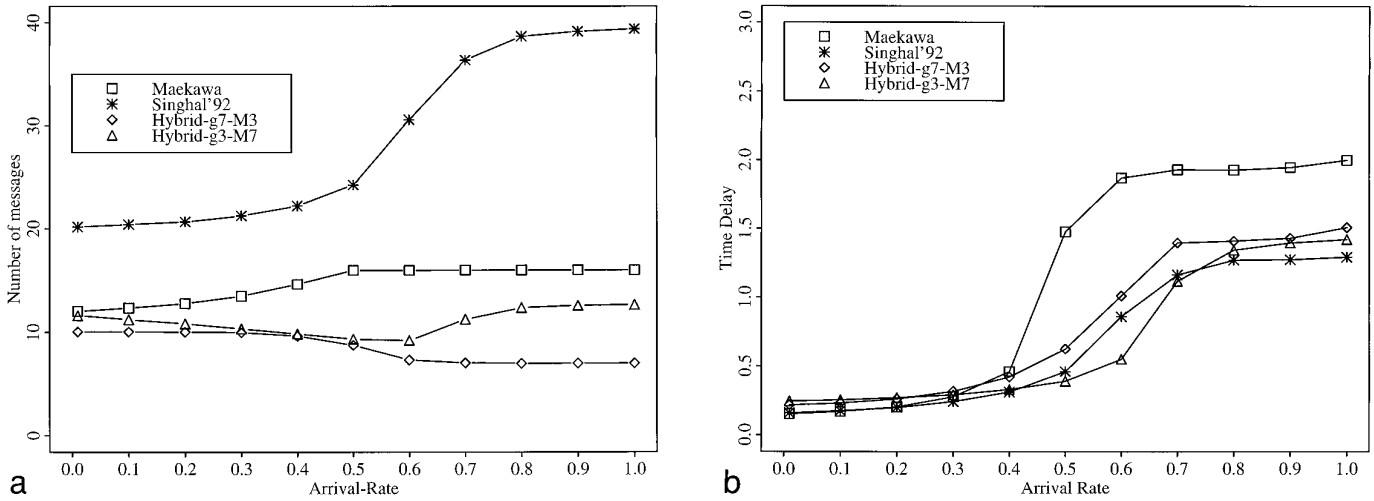

FIG. 3. Comparison among Singhal's dynamic information structure and Maekawa's and Chang *et al.*'s hybrid algorithms: (a) message traffic; (b) time delay.

the CS. Since a site is exclusively locked by some other site, a site X sends a Release message to every member of R_X to release the permission after site X exits the CS, which introduces one more time delay for the next requesting site to enter the CS as compared to Fig. 2b.

Next, we make a comparison of Maekawa's, Singhal's, and Chang *et al.*'s hybrid algorithms [13, 23, 4]. In this simulation, we consider two different combinations of g and M when $N = 21$: *Hybrid-g7-M3* and *Hybrid-g3-M7*, where g is the number of groups and M is the upper bound of successive executions of CS in the same group. For the case of *Hybrid-g7-M3*, it means that 21 sites are divided into 7 groups. Since when $g = 7$, there are three sites per group; that is why we let $M \leq 3$. In a similar way, when $g = 3$, we let $M \leq 7$. Figures 3a and 3b show the message traffic and time delay as a function of the arrival rate, respectively. When $g = 7$ and $M = 3$, message traffic in Chang *et al.*'s hybrid algorithm can be reduced up to 56% as compared to Maekawa's algorithm, and has much less message traffic than Singhal's algorithm. This reduction is resulted from the successive executions of CS in the same group. When $g = 7$ and $M = 3$, time delay in Change *et al.*'s hybrid algorithm can be reduced up to 25% as compared to Maekawa's algorithm. The reason for this reduction is that the hybrid algorithm uses Singhal's algorithm within a group to reduce the long time delay.

One interesting observation is that the number of messages exchanged in the hybrid algorithm is not monotonously increased as the arrival rate is increased. The number of messages needed in the case of *Hybrid-g7-M3* is decreased from 10.05 to 7 as the arrival rate is increased from 0.01 to 1, and the number of message needed in the case of *Hybrid-g3-M7* is changed from 11.61 to 9.21 and then from 9.21 to 12.65 as the arrival rate is increased. Recall that the number of messages required in Singhal's algorithm is between 20 and 40, while it is between 12 and 16 in Maekawa's algorithm. That is, the range of the number of messages needed in Singhal's algorithm is much larger than that in Maekawa's algorithm. Moreover, when g is increased, the hybrid algorithm behaves more like the global algorithm and inherits the advantages of the global algorithm (i.e., Maekawa's algorithm). Therefore, the behavior of the case of *Hybrid-g7-M3* inherits more properties from Maekawa's algorithm. As the arrival rate is increased, many requesting sites in the same group can make use of the implicit global permission to enter the CS, resulting in reducing the average number of messages exchanged. Consequently, the number of messages needed in the case of *Hybrid-g7-M3* is decreased as the arrival rate is increased. To the contrary, as g is decreased, the hybrid algorithm behaves more like the local algorithm, and inherits more characteristics of the local algorithm (i.e., Singhal's algorithm). Therefore, the behavior of the case of *Hybrid-g3-M7* inherits more properties from Singhal's algorithm. When in the light traffic, the number of messages needed in Singhal's algorithm is around 20; therefore, the number of messages in the case of *Hybrid-g3-M7* is decreased due to the effect of Maekawa's algorithm and M. However, when in the heavy traffic, the number of messages in Singhal's algorithm is increased largely and quickly from 20 to 40; therefore, the number of messages in the case of *Hybrid-g3-M7* is increased.

Based on the same reason, for the simulation result of time delay, the behavior of the case of *Hybrid-g7-M3* inherits more properties from Maekawa's algorithms while the behavior of the case of *Hybrid-g3-M7* inherits more properties from Singhal's algorithm. Therefore, the case of *Hybrid-g7-M3* takes longer time delay to enter the CS than the case of *Hybrid-g3-M7* in heavy traffic. Both of them take shorter time delay to enter the CS than Maekawa's algorithm and longer time delay to enter the CS than Singhal's algorithm in heavy traffic. While in light traffic, since a site cannot start to acquire global permission until all local permission is obtained, the hybrid algorithm takes a

little bit longer time delay to enter the CS than Maekawa's and Singhal's algorithms.

