A Study on the Distributed k-Mutual Exclusion Problem

by Hirotsugu Kakugawa

February 25, 1992

Dissertation submitted to the Faculty of the Information Engineering Course of the Graduate School of Engineering of Hiroshima University in partial fulfillment of the requirements for the degree of Master of Engineering.

Abstract

The distributed k-mutual exclusion problem is the problem of guaranteeing that at most k processes can enter a critical section at a time in a distributed system. The distributed 1-mutual exclusion problem is one of fundamental distributed problems in distributed systems and many algorithms for solving the problem have been proposed. In this paper, we introduce the k-coterie as an extension of the coterie used to solve the distributed 1-mutual exclusion problem, and propose a distributed k-mutual exclusion algorithm based on the k-coterie. The algorithm we propose and an algorithm proposed by Raymond [1] are compared by experiments using workstations connected by Ethernet. Since the fault tolerance is a very important issue in distributed systems, we introduce a measure called (k, r)-availability, which is a probability that at least r processes can enter a critical section at a time. And then, we investigate a necessary and a sufficient conditions that a class of k-coteries, k-majority coterie, is optimal in terms of the (k, r)-availability.

Contents

List of Figures

List of Tables

1 Introduction

In this paper, we consider *the distributed k-mutual exclusion problem.* The distributed k-mutual exclusion problem is the problem of guaranteeing that at most k processes can enter a critical section at a time in a distributed system.

For instance, consider a distributed database. The distributed database system consists of some computers on which copies of data are held. The reason why more than one computers have copies of data is to increase the availability of the database. Even if some computers fail and stop, other alive computers may have copies of data and users can read and update the database. Therefore, in general, distributed databases can be considered more fault tolerant than usual databases developed on a computer. This is one of the motivations to introduce distributed systems.

In the distributed system, to update a record of the database, a user must lock the record to guarantee that other users are not updating the same record. Thus, in this sense, we call a fragment of program which updates the data *critical section* and we say that *a process is in a critical section* if it is executing a code in a critical section. And we are required to solve the mutual exclusion problem. In this example, the number of processes in the distributed system which is in a critical section is at most one at a time.

The distributed 1-mutual exclusion problem is one of fundamental distributed problems and many algorithms that solve the problem have been proposed. Lamport proposed a distributed mutual exclusion algorithm in [2]. In his algorithm, a process which enters a critical section must get permissions from all the processes. So, this algorithm is based on 'unanimous' consensus scheme. But because if a process stops by failure then other alive processes can't enter a critical section, it is not considered as a good algorithm from the view point of fault tolerance. Thomas proposed 'majority' consensus algorithm to guarantee mutual exclusion [3]. A process which enters a critical section must get permissions from a majority of all processes. These two schemes are rather simple; other schemes to solve the problem proposed are:

1. *The weighted voting scheme*

   Each process has *a weighted vote*. To enter a critical section, a process ought to collect votes from processes so that the sum of the weights of votes it gets exceeds a pre-specified threshold.

2. *The token scheme*

   There exists a (virtual) *token* in a system. A process which receives the token, sends it to the next process so that it rounds among the processes. The token is regarded as the privilege, i.e., a process which has the token has the privilege to enter a critical section. Since there exists only one token, 1-mutual exclusion is guaranteed.

3. *The coterie scheme*

   A *coterie* is a set of groups of processes such that any two groups have non-empty intersection. Such groups of processes are called quorums. A process has to get permissions from all processes in a quorum of the coterie to enter a critical section and each process does not give permission to more than one process at a time. Since any two quorums have non-empty intersection, more than one process can't enter a critical section at any given time.

4. *The primary site scheme*

   This is a centralized scheme: a process which acts as the *1-mutual exclusion server* of a system is a priori selected and a process which enters a critical section sends a mutual exclusion request to the server. The server decides locally the order of entering the critical section.

Note that the coterie scheme is a generalization of the weighted voting one. To see this, let $U = \{x_1, x_2, .., x_n\}$ be set of processes, $v(x_i)$ be the weighted votes of x_i, and θ be a threshold. Consider a set of subsets of processes $\mathcal{C} = \{Q_1, Q_2, ..., Q_m\}$ ($Q_i \subseteq U$) such that $\sum_{x_j \in Q_i} v(x_j) \geq \theta$ iff $Q_i \in \mathcal{C}$. Then $\mathcal{C}$ is a coterie.

The *Availability* of a coterie is the probability that at least one process can enter a critical section. Barbara and Garcia-Molina [4] proved that the availability of the majority coterie is the largest among all coteries if $p > 0.5$ under assumptions that (1) the network topology is a complete graph, (2) each communication link is error-free, and (3) the aliveness of each process is the same probability p. Since the primary site scheme is a special case of the coterie scheme[1], truly distributed systems are more fault tolerant than single machine systems, provided that $p > 0.5$.

We consider a generalization of the mutual exclusion problem in such a way that k processes can enter a critical section at a time. We call this problem *the distributed k-mutual exclusion problem.* Thus, the mutual exclusion problem is that mutual exclusion problem. We consider the following schemes to solve the distributed k-mutual exclusion problem.

1. *The k 1-mutual exclusions scheme*

   This scheme uses k 1-mutual exclusions for k-mutual exclusion. A process which enters a critical section must select one of entrances.

2. *The k tokens scheme*

   Make k privilege tokens in a system. A process which have a token can enter a critical section. Since there exist k tokens, at most k processes can enter a critical section at a time.

3. *The primary site scheme*

   A process is selected as the k-mutual exclusion server of a system. Each process sends a request message to the server to enter a critical section. The server process permits at most k processes to enter a critical section at a time.

4. *The k primary sites scheme*

   k processes are selected as mutual exclusion servers and each server acts

[1] Let $\mathcal{C} = \{\{x\}\}$ where x is a process. Then, $\mathcal{C}$ is a coterie. We call such a coterie $\mathcal{C}$ *a singleton coterie*. The primary site scheme can be considered as the coterie scheme when a singleton coterie is used.

as a 1-mutual exclusion server. To enter a critical section, a process ought to select a server among the k servers.

5. *The k-coterie scheme*

   We propose a new concept k-coterie for solving distributed k-mutual exclusion problem. The k-coterie is an extension of the coterie such that there exist non-intersecting k quorums but there does not exist non-intersecting $k+1$ quorums. Since there exists k non-intersecting quorums, at most k processes can enter a critical section at a time. Its formal definition will be given in the next section.

The first scheme can make use of the techniques we have developed for the coterie scheme. This scheme is simple but may cause a long delay to enter a critical section. That is, there is a case in which a process wishing to enter a critical section selects a busy entrance and waits its turn for a long time. Althogh, the process may be able to enter a critical section immediately by choosing another entrance. Therefore, the first scheme is inadequate.

