A Token Based Distributed Mutual Exclusion Algorithm based on Quorum Agreements

Masaaki Mizuno* Mitchell L. Neilsen
Raghavendra Rao

Department of Computing and Information Sciences
Kansas State University
Manhattan, Kansas 66506

Abstract

Coteries may be used to construct permission based distributed mutual exclusion algorithms. One such algorithm is Maekawa's \( \sqrt{N} \) algorithm in which the coterie is constructed based on finite projective planes.

This paper presents a token based mutual exclusion algorithm which uses data structures similar to coteries, called quorum agreements. The performance of the algorithm depends upon the quorum agreements used. When a good quorum agreement is used, the overall performance of the algorithm compares favorably with the performance of other mutual exclusion algorithms.

1 Introduction

Many distributed mutual exclusion algorithms exist [5, 6, 8, 10, 11, 12, 13, 14, 15]. These algorithms can be classified into two groups [9, 13]. The algorithms in the first group are token based [8, 11, 13, 14, 15]. The possession of a system-wide unique token gives a node the right to access a resource in mutual exclusion. The algorithms in the second group are permission based [5, 6, 10, 12]. If a node receives permission from a certain set of nodes, it may enter the critical section.

Among the permission based algorithms, Maekawa's algorithm is a very general form of a coterie based mutual exclusion algorithm, which incorporates a deadlock avoidance mechanism. The notion of coteries was first introduced by Garcia-Molina and Barbara [4].

Given a set of nodes \( U \) in the system, where \( N = |U| \), a coterie under \( U \) is a set of subsets of \( U \) in which any two members have a nonempty intersection. In Maekawa's algorithm, a coterie based on finite projective planes is used.

In this paper, we present a token based algorithm which uses data structures similar to coteries, called quorum agreements. Quorum agreements were introduced by Barbara and Garcia-Molina [2]. A quorum agreement is a pair of sets of subsets of \( U \), \( (Q, Q^{-1}) \), where every member of \( Q \) intersects with every member of \( Q^{-1} \). The algorithm based on quorum agreements adopts the mechanism to circulate the token used in Suzuki and Kasami's algorithm [14]. In fact, Suzuki and Kasami's algorithm is a special case of our algorithm.

Recently, Nishio, et al., presented a token regeneration algorithm and an algorithm to recover the data structure at a recovering node, for Suzuki and Kasami's algorithm [7]. Since our algorithm uses essentially the same mechanism, their recovery algorithms can be used in our algorithm. Even though their token regeneration algorithm may be directly applied, it may cause unnecessary blocking of a requesting node during the existence of a failure, due to the differences between Suzuki and Kasami's algorithm and our algorithm. However, the blocking may be prevented with a minimal amount of modification.

The performance of our algorithm depends upon the underlying quorum agreement. When a good quorum agreement is used, the overall performance of our algorithm compares favorably with the performance of other mutual exclusion algorithms.

As in many other distributed mutual exclusion algorithms, we assume the following: (1) the network is
fully connected; and (2) at any time, each node initiates at most one outstanding request for mutual exclusion.

Section 2 briefly reviews quorum agreements. The mutual exclusion algorithm is described in Section 3. Section 4 proves the correctness of the algorithm with respect to guaranteed mutual exclusion, starvation freedom, and deadlock freedom. Section 5 analyzes the performance of the algorithm. Section 6 describes the modification required to prevent unnecessary blocking of a requesting node when the recovery algorithm by Nishio, et al., is applied to the quorum agreement based algorithm.

2 Quorum Agreements

2.1 Coteries

Let $U = \{1, 2, 3, \ldots, N\}$ denote the set of $N$ nodes in the system. A collection of sets, $C$, is a coterie under $U$ [4] iff

1. $G \in Q$ implies that $G \neq \emptyset$ and $G \subseteq U$.
2. (Intersection property): $G, H \in C$ implies that $G \cap H \neq \emptyset$.
3. (Minimality): There are no $G, H \in C$ such that $G \subset H$.