6. ALGORITHMS BASED ON THE BROADCAST-BASED APPROACH

In algorithms based on the *broadcast-based* approach, the network is logically fully connected and requests are sent out to other sites in parallel. In this section, we discuss three algorithms based on the *broadcast-based* approach [20, 22, 25]. Some of these algorithms include a queue in the Token message to store those waiting requests [25], while some of them do not include a queue in the Token message [20, 22]. Some of them send requests to sites whose identifiers are in a *static* request set [20, 25], while some of them send requests to sites whose identifiers are in a *dynamic* request [22].

6.1. Suzuki and Kasami's Algorithm

In Suzuki and Kasami's algorithm [25], a requesting site sends Request messages to all other sites. To detect an out-of-date request message, a Seq_S array (with N entries) at each site S and a TSeq array (with N entries) in the Token message are required. Seq_S[I] at site S, $1 \le I \le N$, records the number of CS requests made by site I as far as site S knows; TSeq[I] records the number of CS executions finished by site I. When a site S invokes mutual exclusion, it increases its sequence number, $Seq_S[S]$, by one and includes this sequence number in each Request message. Upon receiving site S's Request($SN = Seq_S[S], S$) message, a site Y can tell whether this request is out-of-date by testing the following conditions (1) if $SN > Seq_Y[S]$, this is a new Request message, and site Y then updates $Seq_Y[S] = SN$ (= $Seq_S[S]$); (2) otherwise, this is an old Request message, and site Y should ignore it.

A queue TQ included in the Token message contains the waiting requests. After finishing execution of the CS, site Z checks any new waiting requests by testing the following conditions, $1 \le I \le N$: (1) if Seq_Z[I] $\le$ TSeq[I], the value of $Seq_Z[I]$ is out-of-date; (2) if $Seq_Z[I] >$ TSeq[I] (i.e., $Seq_Z[I]$ = TSeq[I] + 1), site I has finished the (TSeq[I])th CS execution and is requesting the ($Seq_Z[I]$)th (= $Seq_I[I]$) CS execution. These new waiting requests I are then added to TQ provided these requests are not already in TQ. Site Z then sends the Token(TQ, TSeq) message to the site whose identifier is the first entry of TQ. The algorithm requires 0 or N messages per CS execution. Nishio *et al.* [15] have proposed techniques to make Suzuki and Kasami's algorithm fault-tolerant to a site failure and token loss.

6.2. Ricart and Agrawala's Token-Based Algorithm

Ricart and Agrawala's token-based algorithm [20] does not include the queue TQ in the Token message as opposed to Suzuki and Kasami's algorithm [25]. The queue TQ used in Suzuki and Kasmmi's algorithm is used to record those waiting requests and it simplifies the determination of the next site to get the token. However, the use of TQ is redundant since those waiting requests can be determined by the relationship between Seq and TSeq arrays, and the order of the next site to hold the token can be determined by several arbitration rules [22]. A site I satisfying $Seq_S[I] >$ TSeq[I] at site S which holds the token, $1 \le I \le N$, implies that site I is waiting to execute the CS. Ricart and Agrawala's token-based algorithm uses a round-robin arbitration rule and also requires 0 or N messages per CS execution.

6.3. Singhal's Heuristically Aided Algorithm

Singhal's heuristically aided algorithm [22] makes use of *state information*, which is defined as the set of states of mutual exclusion processes in the system. A site can be in the state of "requesting its CS" (denoted as "R"), "executing its CS" (denoted as "E"), "not requesting its CS" (denoted as "N"), or "not requesting its CS but holding the token" (denoted as "H"). The state of the system is the set of the states of all the sites. A site maintains an array SV (called *state vector*) to store the state of sites of the system. A TSV array (i.e., token state vector) is included in the Token message to refresh the information in a site's state vector.

In this algorithm, when a site invokes mutual exclusion, it uses a heuristic to guess from its available state information what sites of the system are probably holding or are possibly to have the token and sends token request messages only to those sites rather than to all of the sites. The heuristic is as follows: All the sites for which the state vector SV entries are "R" (i.e., Requesting the CS) are in the probable set. (Note that *sequence numbers* are still used in order to distinguish old/new messages.)

Initially, every site S sets $SV_S[I]$ = "R", $1 \le I \le (S - 1)$, $SV_S[I]$ = "N", $S \le I \le N$, and site 1 holds the token (i.e., $SV_1[1]$ = "H"). The state information will be updated in the following three cases: (1) When a site completes the execution of its CS, it updates its state vector and token vector by using *update rules*. The update rules essentially compare the state vector at the site performing updates with the token vector to determine which one has more current information about the state of the sites and restore the out-dated entries of vectors with more current ones. (2) When a site wants to send the token to one of other requesting sites, it uses *arbitration rules* to determine which of many requesting sites should get the token next. At the same time, the related state information is updated. (3) When a site receives Request messages, its request message handler will update its state vector. The number of the messages, exchanged per CS execution in this algorithm is also between 0 and N, and the average number of the messages exchanged in light traffic is $(N + 1)/2$.