Consider the second scheme. One of the possible solutions is as follows: make k tokens in a system and let them round among the processors forming a logical ring. A process which wants to enter a critical section waits for an arrival of a token. Then it enters a critical section. When it exits from the critical section, it sends the token to the next process in the ring. In this solution, entrances of a critical section are identical. But even if no processes are requesting to enter a critical section, tokens must be continued rounding, i.e., processes must continue receiving and sending tokens. This is undesirable.

The third scheme is trivial. This scheme is a special case of 1-coterie. Let p be the aliveness of processes, then it is easy to see that the availability of a coterie which represents this scheme is p.

The fourth scheme is a special case of the fifth scheme, the k-coterie scheme. Therefore, in this paper, we consider the k-coterie scheme to solve the distributed k-mutual exclusion problem.

In section 2, terms and concepts which are used in this paper are defined.

In section 3, we propose a k-coterie based distributed k-mutual exclusion algorithm, and the correctness of the algorithm is shown. Raymond proposed a distributed k-mutual exclusion algorithm in [1] (it does not based on the k-coterie). The aim of section 4 is to show the advantages of our algorithm by an experiment comparing with the algorithm by Raymond. The experiment is done by using workstations connected by a local area network. In section 5, a goodness measure of the k-coterie, (k, r)-availability, which is an extension of the availability is introduced. And then, we investigate a necessary and a sufficient conditions for a class of k-coteries, the k-majority coterie, to be optimal in the sense of the (k, r)-availability. In section 6, we conclude this thesis by summarizing the results obtained and propose future tasks remained open.

2 Preliminary

In this section, terms and concepts which will be used in this paper are prepared.

2.1 A Model of Distributed Systems

Let us begin with the definition of a distributed system which is assumed in this paper. *A distributed system* consists of n *processes* and *communication links* between processes. (A process is an abstraction of a computer.)

Each process executes the same program, but has unique process identifier (process ID). Without loss of generality, we assume that a process ID is a positive integer. Processing speed of processes may be different. Some processes may execute a program fast and others may do really slow; processing speed of processes may change during the execution of a program. But it is guaranteed that a process can execute its next instruction within a finite time unless it has been terminated.

Each process has its own *local clock*. Each local clock may indicate different time, and no processes can tell the global time[2]. Therefore, to synchronize other processes, processes can't make use of their local clocks.

[2]The definition of the distributed k-mutual exclusion problem requires the existence of the global time.

When a distributed system is initiated, a process knows about its process ID, the number of links, and the adjacent processes. Since there is no centralized control to solve a problem, processes must collect enough information from other processes through communication links.

We assume that the network topology is a complete graph and links are error-free. The only mechanism provided in the system for information exchange between processes is the message passing, i.e., processes do not have shared memory. The message passing is done by the point-to-point communication style. Communication links are bidirectional. If there exists a link between processes x and y, then x can send messages to y and vice versa. Each process has a message queue of infinite length, which stores messages arrived to it. Operations provided for the message passing are as follows.

- The **SEND** operation

  SEND is used to send a message. To send a message, a destination process must be specified. Since the point-to-point communication is assumed, there must be a link between the source process which issues the SEND and the destination process. Messages sent by a process are eventually put into the mseesage queue of the destination process in a finite time.

- The **RECEIVE** operation

  As described, each process maintains a message queue. By issuing RECEIVE, the message which is the first item in the queue is retrieved. If the queue is empty, a special value which indicates that the queue is empty is returned.

The order of messages is kept unchanged during the deliverly. That is, when process x sends messages M_1 and M_2 in this order to y, y receives M_1 and M_2 in the same order. It is guaranteed that each message is delivered in a finite time. But the message delivery delay is unpredictable; the delay may vary during the execution of a program.

2.2 k-coterie

We show the formal definition of the k-coterie. Let $U = \{x_1, ..., x_n\}$ be the set of processes, where n is the number of processes.

Definition 1 (k-coterie) *A non-empty set $\mathcal{C}$ of non-empty subsets Q of U is called a k-coterie if and only if all of the following three conditions hold:*

1. **Non-intersection Property:**
   For any l $(1 \leq l \leq k-1)$ and for any elements $Q_1, ..., Q_l \in \mathcal{C}$, there exists an element $Q \in \mathcal{C}$ such that $Q \cap Q_i = \emptyset$ for $1 \leq i \leq l$.

2. **Intersection Property:**
   There are no $k+1$ elements $Q_1, ..., Q_{k+1} \in \mathcal{C}$ such that $Q_i \cap Q_j = \emptyset$ for all $1 \leq i, j \leq k+1$.

3. **Minimality Property:**
   For any two distinct elements Q_i and Q_j in $\mathcal{C}$, $Q_i \not\subseteq Q_j$.

□

Examples of k-coterie are shown below.

- $k = 1$

$$\mathcal{C}_1 = \{\{1\}\}$$

$$\mathcal{C}_2 = \{\{1, 2\}, \{2, 3\}, \{3, 1\}\}$$

- $k = 2$

$$\mathcal{C}_3 = \{\{1\}, \{2\}\}$$

$$\mathcal{C}_4 = \{\{1, 2\}, \{3, 4\}, \{3, 4\}, \{4, 1\}\}$$

- $k = 3$

$$\begin{aligned} \mathcal{C}_5 = \; & \{\{1, 4\}, \{2, 5\}, \{3, 6\}, \\ & \{1, 5\}, \{2, 6\}, \{3, 4\}, \\ & \{1, 6\}, \{2, 4\}, \{3, 5\}\} \end{aligned}$$

Note that the 1-coterie is the coterie, and therefore, the k-coterie is an extension of the coterie. The followings are two simple classes of k-coteries, the k-majority coterie and the k-singleton coterie.

Definition 2 (k-majority coterie) *Let* $W = \lceil (n+1)/(k+1) \rceil$, *where* n *is the number of processes. The set* $\mathrm{Maj}_k = \{Q \subseteq U \mid |Q| = W\}$ *is called the* k*-majority coterie. Note that* Maj_k *is defined when* $n \geq k^2$. □

Definition 3 (k-singleton coterie) *Let* Sgl_k *be a set* $\{\{v_1\}, \ldots, \{v_k\}\}$, *where* $v_i \in U$ *for* $i = 1, \ldots, k$, *and* v_i*'s are distinct. Then* Sgl_k *is a* k*-coterie.*

□

3 An Algorithm for the Distributed k-Mutual Exclusion Problem

In this section, we introduce a k-coterie based distributed k-mutual exclusion algorithm. The basic idea of the algorithm is simple; select a quorum and get permissions from all processes in the quorum. But it is easy to see that straightforward implementation of the idea may cause deadlocks. The idea we propose to avoid deadlocks is to make permissions preemptive.