The sets $G \in C$ are called quorums. For example, $\{\{a, b\}, \{b, c\}, \{c, a\}\}$ is a coterie under $\{a, b, c\}$.

2.2 Quorum Agreements

A collection of sets, $Q$, is a quorum set under $U$ [2] iff

1. $G \in Q$ implies that $G \neq \emptyset$ and $G \subseteq U$.
2. (Minimality): There are no $G, H \in Q$ such that $G \subset H$.

Let $I_Q = \{H \subseteq U \mid G \cap H \neq \emptyset \ \forall G \in Q\}$. The antiquorum set of $Q$, denoted by $Q^{-1}$, is [2]

$$Q^{-1} = \{H \in I_Q \mid H' \notin H \ \forall H' \in I_Q\}$$

Then, the pair, $QA = (Q, Q^{-1})$, is called a quorum agreement under $U$ [2].

Note that $Q$ and $Q^{-1}$ are not required to be coteries. However, a quorum agreement may be constructed from a coterie, $C$, by assigning $Q = Q^{-1} = C$. One such coterie may be constructed by using finite projective planes [6]. In such coteries, the size of each quorum is approximately $\sqrt{N}$. For example, a coterie based on a finite projective plane for $N = 7$ is shown in Table 1.

<table>
<thead>
<tr>
<th>Table 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>q1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

Weighted voting may be used to generate quorum agreements [2]. Cheung, et al., discussed enumeration and generation of efficient quorum agreements using weighted voting [3].

The grid protocol proposed by Maekawa [6] and by Agrawal and El Abbadi [1] may be modified to obtain quorum agreements. Each node is assigned a location on a rectangular $a \times r$ $(= N)$ grid. Quorums in $Q$ and $Q^{-1}$ are formed by choosing all elements in one row and one element from each row, respectively. For example, consider the following simple grid with 6 nodes:

<table>
<thead>
<tr>
<th>Table 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

Then, the following sets $Q$ and $Q^{-1}$ are formed:

$$Q = \{\{1, 2, 3\}, \{4, 5, 6\}\}$$

$$Q^{-1} = \{\{1, 4\}, \{1, 5\}, \{1, 6\}, \{2, 4\}, \{2, 5\},$$

$$\{2, 6\}, \{3, 4\}, \{3, 5\}, \{3, 6\}\}$$

3 The Algorithm

3.1 Overview

Let $QA = (Q, Q^{-1})$ be the quorum agreement used by the algorithm. Even though it is not necessary, we assume, for simplicity, that exactly one quorum from both $Q$ and $Q^{-1}$ are assigned to each node. We call the quorums from $Q$ and $Q^{-1}$ assigned to node $i$ the request set (denoted by $R_i$) and acquired set (denoted by $A_i$) of node $i$, respectively. That is, $R_i \in Q$ and $A_i \in Q^{-1}$.

Any node that wants to enter the critical section must first acquire the token. A node that is currently holding or has acquired the token may enter and stay in the critical section until it passes the token to some other node. The token is represented by a PRIVILEGE message, i.e., when a node receives a PRIVILEGE message, it may enter the critical section.

The basic mechanism to circulate the token is adopted from Suzuki and Kasami’s algorithm [14]. Each node $i$ maintains an array of sequence numbers of size $N$, denoted by $RN_i$. To obtain the token, node $i$ sends REQUEST messages to all members of its request set $R_i$, excluding itself. The REQUEST message
carries a copy of RN. RN[i] counts the number of requests for the token which node i has issued thus far; that is, when node i issues a new request for the token, it increments RN[i] by one. RN[j] (i ≠ j) holds the largest sequence number of the request from node j that node i has received.

If node i is holding the token when it wants to enter the critical section, it can immediately enter without sending REQUEST messages. In this case, the node does not increment its sequence number RN[i].