6.4. A Comparison of Performance

Obviously, Suzuki and Kasami's algorithm and Ricart and Agrawala's token-based algorithm will have the same

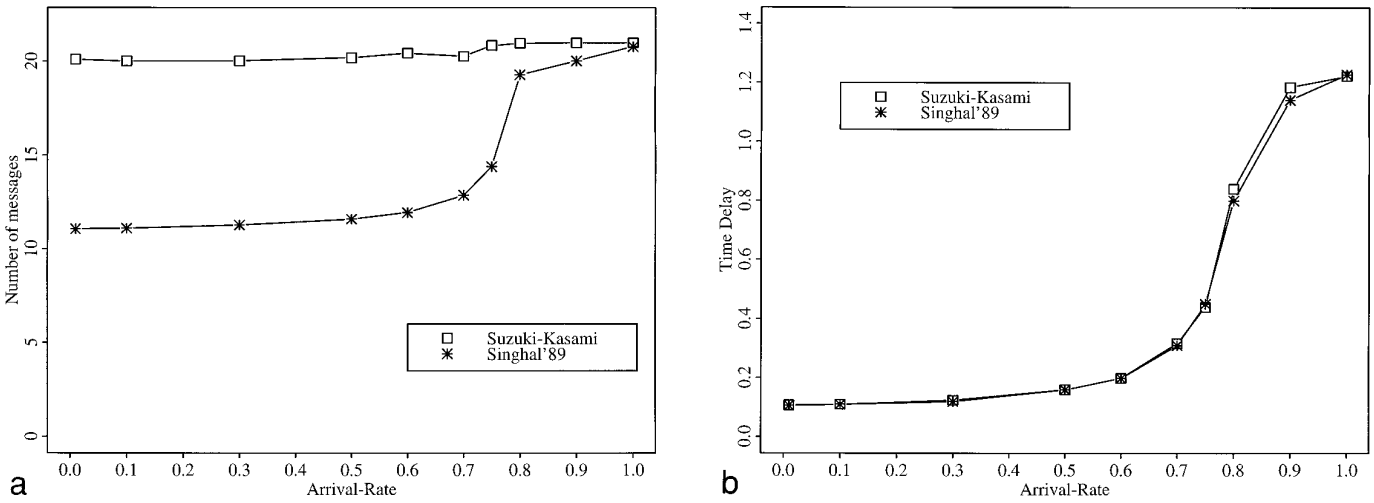

FIG. 4. Comparison of algorithms based on the *broadcast-based* approach: (a) message traffic; (b) time delay.

performance; therefore, we only do the simulation study of Suzuki and Kasami's algorithm [25] and Singhal's algorithm [22]. Figure 4a shows a comparison of message traffic between these two algorithms [22, 25]. In light traffic, Singhal's algorithm needs about 54% $*$ N messages per CS execution. As the traffic is increased, the number of the sites possibly holding the token is increased in Singhal's algorithm. In heavy traffic, both algorithms need N messages. Moreover, both algorithms have similar time delay as shown in Fig. 4b.

7. AN ALGORITHM BASED ON THE STATIC LOGICAL-STRUCTURE-BASED APPROACH

In token-based algorithms based onthe *static logical-structure-based* approach, sites are organized in a special configuration (e.g., tree). Requests must be sequentially propagated through the paths between the requesting site and the site holding the token, and so does the token. Moreover, the direction of edges is dynamically updated such that it always leads to the site holding the token and no cycle exists in the configuration. In this section, we discuss Raymond's algorithm [16].

7.1. Raymond's Algorithm

In Raymond's algorithm [16], the network topology is a tree, and the root holds the token. Every site communicates only with its neighboring sites and holds information only about its neighbors. At every site S, a variable Holder_S records the identifier of its neighbor on the path leading to the site holding the token, and a local FIFO (first-in-first-out) queue, Q_S, records the identifiers of its requesting neighbors. (Note that when $\text{Holder}_S = S$, site S holds the token.)

When a site S which does not hold the token invokes mutual exclusion, it first adds its request to the end of Q_S and then sends a Request message to Holder_S (provided it has not sent out a Request message for a waiting request in Q_S). When a site Y which does not hold the token receives a Request message from one of its neighbors, it first adds the identifier of this neighbor to the end of Q_Y and then sends a Request message to Holder_Y (provided it has not sent out a Request message for a waiting request in Q_Y).

A sequence of Request messages is sent between the requesting site and the site holding the token (along the path constructed by Holder's) until a Request message arrives at the site holding the token. Then, the token is passed along the same path in the reverse direction. As the token passes by, the direction of the edges traveled by the token is reversed such that every path always leads to the site holding the token. When site X receives the token, it sends the token to the site whose identifier is the first entry of Q_X, which is either itself or one of its requesting neighbors Y, and removes this request from Q_X. In the case where the first waiting request is not site X itself and $Q_X \neq \emptyset$, site X will send another Request message to its requesting neighbor Y to ask for the return of the token. Raymond's algorithm requires $O(\log N)$ messages per CS execution.

7.2. The Performance of Raymond's Algorithm

In this simulation, we consider three special cases of a tree structure: a straight line, a star, and a radiating star. (Note that a radiating star is a tree in which the degree of each nonleaf node is the same.) From Fig. 5a, we observe that among these three tree topologies, a straight line has the largest number of messages exchanged and a star has the smallest number of messages exchanged in light traffic. This can be explained as follows. Since messages are sent along the path constructed by Holder's and the topology constructed by Holder's is a tree, the number of messages exchanged in light traffic is proportional to the number of