We consider the distributed k-mutual exclusion problem. A process may choose a *busy* quorum when it tries to enter a critical section and is forced to wait for a long time. The process may be able to enter a critical section immediately by selecting another quorum. Therefore, *retrying* is an important to reduce the delay time to enter a critical section, especially in case that k is large.

3.1 On Ordering in Distributed Environments

To avoid deadlocks and starvations, the timestamp introduced by Lamport [2] is used to define the priority among mutual exclusion requests. The timestamp carried by a message is assumed to have a form (t, x), where t is the logical time at which the message is issued and x is the process issuing the message.

Timestamps are orderd naturally as follows: $(t, x) < (u, y)$ iff $t < u$ or $t = u \,\&\, x < y$. If $(t, x) < (u, y)$ then the priority of (t, x) is higher than that of (u, y).

The logical clock can be implemented as follows. At first, the local clock of each process is initialized to be 0. If a mutual exclusion request is issued at a process x, then x increments local clock by 1. Whenever a process x sends a message to another processes, the current time t (in terms of the local clock) is attached to the message. When y receives it, then y sets its local clock to $\max\{t, t'\} + 1$, where t' is the current time by y's local clock.

3.2 The Algorithm

3.2.1 Overview of the Algorithm

An overview of the algorithm is as follows. Let $\mathcal{C}$ be a k-coterie. When a process x want to enter a critical section, x selects a quorum $Q \in \mathcal{C}$ and sends a REQUEST message to each process in Q. A process y in the quorum sends a OK message (i.e., it gives permission) unless it has sent permission to a process and it has not been returned. If y has sent the permission already, then it sends a WAIT message to x. The WAIT message means that "I cannot give the permission immediately". At this step, y may preempt the permission.

Then x waits for an acknowledge from each process in Q. If all the acknowledgements are OK, then x can enter a critical section. If there contains a WAIT message in the acknowledgements, then y selects another quorum and sends REQUEST messages. This is repeated until y can get the permissions from all processes in a quorum.

3.2.2 Complete Description of the Algorithm

Now, we show a detailed description of the algorithm. Each process x has local variables K and T. K of x maintains the set of processes from which x received OK messages, and T of x maintains the set of processes from which x has received WAIT messages. Later, we will show that the algorithm has the following properties: (1) k-mutual exclusion is guaranteed, (2) it is deadlock-free, and (3) it is starvation free.

- *When x wishes to enters a critical section:*
  Process x selects a quorum Q in $\mathcal{C}$, and sends REQUEST messages to all processes in Q. Then, x waits for acknowledgements (OK or WAIT) from all processes in Q. If there exists a process that sent a WAIT message in the acknowledgements, then x selects another quorum Q' such that $Q' \cap K = \emptyset$ and $|Q' - K| \geq |R - K|$ for all $R \in S$. And x sends REQUEST messages to processes in $Q' - K$, i.e., REQUEST messages are sent out at most once to each process. This action is repeated until $K \subseteq R$ for some $R \in \mathcal{C}$.

  Process x enters a critical section if it gets permissions from all processes in a quorum $R \in \mathcal{C}$.

  If x has no quorum to send request messages, then x waits for permissions from processes to which x sent request message.

- *When x exits from a critical section:*
  Process x sends a RELEASE message to each process in K to return permissions. And x waits for permissions from all processes in W. When x receives a OK message from y in W, x immediately sends a RELEASE message to y.

- *When x receives a* REQUEST *message from y:*
  If x has not sent permission to any processes, x sends a OK message to y. If x has sent permission to a process z, x acts as follows. Let (s, y), (t, z) be timestamps of requests of y, z, resp. If $(s, y) > (t, z)$, then the priority of z is higher than that of y; x sends a WAIT message to y and puts the request of y into x's local priority queue. (The head item of the queue is a request whose timestamp is the smallest in the queue.) If $(s, y) < (t, z)$, then priority of y is higher. To avoid deadlock, x try to preempt permission which is sent to z. But if z is in a critical section, x

cannot preempt; x sends a QUERY message to z to ask whether x can preempt the permission or not. If z is not in a critical section, then z sends a ANSWER_RELEASE and returns the permission. Then, x sends a OK message to y and puts request of z into the queue. If z is in a critical section, then z sends a ANSWER_NO. In this case, x sends a WAIT message to y and puts the request of y into its queue. (Note that the permission is returned to x in finite time as z exits from a critical section in finite time.)

- *When x receives a* RELEASE *message from y:*
  A permission of x is returned by y. If the queue of x is not empty, then x extracts the first item (t, z) in the queue and sends OK message to process z. If queue is empty, then x holds its permission.

- *When x receives a* QUERY *message from y.*
  If x is not in a critical section, then x returns permission, i.e., x sends ANSWER_RELEASE to y. If x is in a critical section, then x sends ANSWER_NO message to y.

3.3 Correctness Proofs

Theorem 1 *The algorithm guarantees k-mutual exclusion.*

Proof: Let $\mathcal{C}$ be a k-coterie which is used in the algorithm. A process x enters a critical section if and only if the following condition holds:

$$\exists Q(Q \in \mathcal{C})[Q \subseteq K].$$

(Note that K is the set of processes from which x has received an OK message.) Each process y sends permission at most one process at a time. Since $\mathcal{C}$ is a k-coterie, the number of processes which hold the above condition is at most k at a time. Therefore, the number of processes in a critical section is at most k at any given time. □

Theorem 2 *The algorithm is deadlock free.*

Proof: Assume that a deadlock happens. Consider the directed graph whose nodes are processes and links are edges defined as follows: there exists an edge from x to y in the graph if and only if y has the permission of a process z and x is requesting it, i.e., x is waiting for its release. Since the system is a deadlock state, there exists a cycle in the graph. Let $x_0, x_1, ..., x_{m-1}$ be processes that forms a cycle such that

$$x_0 \to x_1 \to \cdots \to x_{m-1} \to x_0,$$

and t_i be the priority (the timestamp when a mutual exclusion happened) of process x_i. Note that without loss of generality, we can assume that no process x_i is in a critical section. If such process x_i exists, it eventually exits from a critical section and releases the permissions it keeps and the cycle of the graph is broken in a finite time.