The PRIVILEGE message carries an array LN of size N and a queue of pending requests TOKEN.Q. LN[j] holds the largest sequence number of the request from node j that has already been serviced or is in TOKEN.Q waiting to be serviced. If node i receives a PRIVILEGE message and finds that the relation LN[j] < RN[i] holds for some j, then the most recent request from node j has neither been serviced nor been placed in TOKEN.Q. Thus, node i places the request by such a node j into TOKEN.Q and updates LN[j] to RN[j].

Another message to be exchanged among nodes in our algorithm is an ACQUIRED message. When node j receives a PRIVILEGE message, it sends ACQUIRED messages to all nodes in its acquired set Aj, excluding itself. An ACQUIRED message contains a copy of array LN, denoted by LNj for node j's view of LN. When node k receives an ACQUIRED message from node j (i.e., k ∈ Aj), it sends a REQUEST message back to node j if RNk contains more recent requests than LNj.

Unlike Suzuki and Kasami’s algorithm, the initial REQUEST messages issued by node i reach only the nodes in Ri. Thus, this mechanism of exchanging ACQUIRED and subsequent REQUEST messages is essential in our algorithm to propagate the request to nodes outside of Ri.

Note that if node k does not have new requests relative to LNj in RNk, it waits to send a REQUEST message back to node j until it receives more recent requests or node k itself issues a new request. Thus, each node maintains a set of node identifiers, denoted by PENDING, to which it must send pending REQUEST messages back when it receives more recent requests. In order to be able to identify new requests, node k updates RNk every time when it receives a REQUEST(RNj) message, a PRIVILEGE(TOKEN.Q, LN) message, or an ACQUIRED(j, LNj).

3.2 Details of the algorithm

Besides RNj and PENDING, each node i maintains a boolean variable HOLDING, which is true if and only if the node holds the token but is neither in nor requesting to enter the critical section.

The algorithm assumes the existence of the following four sub-function/procedures:

1. function head(TOKEN.Q) which returns the node identifier of the request in the head of TOKEN.Q,
2. procedure dequeue(TOKEN.Q) which dequeues the request in the head of TOKEN.Q,
3. procedure max(RN1, RN2) which returns an array containing the componentwise maximum values of the two arrays RN1 and RN2, and
4. procedure update(TOKEN.Q, LN, RNi), described below.

Procedure update(TOKEN.Q, LN, RNi)
Var j : node identifier;
begin
for j := 1 to N do
if LN[j] < RN[j] then LN[j] := RN[j], and enqueue the request by node j into TOKEN.Q;
else RN[j] := LN[j];
end

The algorithm for node i is shown in Figure 1. The algorithm consists of three major procedures. Each procedure is executed indivisibly. While a node is waiting for the token or executing in the critical section, it can receive REQUEST and ACQUIRED messages and execute the corresponding procedures.

Note that in Procedure P1, node i may not have received REQUEST messages back from all of the nodes in Ai when it leaves the critical section. However, the node does not wait for the arrival of these REQUEST messages to send the token. These REQUEST messages will eventually reach node i. It is shown in Section 4 that the requests carried by these late REQUEST messages will eventually be enqueued into TOKEN.Q by other nodes (when they acquire the token) or by node i itself when it again requests for mutual exclusion and receives the token later.

At start up time, the system is initialized as follows:

At each node \( i \), \( RN_i \) is initialized to be all zero, and \( \text{PENDING} \) is initialized to be empty. \( \text{TOKEN\_Q} \) is empty and \( LN \) is initialized to be all zero. The node initially holding the token sets \( \text{HOLDING} \) to true and sends \( \text{ACQUIRED} \) messages to all nodes in its acquired set, excluding itself.