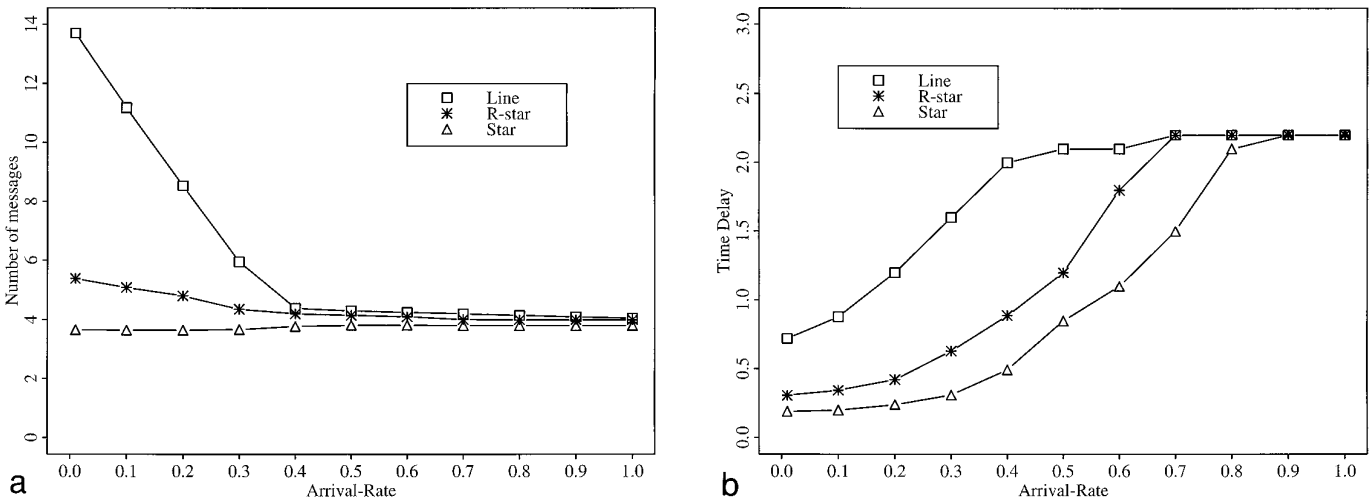

FIG. 5. Performance of Raymond's algorithm with three different topologies: (a) message traffic; (b) time delay.

hops in the longest path of the tree. Moreover, since the number of hops in the longest path of an arbitrary, tree is proportional to $O(\log N)$ in average, the number of messages exchanged in light traffic is also proportional to $O(\log N)$ in average. From Fig. 5b, we observe that a straight line has the largest time delay and a star has the shortest time delay in light traffic. This can be explained by the same reason as explained above. This algorithm needs longer time delay as compared to algorithms studied in Fig. 4b, because messages are propagated serially.

As the arrive rate is increased, the average number of messages exchanged per CS execution is decreased in heavy traffic. This message reduction is due to the following reasons: (1) Only one Request message is sent out for more than one incoming request, which can be controlled by testing the size of Q. (2) When a requesting site S receives the token from its neighbor Y, its other neighbors whose requests arrive before site Y's request can use the token before the token is returned to site Y.

8. ALGORITHMS BASED ON THE DYNAMIC LOGICAL-STRUCTURE-BASED APPROACH

In algorithms based on the *dynamic logical-structure-based* approach, the network is logically fully connected and a dynamic logical tree is maintained such that the root is always the site which will hold the token in the near future. That is, the root is the last site to get the token among the current requesting sites when no message is in transit. A request is sequentially forwarded along a path (in the logical tree) to the root; however, the token is directly sent to the next requesting site to excute the CS. In this section, we first discuss two algorithms based on the *request forwarding approach* [14, 26]. Next, we discuss Helary *et al.*'s general scheme for token- and tree-based distributed mutual exclusion algorithms [11], which covers those algorithms [14, 16, 26] based on the static and dynamic *logical-structure-based* approaches.

8.1. Trehel and Naimi's Algorithm

In Trehel and Naimi's algorithm [26], two special variables are used at each at each site S: Last_S and Next_S. Last_S concerns about where site S sends Request messages; Next_S indicates the next site to enter the CS. A site S satisfying $\text{Last}_S = S$ (the root) implies that site S is the last site to get the token among the current requesting sites when no message is in transit; moreover, it holds the token if no site is requesting. The queue is distributed in the algorithm; i.e., every requesting site S only records the identifier of the requesting site next to it to get the token in a variable Next_S. If site S is not requesting, $\text{Next}_S = 0$. On the other hand, every requesting site S except the last site in the distributed queue will have $\text{Next}_S \neq 0$.

When a site Y invokes mutual exclusion, it sends a Request(Y) message to the site possibly holding the token (i.e., Last_Y). Upon receiving the Request(Y) message, a site X then forwards this Request(Y) message to Last_X in the following cases: (1) site X is not requesting and it does not hold the token; (2) site X is requesting and its queue is not empty. Otherwise, site X records $\text{Next}_X = Y$. After that, site X updates $\text{Last}_X = Y$ since site Y will have the token in the near future. Finally, the Request(Y) message will be forwarded to the root in finite time and site Y then becomes the new root.

Due to the distributed queue data structure, the Token message does not have to contain the queue as in [22, 25]; therefore, this algorithm also simplifies the data structure in the Token message. This algorithm requires $O(\log N)$ messages per CS execution; however, a request can be forwarded up to $(N - 1)$ sites in the worst case.

8.2. Trehel and Naimi's Improved Algorithm

In Trehel and Naimi's improved algorithm [14], at every site S, instead of a variable Next_S, a local queue Q_S is used to store the waiting requests. When a site S is requesting or executing the CS, it adds all incoming requests to Q_S

(instead of forwarding these requests to Last_S immediately). After finishing execution of the CS, site S sets Last_S to the identifier of the last entry in Q_S and includes Q_S in the Token message. A site Y receiving the Token(Q_S) message adds the entries in Q_S to the front of Q_Y. This algorithm requires $O(\log N)$ messages with a smaller multiplying constant per CS execution as compared to their original algorithm presented in Section 8.1. However, the size of the queue in the Token message and at each site can be as big as $(N-1)$. A distributed queue data structure may be degenerated to a single queue in the worst case.

8.3. Helary et al.'s General Scheme

Helary *et al.* has proposed a general information structure and the associated generic algorithm for token- and tree-based distributed mutual exclusion algorithms [11]. Their information structure contains a dynamic rooted tree structure logically connecting the sites involved in the system, and a *behavior* attribute (*transit* or *proxy*) dynamically assigned to each site. A variable Father_S indicates, according to current site S's knowledge, the site through which the token can be reached, which is the same as the variable Holder in Raymond's algorithm and the variable Last in Trehel and Naimi's algorithms. All Father variables are set in such a way they define a rooted tree stricture over the sites with the token located at the root. When a site Y invokes mutual exclusion, it sends a Request(Y) message to Father_Y and sets a variable $\text{Mandator}_Y = Y$ to record that it is the one really asking for the token. Upon receiving a Request(Y) message, site X can react to this message with for requesting the token either for itself or others, it adds incoming requests to its local queue.)

The *transit* behavior means that site X which does not hold the token will forward only the message Request(Y) to Father_X. If site X with a *transit* behavior is the token owner, site X will send a Token(Nil) message to site Y, where the Nil information in the Token message implies that it is not necessary to return the token back to the original token holder. In both cases, site X then sets $\text{Father}_X = Y$. That is, upon receiving a request message, a *transit* site behaves like the way in Trehel and Naimi's algorithm [26].