Each process x preempts its permission having sent to a process y if a new request whose priority (defined by timestamp) is higher than y's. Since a cycle is formed, the permission which is kept by $x_{i+1 \bmod m}$ is not preempted by x_i for all i. But $t_i > t_{i+1 \bmod m}$ holds for each i, we have $t_0 > t_0$; a contradiction. □

Theorem 3 *The algorithm is starvation free.*

Proof: Assume that there exists a process x which is starved. Let S be a set of processes to which x sent request message. Since x is starved, there exists (1) a process y such that y received a request of x, (2) a process z such that z enters a critical section infinitely often and receives a permission from y. But this situation can't happen. Since the timestamp of z increases when it enters a critical section and it becomes any large, the timestamp of z becomes larger than that of x in finite time. Therefore, eventually x can have higher priority and receives permission from y. □

3.4 Discussions

• Message complexity

Let $\mathcal{C}$ be a k-coterie used by our algorithm. In the best case, the number of messages that our algorithm requires per mutual exclusion is $3|Q|$, where Q is a quorum in $\mathcal{C}$. In case that $\mathcal{C} = \mathrm{Maj}_k$, the message complexity is $3\lceil (n+1)/(k+1) \rceil$. But we can find another k-coterie of whose quorum size is small. For instance, a method to construct a k-coterie whose quorum size is $\mathcal{O}(\sqrt{n}\log n)$ is shown in [5]. So, the message complexity of ours is smaller than that of Raymond's.

But, in worst case, our algorithm requires at most $6|N|$ messages per mutual exclusion. This case happens when a requesting process fails to preempt every permissions that a process requested. This seems to be serious problem, but we can restrict the maximum number of messages by modifying our algorithm. This modification is discussed below.

• Variation

We describe how to reduce the number of messages in worst case. The algorithm try to find another quorum if a quorum is busy; this 'retrying' causes increase of messages. Therefore, if retrying is restricted, then the the message complexity in worst case become smaller. As an extreme case, if we do not retry at all, then the message complexity in worst case is $6|Q|$. But this may causes long delay time to enter a critical section.

A process x sends ANSWER_NO message when x is in a critical section and receives QUERY message. But this message, ANSWER_NO, can be omitted. This modification decreases the number of messages in worst case; but it may cause long delay time if a process is in a critical section long time.

4 Experimental Evaluation of Distributed k-Mutual Exclusion Algorithms

In [6], a distributed k-mutual exclusion algorithm is proposed. The aim of this section is to show advantages of our algorithm comparing to Raymond's algorithm by experiment. The experiment is done by using workstations connected by a local area network (Ethernet). On each workstation, only one process executes a mutual exclusion algorithm; therefore, processes are executed completely in parallel. Message exchange between processes is implemented by using Inter Process Communication (IPC) facility.

4.1 Algorithm by Raymond

Now, we give a brief explanation of the distributed k-mutual exclusion algorithm proposed by Raymond. Her algorithm is a modification of Ricart and Agrawala's distributed 1-mutual exclusion algorithm [7]. Raymond's algorithm also uses the timestamp to define priority among mutual exclusion requests to avoid deadlock and starvation.

- *When a process enters a critical section.*
  A process x sends a REQUEST message to all the other process (i.e., $n-1$ processes). And when x receives $n-k$ REPLY (permission) messages, x can enter a critical section.

- *When a process receives a* REQUEST *message.*
  Let x be a process which sent a REQUEST message and y be a process which received the REQUEST message. If y is not requesting to enter a critical section nor in a critical section then y sends a REPLY message to x. Otherwise, if $t_x > t_y$, where t_x (t_y) is a timestamp of x (y), then y defers sending a REPLY to x until it exits from a critical section. if $t_x < t_y$ then y (immediately) sends REPLY message to x.

4.2 A Model of Distributed System

In section 2, we described that the processing speed and speed of time flow of any two processes are not the same in general. But, to evaluate two algorithms, we change these assumptions such that (i) processing speed of all processes are the same, and (ii) all processes have the same time flow, i.e., their clock are identical.

4.3 A Model of Behavior of Processes

Each process has four states and transits state to state on some events.

- **The Normal State**

  In this state, a process is passive, i.e., it may receive a message from other process, and may sends response. At every unit time, with probability $p\,(0 \leq p \leq 1)$, a request to enter a critical section happens at the process. If a request happens, it changes its state to the Requesting State. If not, it stays in the same state.

- **The Requesting State**

  In this state, a process is sending request messages to processes or waiting to get permissions from processes. If a condition to proceed to a critical section holds, then it enters a critical section and changes its state to the Critical Section State.

- **The Critical Section State**

  In this state, a process is in a critical section. After having Q_{CS} unit time passed since it entered a critical section, it exits from a critical section and changes its state to the Exiting State.

- **The Exiting State**

  In this state, a process is releasing permissions which is gotten to enter a critical section. When it finishes releasing all permissions, it changes its state to the Normal State.

4.4 The Evaluation System

Here, we describe the simulation system to evaluate two algorithms. The simulation is done by using workstations which are connected by local area network (Ethernet). Workstations on which UNIX operation system are available are used and the programming language C is used to implement mutual exclusion algorithms. On each workstation, only one process which executes a distributed mutual exclusion algorithm is executed. Therefore, n workstations are necessary to simulate a distributed system of n processes. Message exchange between processes is done by Inter Process Communication (IPC) facility.

We are assuming that the speed of time flow at each process is the same. To implement such situation, the simplest solution is letting the time flow of a process be the same as (or proportional to) that of real time. We let the time unit at processes be T_Q second. (In our experiment, one unit of time, T_Q is set to be 1 second.) Therefore, the speed of time flow at a process does not depend on the processing speed of workstation, i.e., the same time flow is guaranteed. Each workstation has real time clock; therefore this idea is be able to be implemented easily. Since 1 second is enough long time for CPUs, the local computation time at processes is negligibly short.

Since stream communication is synchronous, if two processes try to send message at the same time then these processes fail into deadlock state. (A process waits for message reception of the other process, and the other process waits for message reception of another one.) Therefore, message passing must be asynchronous. So, message exchange between processes is implemented by using (asynchronous) datagram communication.

4.5 Results and Discussions

Conditions of the experiment are as follows:

- the unit time T_Q is 1 second,
- T_{CS}, the time that a process is in a critical section, is 1 unit time,
- a k-coterie used by our algorithm is the k-majority coterie, and
- the experiment is done for 500 unit time.

The experiment is done for:

- $k = 2$, $n = 5, 8, 11$,
- $k = 3$, $n = 7$, and

- $k = 4$, $n = 9$.

For each experiment, p, the probability of mutual exclusion request, is varied from 0.01 to 1.0. Workstations used for the experiment are 7 AV-300's (Nippon Data General) and 4 DS-7400's (Nippon Data General) on which the DG/UX operating system (version 4.32 for AV300, version 4.02 for DS7400) is available[3]. The experiment system is written in the language C. The size of it is about 4000 lines (600 lines for our algorithm, 300 lines for Raymond's algorithm, 3000 lines for common subroutines).

Under condition as described above, following two are counted

- the number of messages that each process sends, and
- the number of times that each process enters a critical section.