**Const**
- \( R_i \): request set;
- \( A_i \): acquired set;
- \( N \): number of nodes;
- \( i \): node identifier;

**Var**
- \( RN_i \): array \([1..N]\) of integer;
- \( \text{HOLDING} \): boolean;
- \( \text{PENDING} \): set of node identifiers;

**Procedure P1** (*Enter Critical Section*)
\[
\text{begin}
\text{if (not \text{HOLDING}) then}
\text{\hspace{1em}begin}
\text{\hspace{2em}RN_i[i] := RN_i[i] + 1;}
\text{\hspace{2em}for each } j \in (R_i \cup \text{PENDING}) (i \neq j)
\text{\hspace{3em}send a \text{REQUEST}(RN_i) message to node } j;
\text{\hspace{2em}PENDING := \emptyset;}
\text{\hspace{2em}wait until a \text{PRIVILEGE}(\text{TOKEN\_Q}, LN) message is received;}
\text{\hspace{2em}dequeue(\text{TOKEN\_Q});}
\text{\hspace{2em}update(\text{TOKEN\_Q}, LN, RN_i);}
\text{\hspace{3em}for each } j \in A_i (i \neq j)
\text{\hspace{4em}send an \text{ACQUIRED}(i, LN) message to node } j;
\text{\hspace{2em}end}
\text{\hspace{1em}end}
\text{HOLDING := false;}
\text{\hspace{1em}CRITICAL SECTION}
\text{update(\text{TOKEN\_Q}, LN, RN_i);}
\text{if } \text{TOKEN\_Q} \neq \emptyset
\text{\hspace{1em}then send a \text{PRIVILEGE}(\text{TOKEN\_Q}, LN) message to head(\text{TOKEN\_Q});}
\text{else HOLDING := true;}
\text{end}
\]

**Procedure P2** (*Received ACQUIRED\((j, LN_j)\) message*)
\[
\text{begin}
\text{\hspace{1em}RN_i := max(RN_i, LN_j);}
\text{\hspace{1em}if RN_i \neq LN_j}
\text{\hspace{2em}(* RN_i contains more recent requests *)}
\text{\hspace{3em}then send \text{REQUEST}(RN_i) message to node } j
\text{\hspace{2em}else PENDING := PENDING \cup \{j\};}
\text{end}
\]

**Procedure P3** (*Received REQUEST\((RN_j)\) message*)
\[
\text{Var old\_RN_i : array \([1..N]\) of integer;}
\text{begin}
\text{\hspace{1em}old\_RN_i := RN_i;}
\text{\hspace{1em}RN_i := max(RN_i, RN_j);}
\text{\hspace{1em}if RN_i \neq old\_RN_i then}
\text{\hspace{2em}(* RN_i contains more recent requests *)}
\text{\hspace{3em}begin}
\text{\hspace{4em}if PENDING \neq \emptyset then}
\text{\hspace{5em}begin}
\text{\hspace{6em}for each } k \in \text{PENDING}
\text{\hspace{7em}send a \text{REQUEST}(RN_i) message to node } k;
\text{\hspace{6em}PENDING := \emptyset}
\text{\hspace{5em}end}
\text{\hspace{4em}end}
\text{\hspace{2em}if \text{HOLDING} then}
\text{\hspace{3em}begin}
\text{\hspace{4em}update(\text{TOKEN\_Q}, LN, RN_i);}
\text{\hspace{4em}send \text{PRIVILEGE}(\text{TOKEN\_Q}, LN) to head(\text{TOKEN\_Q});}
\text{\hspace{3em}end}
\text{end}
\]

**Figure 1. Algorithm**

### 4 Correctness

We justify the correctness of the algorithm with respect to (1) guaranteed mutual exclusion, (2) starvation freedom, and (3) deadlock freedom.

The algorithm guarantees mutual exclusion because there is only one token in the system and only the token holder can enter the critical section.