The *proxy* behavior means that site X which does not hold the token considers Y as its *mandator* (by setting $\text{Mandator}_X = Y$) and requests the token (to Father_X) for itself. That is, upon receiving a request message, a *proxy* site which does not hold the token behaves like the way in Raymond's algorithm [16]. If site X with a *proxy* behavior is the token owner, it will send a Token(X) message to site Y, where the site identifier X in the Token message implies that the token must be returned back to site X. The variable Father_X is not changed at a site with the *proxy* behavior. (Note that in Raymond's algorithm, a token owner does not ask for the return of the token if its waiting queue is empty. Moreover, in Raymond's algorithm, a site reverses the direction of an incoming edge where the request passes by to an outgoing edge, when the token is sent out.)

Upon receiving a Token(Z) or Token(Nil) message from site K, where Z is a site identifier which is called the token *lender*, site X takes different actions in the following three cases: (1) $\text{Mandator}_X = \text{nil}$; (2) $\text{Mandator}_X = X$; (3) $\text{Mandator}_X \neq X$.

(1) $\text{Mandator}_X = \text{nil}$: In this case, site X does not ask for the token for itself or its neighbors. Site X just keeps the token. (2) $\text{Mandator}_X = X$: In this case, site X is the one really asking for the token. If a Token(Z) message is received, site X sets $\text{Lender}_X = Z$ to record that site Z is the token lender and $\text{Father}_X = K$. In this way, when site X exits the CS, it will send the token back to Lender_X (i.e., site Z) directly. If a Token(Nil) message is received, site X sets $\text{Lender}_X = X$ and $\text{Father}_X = \text{nil}$. In this way, site X keeps the token after it exits the CS. Finally, site X updates $\text{Mandator}_X = \text{nil}$. (3) $\text{Mandator}_X \neq X$: In this case, if a Token(Z) message is received, site X will send a Token(Z) message to Mandator_X and sets $\text{Father}_X = K$. If a Token(Nil) message is received, site X takes two different actions according to its behavior: (i) Site X with a *proxy* behavior sets $\text{Lender}_X = X$, $\text{Father}_X = \text{nil}$, and then sends a Token(X) message to Mandator_X. In this way, site X will get the token back in the future. (ii) Site X with a *transit* behavior sets $\text{Lender}_X = \text{nil}$, $\text{Father}_X = \text{Mandator}_X$, and then sends a Token(Nil) message to Mandator_X. Finally, site X updates $\text{Mandator}_X = \text{nil}$.

In Helary *et al.*'s general algorithm, when every site has a *transit* behavior, the resulting algorithm is a variant of Trehel and Naimi's improved algorithm [14]. In the resulting algorithm, for each request in a waiting queue, a site X forwards the request to Father_X and then updates the variable Father_X, instead of sending the whole waiting queue to the next token-holder in [14], when site X is not requesting the token. On the other hand, when the behavior of every site is *proxy* when it has the token and *transit* otherwise, the resulting algorithm is the same as Raymond's algorithm. (Note that when every site has a *proxy* behavior, the resulting algorithm is a centralized algorithm, in which the direction of paths constructed by Father's will not be changed and the token will always be sent back to the root when a site exits the CS.) The performance of Helary *et al.*'s algorithm really depends on the given topology and every site's behavior.

8.4. A Comparison of Performance

We first discuss the performance of Trehel and Naimi's algorithms [14, 26] and Helary *et al.*'s algorithm with every site a *transit* behavior (denoted as *Helary-transit*) as shown in Fig. 6, where we consider a star topology initially. In light traffic, these three algorithms have a similar number of the messages exchanged that are smaller than four messages. In heavy traffic, the number of messages exchanged is reduced to 2 in Trehel and Naimi's improved algorithm and 2.5 in the *Helary-transit* algorithm. The reason is that the performance in light traffic is proportional to the number of the hops in the longest path constructed by Last's plus one Token message. While in heavy traffic, since every

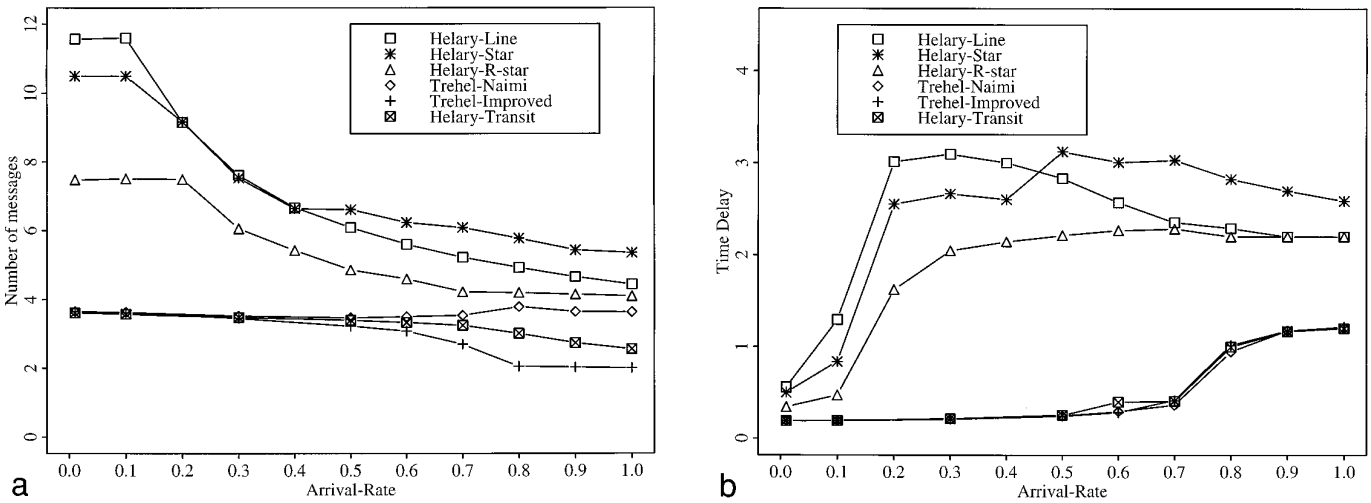

FIG. 6. Comparison of algorithms based on the *dynamic logical-structure-based* approach: (a) average message traffic; (b) time delay.

site will keep updating the latest information about the site possibly holding the token, which implies that the height of the tree is reduced, the average number of the messages exchanged per CS execution is reduced. Depending on how up-to-date information about $Last_S$ at a site S is maintained, these three algorithms have different performance results in heavy traffic. Moreover, these three algorithms have similar time delay.