From these two data, the average of the number of messages which is necessary to enter a critical section is computed. Let this value be μ, which is computed by the following formula.

$$\mu = \frac{\sum\limits_{1 \le i \le n} M_i}{\sum\limits_{1 \le i \le n} C_i},$$

where M_i is the number of messages that process i sends during the experiment and C_i be the number of times that process i enters a critical section during the experiment $(1 \le i \le n)$. [4]

Results of the experiment are shown in figure 1–5.

In case that p is small (for instance, in case of $k = 4$, $n = 9$; see figure 5), μ, the number of messages which our algorithm requires to enter a critical section, is smaller than that of Raymond's algorithm, as being expected. We can see from figures that μ gradually increases with the increase of p if p is small (for instance, $p < 0.2$ in case of $k = 4$, $n = 9$). But when p become larger, μ suddenly increases and when p comes near to 1.0, μ saturates. This

[3]the DG/UX operating system is a flavor of the UNIX operating system.

[4]For convenience, let process ID be an integer between 1 and n.

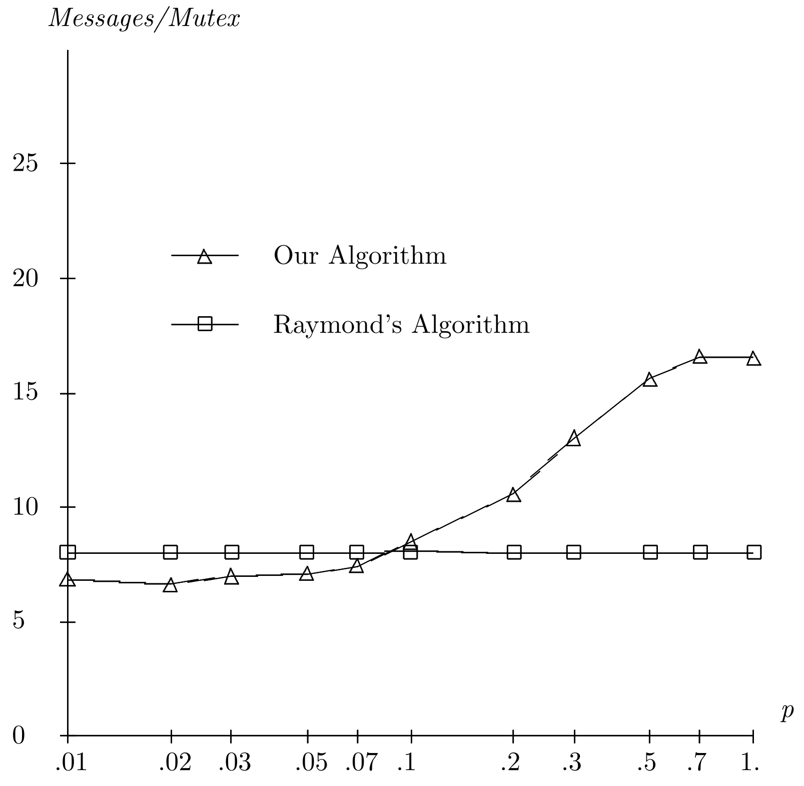

Figure 1: The average number of messages ($k = 2$, $n = 5$).

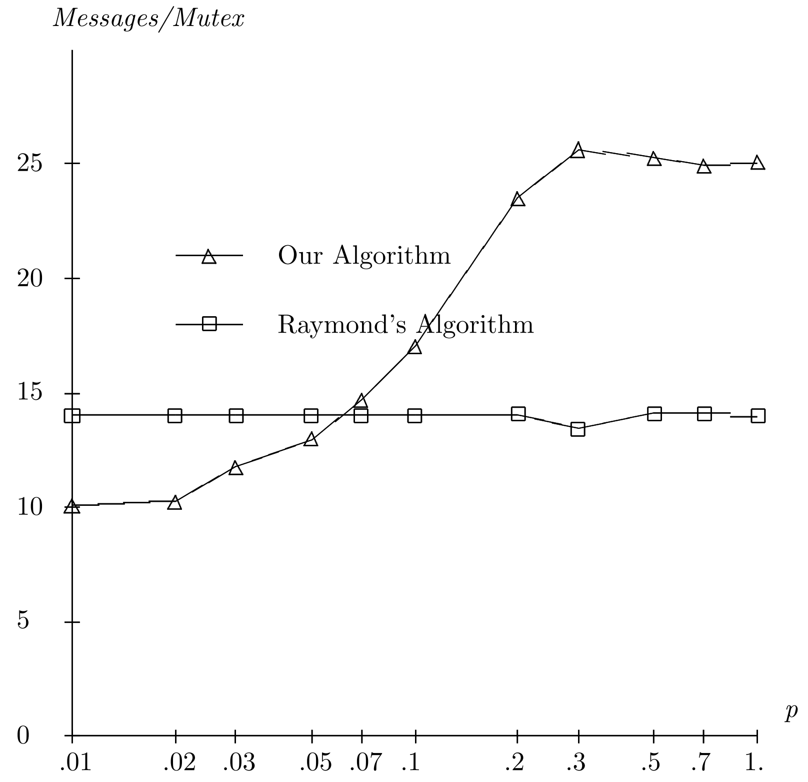

Figure 2: The average number of messages ($k = 2$, $n = 8$).

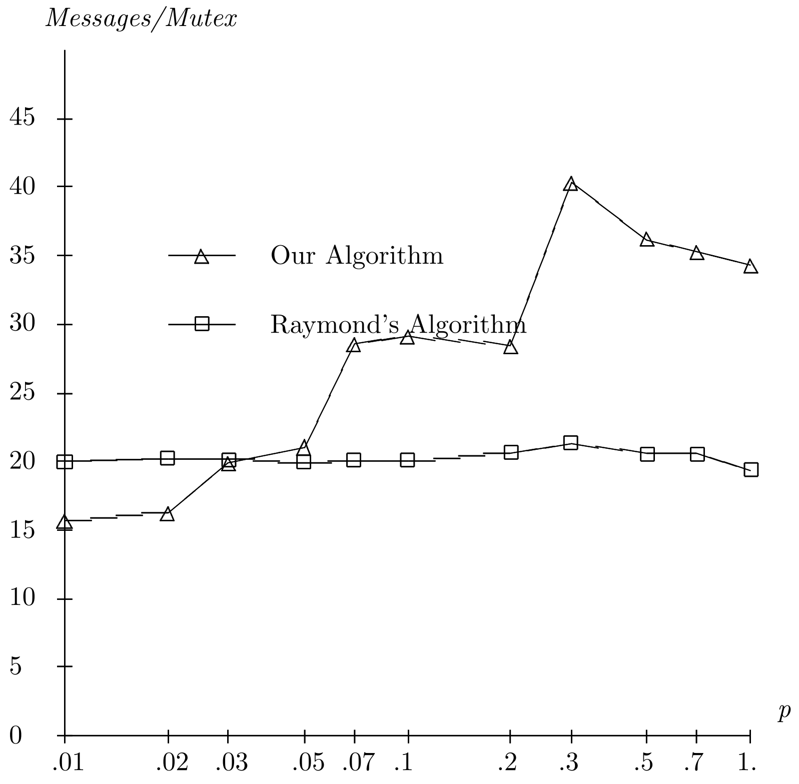

Figure 3: The average number of messages ($k = 2$, $n = 11$).