**Theorem:** The algorithm is starvation free and deadlock free.

**Proof:** First, we show that if a request is issued by an arbitrary node, say node \( i \), then the request will reach a node which holds the token or will hold the token in finite time. If the request reaches a node which holds the token, the request is enqueued in \( \text{TOKEN\_Q} \). Once the request is enqueued in \( \text{TOKEN\_Q} \), node \( i \) only has to wait until every other node is allowed to enter the critical section at most once. Thus, starvation freedom and deadlock freedom are guaranteed.

Assume that the request from node \( i \) never reaches a node holding the token or a node which will hold the token within finite time. We proceed by induction on the number of nodes that know about the request from node \( i \).
**Base Step:** Initially, node $i$ sends requests to all nodes in $R_i$. The requests arrive in finite time. Thus, at least $|R_i|$ nodes know about the request from node $i$ in finite time. If any node in $R_i$ holds the token or receives the token later, this is a contradiction. Thus, we may assume that none of the nodes in $R_i$ hold the token nor will they receive the token in finite time.

**Hypothesis:** Suppose that $M$ nodes know about the request from node $i$, for some $|R_i| \leq M < N$, and that none of these $M$ nodes hold the token nor will they receive the token in finite time.

**Inductive Step:** Then, there exists a node, say node $j$, that does not know about the request from node $i$ and is holding the token. Since $R_i \cap A_j \neq \emptyset$, there is some node $k \in R_i \cap A_j$. When the ACQUIRED message from node $j$ arrives at node $k$, the request from node $i$ will be forwarded to node $j$ in finite time. Then, $M + 1$ nodes will know about the request from node $i$. By our assumption, none of these $M + 1$ nodes can hold the token nor will they be able to receive the token. Thus, node $j$ must forward the token to some other node, before node $k$ transfers the request from node $i$ to node $j$.

By induction, all $N$ nodes will know about the request from node $i$ and none of these $N$ nodes can hold the token nor will they be able to receive the token in finite time. This is a contradiction. Thus, the request from node $i$ must reach a node which is holding or will hold the token.

Therefore, the algorithm is starvation free and deadlock free. \(\square\)

As an example, consider the quorum agreement QA = $(Q, Q^{-1})$ such that $Q = Q^{-1} = C$, where $C$ is the coterie $\{q_1, \ldots, q_7\}$ given in Table 1. Assume that $R_i = A_i = q_i$. Suppose that node 1 issues a request for mutual exclusion, and REQUEST messages from node 1 arrive at nodes 2 and 3. If either node 2 or 3 holds the token, the request is enqueued in TOKEN_Q. Suppose that node 7 holds the token. It sends ACQUIRED messages to nodes 3 and 4. Since $R_i \cap A_7 = \{3\}$, node 3 transfers the request from node 1 to node 7 in a REQUEST message.

Assume that node 7 passes the token to node 5 before receiving the REQUEST message from node 3. Note that nodes 1, 2, 3, and 7 know about the request from node 1 after node 7 receives the REQUEST message from node 3.

Now, node 5 has the token, but it passes the token to node 6 before receiving a REQUEST message from node 2 ($R_i \cap A_5 = \{2\}$) which contains the request from node 1. Then, node 1 tries to pass its own request to node 6 ($R_i \cap A_6 = \{1\}$). This scenario continues at most 4 times until some token holder has received the request from node 1 before it passes the token.

### 5 Performance Analysis

We evaluate the performance of distributed mutual exclusion algorithms using the following criteria:

1. The number of messages required for a node to enter the critical section should be minimized.
2. Heavy-load synchronization delay should be minimized. We define synchronization delay to be the number of rounds of message passing required for one node to leave the critical section and another node to enter the critical section. We count a set of message exchanges which could take place concurrently as one round. **Heavy-load** means that there are always nodes waiting to enter the critical section.
3. Light-load synchronization delay should be minimized. **Light-load** means that there is no other node requesting when a node leaves the critical section.

#### 5.1 Number of messages

If a node holds the token when it requests for mutual exclusion, no messages need to be exchanged. Thus, the lower bound is zero messages.