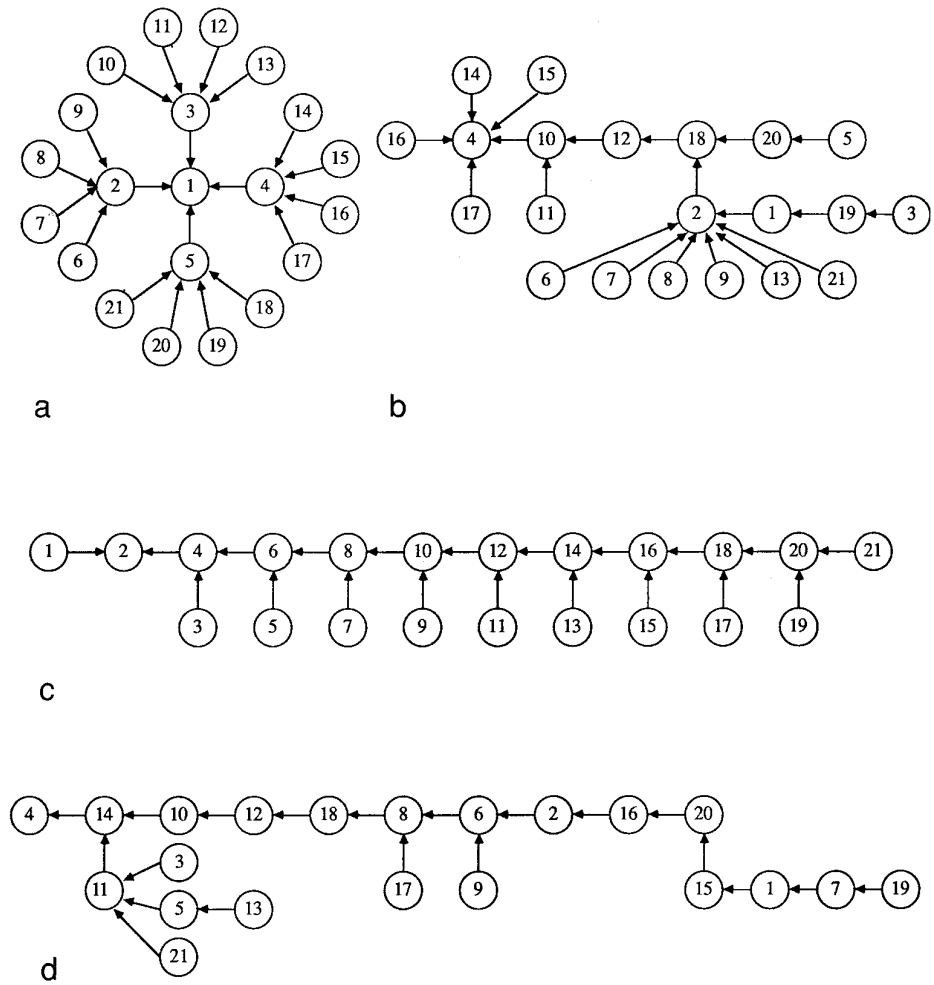

FIG. 7. Tree structure: (a) a radiating star (before); (b) a radiating star (after); (c) a line (after); (d) a star (after).

Next, we discuss the simulation result of Helary *et al.*'s algorithm as shown in Figure 6, where we consider three tree structures: a line, a star and a radiating star, which are denoted as *Helary-line*, *Helary-star*, and *Helary-R-star*, respectively. Moreover, initially, we assume that site 1 holds the token, a site will have a small identifier as it is near the root (site 1), a site with an odd number of site identifier has a *transit* behavior, and a site with an even number of site identifier has a *proxy* behavior. Figure 7a shows the radiating star structure initially, and Figs. 7b, 7c, and 7d show the final tree structures for the given radiating star, line and star, respectively, when 5000 CS executions have been finished and no message is in transit. It is interesting to see that for sites with an even number of site identifiers, they always keep the relative position once requests along the path between them occur. Those final tree structures shown in Figs. 7b–7d explain why *Helary-star* needs the largest number of messages and *Helary-R-star* needs the smallest number of messages among these three topologies in heavy traffic, which also explain the case of time delay. In general, the performance of Helary *et al.*'s algorithm really depends on the given topology and every site's behavior.

9. A COMPARATIVE ANALYSIS

In this section, we present a comparative analysis of these algorithms based on the performance. In general, these algorithms with different approaches have their own merits and limitations as shown in Table I. The control and data structures in algorithms based on the *Ricart–Agrawala-type* are more fully distributed and symmetric than the algorithms based on any other approach. However, algorithms based on the *Ricart–Agrawala-type* usually require more messages. Algorithms based on the *Maekawa-type* require $O(\sqrt{N})$ messages on average, but they have long time delay. Algorithms based on the *hybrid* approach are a compromise between algorithms based on the *Ricart–Agrawala-type* and *Maekawa-type* approaches: Chang *et al.*'s hybrid algorithm has shorter time delay than algorithms based on the *Maekawa-type* approach, and requires fewer messages than algorithms based on the *Ricart–Agrawala-type* approach. Moreover, due to successive executions of CS in the same group, Chang *et al.*'s hybrid algorithm also requires fewer messages than algorithms based on the *Maekawa-type* approach.