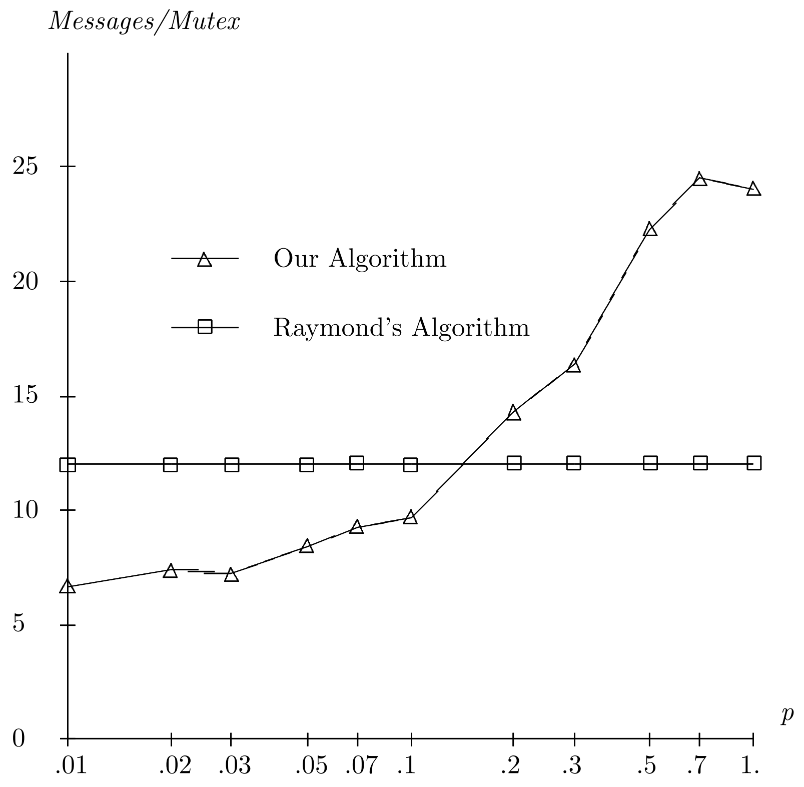

Figure 4: The average number of messages ($k = 3$, $n = 7$).

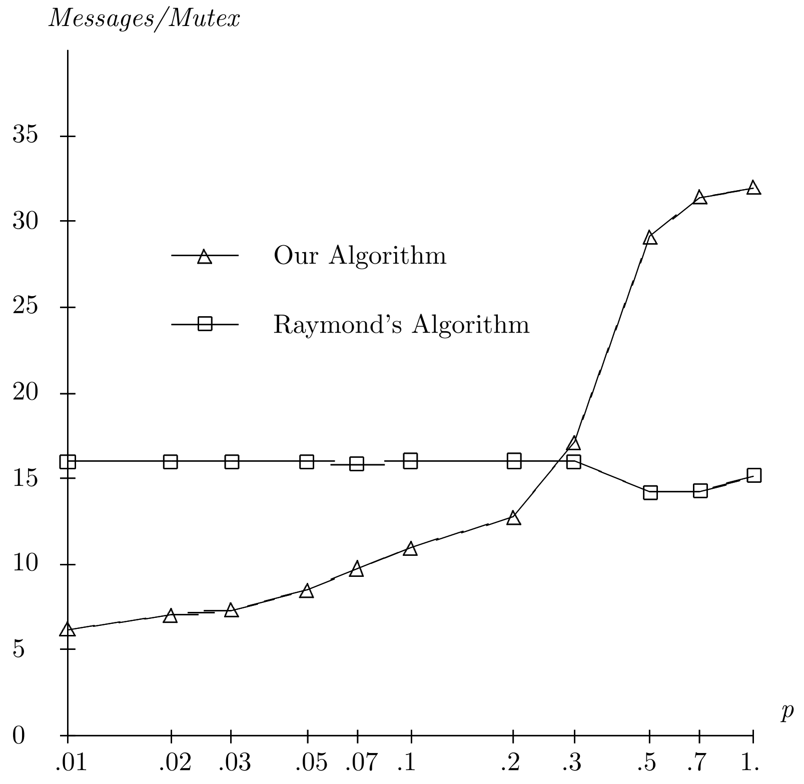

Figure 5: The average number of messages ($k = 4$, $n = 9$).

observation is described as follows. When p is enough small, mutual exclusion requests do not collide often with. In addition to it, even if a process fails to get permissions from a quorum, the probability that it gets permissions from a next quorum is large. Therefore, the number of additional messages is rather small. But p increases, collisions often happen and the probability that processes choose another coterie but fails to get permissions become large; this cause a sudden increase of μ.

Consider the case that k is fixed but n increases (see figures 1, 2, and 3). in this case, the increase of n causes the increase of the probability of collision of mutual exclusion requests. Therefore, μ increases.

The larger k becomes (for instance, compare cases $k = 2,\ n = 5$ and $k = 4,\ n = 9$; see figure 1 and figure 5), the smaller μ becomes (if p is small). This is why that the size of quorum become smaller if k become larger. Note that the k-coterie is used in this experiment. If we use other coteries whose quorum size is small, μ becomes smaller.

So, we can expect our algorithm requires less messages than Raymond's algorithm does. As discussed in previous section, evenif p is large, the μ can be limited by modifyng our algorithm such that the algorithm does not retry requesting so many quorums. Therefore, we can conclude that our algorithm is better than Raymond's algorithm.

5 Availability of k-Coterie

Consider a distributed system in whose processes may fail. This is often happen in real distributed systems. Under such circumstance, it is important to consider the fault tolerance of the system. In this section, the availability of k-coterie is investigated. The availability is one of criteria of distributed system, which is a probability that a system is operational. As a metric of goodness of k-coterie, the (k, r)-*availability* is introduced, which is an extension of the *availability* ([4]). We show a necessary and a sufficient conditions such that the k-majority coterie is optimal the k-coterie in terms of the (k, r)-availability.

5.1 Preliminary

Here, we define concepts which is necessary in this section.