Otherwise, REQUEST, ACQUIRED, and PRIVILEGE messages are exchanged among nodes for each critical section entry. We assume that $i \in R_i$, and $i \in A_i$. Node $i$ needs to send $(|R_i| - 1)$ REQUEST messages. One message is required for the token to be passed to node $i$. When node $i$ receives the token, it sends $(|A_i| - 1)$ ACQUIRED messages and receives REQUEST messages from up to $(|A_i| - 1)$ nodes. Thus, at most $(|R_i| - 1) + 2(2 + (|A_i| - 1) + 1)$ messages are exchanged per critical section entry for node $i$. If node $k \in A_i$ has a pending REQUEST message to node $i$, when it receives a new ACQUIRED message from node $i$, only one REQUEST message is returned to node $i$. Thus, if node $i$ does not hold the token when it requests for mutual exclusion, between $(|R_i| - 1) + (|A_i| - 1) + 1$ and $(|R_i| - 1) + 2 + (|A_i| - 1) + 1$ messages are exchanged per critical section entry.
For example, if the quorum agreement is formed based on the binary tree protocol, introduced by Agrawal and El Abbadi [1], the minimum sizes of \( R_i \) and \( A_i \) are approximately \( \log N \). Thus, the algorithm could attain \( O(\log N) \) performance. The quorums generated by the binary tree protocol, however, are not symmetric, i.e., some nodes appear in more quorums than others.

Quorums formed using finite projective planes are symmetric, i.e., each node appears in the same number of quorums. The number of required messages using quorum agreement based on finite projective planes is between 0 and approximately \( 3 \cdot \sqrt{N} \).

A modified grid-set protocol (Table 2) may be used to generate symmetric quorums. Let the size of the grid be \( a \cdot r = N \). Then, the maximum number of messages per critical section entry is \( ((r-1) + 2 \cdot (a-1) + 1) \). Let \( f(r) = ((r-1) + 2 \cdot (a-1) + 1) \), where \( a = N/r \). If we view \( f \) as a differentiable function with respect to \( r \), the minimal value of \( f(r) \) occurs when \( r = \sqrt{2 \cdot \sqrt{N} - 2} \). This is slightly better than quorum agreements based on finite projective planes.

Note that if the quorum agreement is formed such that \( R_i = \{1, 2, \ldots, N\} \) and \( A_i = \{i\} \) for \( 1 \leq i \leq N \), the algorithm becomes essentially the same as Suzuki and Kasami's algorithm.

For comparison, the following list shows the upper bound of the performance attained by other mutual exclusion algorithms:

- Maekawa's algorithm: between \( 3 \cdot \sqrt{N} \) and \( 5 \cdot \sqrt{N} \)
- Suzuki and Kasami's algorithm: \( N \)
- Raymond's algorithm: \( 2 \cdot D \), where \( D \) is the diameter of the network
- Trehel and Naimi's algorithm: \( O(\log N) \), on the average

5.2 Heavy-load synchronization delay

In our algorithm, since only one PRIVILEGE message needs to be passed, the heavy-load synchronization delay is one.

The following list shows the synchronization delay of other algorithms:

- Maekawa's algorithm: 2
- Suzuki and Kasami's algorithm: 1
- Raymond's algorithm: \( D \), where \( D \) is the diameter of the network
- Trehel and Naimi's algorithm: 1

5.3 Light-load synchronization delay

In our algorithm, the worst case occurs when the token holder is not in the request set of the requesting node \( i \). In order for node \( i \) to acquire the token, one round of REQUEST messages, another round of REQUEST messages, and one PRIVILEGE message must be exchanged. Thus, the synchronization delay is three.