Algorithms based on the *broadcast-based* approach require 0 or N messages per CS execution and have simple control and data structures. However, they suffer the problem of heavy traffic congestion in the token site. Algorithms based on the *static logical-structure-based* approach require $O(\log N)$ messages in light traffic and the number of the messages exchanged can be reduced to four messages in heavy traffic. However, algorithms based on the *static logical-structure-based* approach may have long time delay since messages are propagated serially. Algorithms based on the *dynamic logical-structure-based* approach also require $O(\log N)$ messages in light traffic and the number of the messages exchanged can be reduced to two messages in heavy traffic. Algorithms based on the *dynamic logical-*

TABLE I

Class	Approach		Algorithm	Messages (analysis)	Messages[a] ($N = 21$)	Time delay[a] (s)	Note
Non-token-based	Ricart–Agrawala type	S	Ricart *et al.*'s non-token-based (Section 3.2)	$2(N-1)$	40	0.2 .. 1.2	Are most fully distributed and symmetric and have a higher degree of fault tolerance, but need a large number of messages
		D	Singhal's dynamic information structure (Section 3.4)	0 .. $2(N-1)$ $(N-1)$ in L[b]	20 .. 40	0.2 .. 1.3	
	Maekawa type		Maelawa (section 4)	$3(K-1)$ .. $5(K-1)$ $N = K(K-1)+1$	12 .. 16	0.2 .. 2	Needs fewer messages but longer time delay
Hybrid			Chang *et al.*'s hybrid (Section 5.1)	0.44*4$(K-1)$ $N = K(K-1)+1$	7 .. 10 $g = 7$, $M = 3$	0.2 .. 1.5	As M increases, number of messages decreases
Token-based	Broadcast-based	S	Suzuki *et al.* (Section 6.1)	0 or N	20 .. 21	0.1 .. 1.2	Have simple control and data structures but poor fault tolerance
		D	Singhal's heuristically aided (Section 6.3)	0 .. N $(N+1)/2$ in L[b]	11 .. 21	0.1 .. 1.2	
	Logical-structure-based	S	Raymond (Section 7.1)	$O(\log N)$	Line: 4 .. 13.7 RStar: 4 .. 5.4 Star: 3.7 .. 3.8	Line: 0.7 .. 2.2 RStar: 0.2 .. 2.2 Star: 0.2 .. 2.2	Needs longer time delay since messages are sent out serially
		D	1. Trehel *et al.* (Section 8.1)	$O(\log N)$	3.6	0.2 .. 1.2	
			2. Trehel *et al.*'s improved (Section 8.2)	$O(\log N)$ 2 in H[b]	2 .. 3.6	0.2 .. 1.2	May need a queue with size = N
			3. Helary *et al.* (transit behavior) (Section 8.3)	$O(\log N)$	2.5 .. 3.6	0.2 .. 1.2	

Note: M, the upper bound of successive execution of CS in the same group; g, the number of groups; S, static; D, dynamic.
[a] Messages and time delay for simulation results with $N = 21$, $T = 0.1$, and $E = 0.01$.
[b] L*, light traffic; H, heavy traffic.

structure-based approach use two special data structures (a dynamic logical tree and a distributed queue) to reduce messages. Over all, Chang *et al.*'s hybrid algorithm [4] with $g = 7$ (i.e., the number of groups = 7) and $M = 3$ (i.e., the upper bound of successive executions of CS in the same group = 3) has the lowest average message traffic among the non-token-based algorithms. (Note that Chang *et al.*'s hybrid algorithm is based on two non-token-based algorithms.) Trehel and Naimi's improved $O(\log N)$ algorithm [14] has the lowest average message traffic among the token-based algorithms in a fully connected network. Almost all of the algorithms have the similar time delay, except the algorithms based on the *Maekawa-type*, *hybrid*, and *static logical-structure-based* approaches. In the algorithms based on the *Maekawa-type* [13, 21] and *hybrid* approach [4], a site is exclusively locked by one requesting site; therefore, one more time delay is needed to release the lock. In the algorithms based on the *static logical-structure-based* approach [16], since sites cannot communicate directly with each other due to the nonfully connected network topology, longer time delay is needed.

10. CONCLUSIONS

In this paper, we have classified several well-known distributed mutual exclusion algorithms into two classes: token-based and non-token-based. Depending on how a request set is formed, non-token-based algorithms can be classified into two approaches: the *Ricart–Agrawala-type* and the *Maekawa-type*. Depending on whether or not a logical configuration is imposed on a site, token-based algorithms can be classified into two approaches: the *broadcast-based* and the *logical-structure-based*. To minimize both message traffic and time delay at the same time, there is one more approach to distributed mutual exclusion: a *hybrid* approach in which two algorithms are combined such that one minimizes message traffic and the other minimizes time delay. Over all, Chang *et al.*'s hybrid algorithm [4] has the lowest average message traffic among the non-token-based algorithms. Trehel and Naimi's improved $O(\log N)$ algorithm [26] based on the *dynamic logical-structure-based* approach has the lowest average message traffic among the token-based algorithms in a fully connected network. In general, non-token-based algorithms are more symmetric than token-based algorithms, while token-based algorithms requires fewer message traffic than non-token-based algorithms. Depending on different communication topologies, different system requirements (i.e., a trade-off between the number of messages exchanged and time delay) and different system environments (heavy traffic or light traffic), each algorithm has its merits and limitations and has been discussed and analyzed in this paper.

One other class of distributed mutual exclusion algorithms, on which we do not do a simulation study is quorum-based (or voting) [1, 8]. (Note that this approach is classified as *permission-based* in [17].) The basic idea in quorum-based algorithms is similar to Maekawa's algorithm. The main difference is the way to construct request sets, and how those quorum-based algorithms can dynamically reconstruct request sets when site failures occur. In a real-time distributed system and a system which uses priorities for scheduling, events for requesting the CS should be ordered on the basis of priorities of the processes (as first proposed in Goscinski's algorithm), rather than on the basis of the time when these events happened. However, algorithms mentioned in this paper grant the permission to enter the CS in a first-come-first-served manner. Goscinski has proposed a priority-based approach in mutual exclusion in real-time distributed systems [9, 10]. How to modify some of the algorithms mentioned in this paper for a real-time distributed system has been discussed in [6, 7]. Therefore, a simulation study of quorum-based algorithms and algorithms for real-time distributed systems is the future research direction.