Definition 4 ((k, r)-characteristic function) *Let $\mathcal{C}$ be a k-coterie over U, and r $(1 \leq r \leq k)$ be an integer. The (k, r)-characteristic function $F_{\mathcal{C},k,r}$ of $\mathcal{C}$ is a function from 2^U to $\{0, 1\}$ defined as follows:*

For each $S \subseteq U$, $F_{\mathcal{C},k,r}(S) = 1$ if and only if there exist r quorums $Q_1, ..., Q_r \in \mathcal{C}$ satisfying both of the following two conditions;

$$Q_i \cap Q_j = \emptyset \text{ for } 1 \leq i, j \leq r,\ i \neq j, \text{ and}$$
$$\text{for all } i,\ Q_i \subseteq S.$$

□

Definition 5 ((k, r)-availability) *Let $\mathcal{C}$ be a k-coterie, and $r\,(1 \leq r \leq k)$ be an integer. The (k, r)-availability $R_{k,r}(\mathcal{C})$ of $\mathcal{C}$ is the probability that at least r processes can enter a critical section.*

More formally, let $G = (V, E)$ be the topology of the distributed system under consideration. Let V' and E' be, respectively, the sets of processes and links in operation, and by $P_r(V', E')$, denote the probability that this situation occurs. The topology of the distributed system in operation is then the graph $G' = (V', (V' \times V') \cap E')$. We say a quorum $Q \in \mathcal{C}$ is available with respect to G' if Q is a subset of the vertex set of a connected component of G'. If there are r distinct available quorums $Q_1, \ldots, Q_r \in \mathcal{C}$ with respect to G' such that $Q_i \cap Q_j = \emptyset$ for $1 \leq i, j \leq r,\ i \neq j$, we say that G' is r-available. Then the (k, r)-availability of $\mathcal{C}$ on G is defined as follows:

$$R_{G,k,r}(\mathcal{C}) = \sum_{G' \text{ is } r\text{-available}} P_r(V', E')$$

The (k, r)-availability depends on G. But, since throughout this paper, we assume that G is complete, we omit G from $R_{G,k,r}$. □

Note that the (1,1)-availability coincides with the availability.

Let S be the set of processes being in operation. Then, since the topology of the distributed system is a complete graph, $F_{\mathcal{C},k,r}(S) = 1$ if and only if at least r processes can enter a critical section (i.e., $G' = (S, (S \times S) \cap E)$ is r-available). On the other hand, the probability that the set of processes being in operation is exactly S is $p^{|S|}(1-p)^{n-|S|}$. Thus, the (k,r)-availability of a coterie $\mathcal{C}$ can be calculated using the following formula:

$$R_{k,r}(\mathcal{C}) = \sum_{S \subseteq U} F_{\mathcal{C},k,r}(S) p^{|S|}(1-p)^{n-|S|}.$$

Let $\mathcal{C}$ be a k-coterie, and r $(1 \leq r \leq k)$ be an integer. Now, we construct a new k'-coterie $\mathcal{C}'$ as follows:
First, let

$$\begin{aligned} \mathcal{C}' &= \{Q \mid Q = Q_1 \cup \cdots \cup Q_r,\ Q_i \in \mathcal{C} \text{ for } 1 \leq i \leq r, \\ &\quad \text{and } Q_i \cap Q_j = \emptyset \text{ for } 1 \leq i, j \leq r,\ \ i \neq j\}. \end{aligned}$$

Next, from $\mathcal{C}'$, we remove all elements Q such that $Q' \subseteq Q$ for some $Q' \in \mathcal{C}'$, in order for the resultant $\mathcal{C}'$ satisfying the minimality property. Then $\mathcal{C}'$ has the following properties.

Property 1 *$\mathcal{C}'$ is a $\lfloor \frac{k}{r} \rfloor$-coterie.* □

Property 2 *Let $k' = \lfloor \frac{k}{r} \rfloor$. Then,*

$$F_{\mathcal{C},k,r} = F_{\mathcal{C}',k',1}.$$

Hence,

$$R_{k,r}(\mathcal{C}) = R_{k',1}(\mathcal{C}').$$

□

We call $\mathcal{C}'$ the r-contracted coterie of $\mathcal{C}$.

5.2 Availability of k-Majority Coterie

We investigate the k-majority coterie Maj_k in terms of the (k, r)-availability.

Theorem 4 *Let n be the number of processes, k be an integer such that $(n+1)$ is a multiple of $(k+1)$, and r $(1 \leq r \leq k)$ be an integer. Then, there is a constant $p_u(n, k, r)$ such that for any process reliability p $(p_u(n, k, r) \leq p \leq 1)$, Maj_k achieves the maximum (k, r)-availability. Hence, Maj_k is the best k-coterie in terms of the (k, r)-availability if $p \geq p_u(n, k, r)$, where*

$$p_u(n, k, r) = \frac{c(n, k, r)}{c(n, k, r) + 1},$$

and

$$c(n, k, r) = \sum_{i=0}^{rW-1} \binom{n}{i}.$$

Proof: Let $\mathcal{C}$ $(\neq \mathrm{Maj}_k)$ be any k-coterie. We show that $R_{k,r}(\mathrm{Maj}_k) \geq R_{k,r}(\mathcal{C})$ for any $p \geq p_u(n, k, r)$. Let $W = (n+1)/(k+1)$ (i.e., W is the size of each quorum in Maj_k). If each quorum Q in $\mathcal{C}$ has size at least W, then $R_{k,r}(\mathrm{Maj}_k) \geq R_{k,r}(\mathcal{C})$ clearly, since $F_{\mathrm{Maj}_k,k,r} \geq F_{\mathcal{C},k,r}$. Therefore, in $\mathcal{C}$, there exists a quorum Q_0 with size less than W.

First, we show that for some S $(\subseteq U)$ with size rW, $F_{\mathcal{C},k,r}(S) = 0$ holds. Suppose that for each S with size rW, $F_{\mathcal{C},k,r}(S) = 1$ holds. Let $U_1 = U - Q_0$. Since $|U_1| \geq n - W + 1$, $|U_1| \geq kW$. Arbitrarily choose a set S $(\subseteq U_1)$ with size rW. Since $F_{\mathcal{C},k,r}(S) = 1$, there is a quorum Q_1 $(\subseteq S)$ in $\mathcal{C}$ whose size is at most W. Then we repeat this procedure for $U_2 = U_1 - Q_1$. In this way, we repeat this procedure $(k - r)$ times and can find a sequence of quorums $Q_0, ..., Q_{k-r}$ in $\mathcal{C}$. Clearly, $Q_i \cap Q_j = \emptyset$ for $0 \leq i, j \leq (k - r)$, $i \neq j$. Since $|Q_i| \leq W$ for $0 \leq i \leq (k - r)$, $|U_{k-r+1}| \geq rW$. Thus, in U_{k-r+1}, there exist r quorums $Q_{k-r+1}, ..., Q_k (\in \mathcal{C})$, such that $Q_i \cap Q_j = \emptyset$ for $k - r + 1 \leq i, j \leq k, i \neq j$. It is a contradiction, since $Q_i \cap Q_j = \emptyset$ for $0 \leq i, j \leq k,\ \ i \neq j$.