The following list shows the synchronization delay of other algorithms:

- Maekawa's algorithm: 2
- Suzuki and Kasami's algorithm: 2
- Raymond's algorithm: \( 2 \cdot D \), where \( D \) is the diameter of the network
- Trehel and Naimi's algorithm: \( O(\log N) \), on the average

5.4 Overall performance comparison

Raymond's algorithm and Trehel and Naimi's algorithm reduce the number of required messages but have higher (light-load) synchronization delay. On the other hand, Maekawa's algorithm and Suzuki and Kasami's algorithm perform well in terms of synchronization delay, but require more messages. The quorum agreement based algorithm falls in between those two groups.

The quorum agreement based algorithm requires, on the average, slightly more messages than Raymond's algorithm and Trehel and Naimi's algorithm, but fewer messages than Maekawa's algorithm and Suzuki and Kasami's algorithm, if quorum agreements are formed based on finite projective planes or the modified grid-set protocol. In terms of synchronization delay, the quorum agreement based algorithm generally shows better performance than Raymond's algorithm and Trehel and Naimi's algorithm, but slightly worse performance than Maekawa's algorithm and Suzuki and Kasami's algorithm.

6 Failure Recovery

The token regeneration algorithm proposed by Nishio, et al., allows the token to be regenerated only when all of the nodes and all of the communication lines are functional; otherwise, the entire system is blocked. Because of this blocking feature, however, the algorithm guarantees mutual exclusion under network partitioning.
In this section, we discuss the modification to their token regeneration algorithm required to prevent unnecessary blocking of a requesting node when the algorithm is applied to the quorum agreement based algorithm. In Suzuki and Kasami's algorithm, REQUEST messages are directly sent to all nodes. On the other hand, in the quorum agreement based algorithm, there is an intermediate node between the requesting node and a node outside the request set of the requesting node. This difference may cause unnecessary blocking of a requesting node during the existence of a failure.

Suppose that node $i$ requests for mutual exclusion. Let nodes $j$ and $k$ be such that node $j$ holds the token and $j \notin R_i$ and $R_i \cap A_j = \{k\}$. In the quorum agreement based algorithm, unnecessary blocking of node $i$ may occur when node $j$ cannot receive a request from node $i$ due to the failure of node $k$ (or the failure of a communication line between node $i$ and node $k$ or a communication line between node $k$ and node $j$). For example, suppose that node $j$ holds the token with an empty TOKEN.Q. Node $k$ receives a REQUEST message from node $i$ and an ACQUIRED message from node $j$. Then, suppose that node $k$ fails before it sends a REQUEST message back to node $j$. If node $i$ and node $j$ do not know of the failure, the token will be held by node $j$ and will not be forwarded to node $i$.

If the token regeneration algorithm by Nishio, et al., is used, node $i$ will eventually timeout and send TOKEN.MISSING messages to all other nodes. When node $j$ receives the TOKEN.MISSING message, it replies with a NACK message to node $i$. Should node $k$ still be down, node $i$ will not receive a reply message from node $k$ in time, and the same procedure will be repeated. This recovery procedure is repeated until all of the nodes, including node $k$, recover, or other nodes issue new requests and the token is passed to these nodes and is eventually forwarded to node $i$.

This unnecessary blocking of node $i$ may be prevented if node $i$ includes its current sequence numbers RN$_i$ in the TOKEN.MISSING messages. When a node receives the TOKEN.MISSING message, it follows Procedure P3 described in Section 3, behaving as if it received a REQUEST message, as well as sending an ACK or NACK message back to node $i$. With this modification, when node $j$ receives the TOKEN.MISSING message from node $i$, it notices the new request from node $i$ and forwards the token to node $i$ immediately.

7 Conclusion

This paper presented a token based distributed algorithm for mutual exclusion based on quorum agreements. The algorithm requires slightly more messages than the most efficient existing mutual exclusion algorithms [8, 15], but performs better in terms of synchronization delay than those algorithms. With the existence of a complete failure recovery algorithm [7], the quorum agreement based algorithm compares favorably with other mutual exclusion algorithms.

References

INRIA, 78153 Le Chesnay Cedex, France, 1990, will appear in Operating Systems Review.