ACKNOWLEDGMENTS

The author is grateful to anonymous referees for their careful reading and helpful comments.

REFERENCES

1. Agrawal, D., and El Abbadi, A. An efficient and fault-tolerant solution for distributed mutual exclusion algorithm. *ACM Trans. Computer Systems* **9,** 1 (Feb. 1991), 1–20.
2. Buckley, G., and Siberschatz, A. A failure tolerant centralized mutual exclusion algorithms. *Proc. 1984 International Conference on Distributed Computing Systems.* IEEE Comput. Soc., Austin, TX, 1984, pp. 347–356.
3. Carvalho, O. S. F., and Roucairol, G. On mutual exclusion in computer networks, technical correspondence. *Comm. ACM* **26,** 2 (Feb. 1983), 146–148.
4. Chang, Y. I., Singhal, M., and Liu, M. T. A hybrid approach to mutual exclusion for distributed systems. *Proc. 1990 Annual International Computer Software and Application Conference.* IEEE Comput. Soc., Chicago, IL, 1990, pp. 289–294.
5. Chang, Y. I., and Singhal, M. A correct $O(\sqrt{N})$ distributed mutual exclusion algorithm. *Proc. 1992 ISMM International Conference on Parallel and Distributed Computing and Systems.* ISMM, Pittsburgh, PA, 1992, pp. 56–61.
6. Chang, Y. I. Comments on two algorithms for mutual exclusion in real-time distributed computer systems. *J. Parallel Distrib. Comput.* **23,** 3 (Dec. 1994), 449–454.
7. Chang, Y. I. Design of mutual exclusion algorithms for real-time distributed systems. *J. Inform. Sci. Engrg.* **10,** 4 (Dec. 1994), 527–548.
8. Garcia-Molina, H., and Barbara, D. How to assign votes in distributed systems. *J. Assoc. Comput. Mach.* **32,** 4 (Oct. 1985), 841–860.
9. Goscinski, A. A synchronization algorithm for processes with dynamic priorities in computer networks with node failures. *Inform. Process. Lett.* **32,** 8 (Aug. 1989), 129–136.
10. Goscinski, A. Two algorithms for mutal exclusion in real-time distributed computer systems. *J. Parallel Distrib. Comput.* **9,** 12 (Dec. 1990), 77–82.
11. Helary, J. M., Mostefaoui, A., and Raynal, M. A general scheme for token- and tree-based distributed mutual exclusion algorithm. *IEEE Trans. Parallel Distrib. Systems* **5,** 2 (Nov. 1994), 1185–1196.
12. Lamport, L. Time, clocks and ordering of events in distributed systems. *Comm. ACM* **21,** 7 (July 1978), 558–565.

13. Maekawa, M. A. $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems. *ACM Trans. Comput. Systems* **3,** 2 (May 1985), 145–159.
14. Naimi, M., and Trebel, M. An improvement of the log (*N*) distributed algorithm for mutual exclusion. *Proc. 1987 International Conference on Distributed Computing Systems.* IEEE Comput. Soc., Columbus, Ohio, 1987, pp. 371–375.
15. Nishio, S., Li, K. F., and Manning, E. G. A resilient mutual exclusion algorithm for computer network. *IEEE Trans. Parallel Distrib. Systems* **1,** 3 (July 1990), 344–355.
16. Raymond, K. A tree-based algorithm for distributed mutual exclusion. *ACM Trans. Comput. Systems* **7,** 1 (Feb. 1989), 61–77.
17. Raynal, M. A simple taxonomy for distributed mutual exclusion algorithms. *ACM Oper. System Rev.* **23,** 2 (Dec. 1991), pp. 47–51.
18. Ricart, G., and Agrawala, A. K. Performance of a distributed network mutual exclusion algorithm. Technical report TR-774, Dept. of Computer Science. Univ. of Maryland, College Park, MD, Mar. 1979.
19. Ricart, G., and Agrawala, A. K. An optimal algorithm for mutual exclusion in computer networks. *Comm. ACM* **24,** 1 (Jan. 1981), 9–17.
20. Ricart, G., and Agrawala, A. K. Author's response to on mutual exclusion in computer networks, technical correspondence. *Comm. ACM* **26,** 2 (Feb. 1983), 146–148.
21. Sanders, B. A. The information structure of distributed mutual exclusion algorithms. *ACM Trans. Comput. Systems* **5,** 3 (Aug. 1987), 284–299.

Received June 10, 1992; revised May 4, 1995; accepted July 28, 1995

22. Singhal, M. A heuristically-aided algorithm for mutual exclusion in distributed systems. *IEEE Trans. Comput.* **38,** 5 (May 1989), 651–662.
23. Singhal, M. A dynamic information structure mutual exclusion algorithm for distributed systems. *IEEE Trans. Parallel Distrib. Systems* **3,** 1 (Jan. 1992), 121–125.
24. Singhal, M. A taxonomy of distributed mutual exclusion. *J. Parallel Distrib. Comput.* **18,** 12 (Dec. 1993), 94–101.
25. Suzuki, I., and Kasami, T. A distributed mutual exclusion algorithm. *ACM Trans. Comput. Systems* **3,** 4 (Nov. 1985), 344–349.
26. Trehel, M., and Naimi, M. A distributed algorithm for mutual exclusion based on data structures and fault tolerance. *Proc. 1987 Phoenix Conference on Computer and Communications.* IEEE Comput. Soc. Phoenix, AZ, 1987, pp. 35–39.

YE-IN CHANG was born in Taipei, Taiwan in 1964. She received the B.S. degree in computer science and information engineering from National Taiwan University, Taipei, Taiwan in 1986 and the M.S. and Ph.D. degrees in computer and information science from The Ohio State University, Columbus, Ohio, in 1987 and 1991, respectively. She joined the Department of Applied Mathematics, National Sun Yat-Sen University, Kaohsiung, Taiwan in 1991, where she is now an associate professor. Her research interests include database systems, distributed systems, knowledge-based systems, and computer networks.