Then, there exists a set S $(\subseteq U)$ with size rW such that $F_{\mathcal{C},k,r}(S) = 0$. Let $\Delta = R_{k,r}(\mathrm{Maj}_k) - R_{k,r}(\mathcal{C})$. Since $F_{\mathrm{Maj}_k,k,r}(S') = 1$ for every S' with size rW, by definition,

$k,\ W,\ n$	$r=1$	$r=2$	$r=3$	$r=4$	$r=5$	$r=6$
$k=1,\ W=5,\ n=9$	0.9961089	—	—	—	—	—
$k=2,\ W=5,\ n=14$	0.9993207	0.9999329	—	—	—	—
$k=3,\ W=4,\ n=15$	0.9982669	0.9999390	0.9999689	—	—	—
$k=4,\ W=3,\ n=14$	0.9906542	0.9997121	0.9999226	0.9999386	—	—
$k=5,\ W=3,\ n=17$	0.9935484	0.9998937	0.9999847	0.9999918	0.9999924	—
$k=6,\ W=3,\ n=20$	0.9952830	0.9999539	0.9999962	0.9999987	0.9999990	0.9999990

Table 1: $p_u(n,r,k)$ for some n $(k=1,...,6,\ r=1,...,k)$.

$$\begin{aligned}\Delta &\geq p^{|S|}(1-p)^{n-|S|} - \sum_{i=0}^{rW-1}\binom{n}{i}p^i(1-p)^{n-i}\\ &\geq p^{rW}(1-p)^{n-rW}\\ &\quad -c(n,k,r)p^{rW-1}(1-p)^{n-rW+1},\end{aligned}$$

where

$$c(n,k,r)=\sum_{i=0}^{rW-1}\binom{n}{i}.$$

It is easy to show that $\Delta \geq 0$ if

$$p \geq \frac{c(n,k,r)}{1+c(n,k,r)} = p_u(n,k,r).$$

$\square$

Since $c(n,k,r) < c(n,k,r+1)$, the following corollary holds.

Corollary 1 *If $p \geq p_u(n,k,k)$ then* Maj_k *is optimal in the sense of (k,r)-availability for all $1 \leq r \leq k$.* $\square$

Table 1 shows $p_u(n,k,r)$ $(k=1,\dots,6$ and $r=1,\dots,k)$ for some n.

Theorem 5 *For any non-negative integer m, $(2m+1)$-majority coterie* Maj_{2m+1} *achieves the maximum $(2m+1,m+1)$-availability, if the process reliability $p \geq \frac{1}{2}$ and $(n+1)$ is a multiple of $2(m+1)$.*

Proof: Let $\mathcal{C}$ ($\neq \mathrm{Maj}_{2m+1}$) be any $(2m+1)$-coterie, and assume that $\mathcal{C}$ achieves a better $(2m+1, m+1)$-availability than Maj_{2m+1} for some $p \geq \frac{1}{2}$. By $\mathcal{C}'$, we denote the $(m+1)$-contracted coterie of $\mathcal{C}$. Then by Property 1, $\mathcal{C}'$ is a 1-coterie. By definition of Maj_k , the $(m+1)$-contracted coterie of Maj_{2m+1} is 1-majority coterie Maj_1 , since $(n+1)$ is a multiple of $2(m+1)$. Since Maj_1 (i.e., majority coterie) achieves the maximum $(1,1)$-availability (i.e., availability) for all $p \geq \frac{1}{2}$ (Theorem 3.1 of [4]), the $(1,1)$-availability of Maj_1 is not smaller than that of $\mathcal{C}'$, a contradiction by Property 2. $\Box$

So far, we have derived a sufficient condition for k-majority coterie to be optimal in terms of the process reliability p. Now, we proceed to a necessary condition. We first present how to construct a new k-coterie $\mathcal{C}$ from k-majority coterie Maj_k , and then by comparing their (k,r)-availabilities, derive the necessary condition.

Arbitrarily choose n, k, and r (such that $(n+1)$ is a multiple of $(k+1)$, and fix them. We construct a k-coterie $\mathcal{C}$ from Maj_k as follows: Let Q_0 be any quorum in Maj_k, and u_0 be any element in Q_0. Let $Q_1 = Q_0 - \{u_0\}$. Then,

$$\begin{aligned}\mathcal{C} \quad = \quad & \mathrm{Maj}_k + \{Q_1\} - \{Q \in \mathrm{Maj}_k \bigm| Q = Q_1 \cup \{u\},\ u \in U - Q_1\} \\ & -\{Q \in \mathrm{Maj}_k \bigm| Q \cap Q_0 = \{u_0\}\}.\end{aligned}$$

We compare their availabilities. Observe that $F_{\mathcal{C},k,r}(S) = 1$ for all $S \subseteq U$ with size at least $rW+1$, and that $F_{\mathcal{C},k,r}(S) = 0$ for all $S \subseteq U$ with size at most $rW-2$, where $W = (n+1)/(k+1)$ (i.e., the size of quorum in Maj_k). On the other hand, by definition, $F_{\mathrm{Maj}_k,k,r}(S) = 1$ if and only if $|S| \geq rW$. Define Γ^+ and Γ^- as follows:

$$\begin{aligned}\Gamma^+ \quad &= \quad \{S \subseteq U \bigm| F_{\mathrm{Maj}_k,k,r}(S) = 0 \ \&\ F_{\mathcal{C},k,r}(S) = 1\} \\ \Gamma^- \quad &= \quad \{S \subseteq U \bigm| F_{\mathrm{Maj}_k,k,r}(S) = 1 \ \&\ F_{\mathcal{C},k,r}(S) = 0\}\end{aligned}$$

Note that by the observations, $|S| = rW - 1$ if $S \in \Gamma^+$, and $|S| = rW$ if $S \in \Gamma^-$. Since Q_1 is the only quorum with size $W - 1$ in $\mathcal{C}$, $S \in \Gamma^+$ if and only if $Q_1 \subseteq S$, $u_0 \notin S$, and $|S| = rW - 1$, by definition of $\mathcal{C}$. Therefore,

$$\begin{aligned} |\Gamma^+| &= \binom{n-W}{rW-1-(W-1)} \\ &= \binom{kW-1}{(r-1)W}. \end{aligned}$$

Next, we show that $S \in \Gamma^-$ if and only if $Q_1 \cap S = \emptyset$, $u_0 \in S$ and $|S| = rW$. To show if part, assume that $F_{\mathcal{C},k,r}(S) = 1$ holds (since $F_{\mathrm{Maj}_k,k,r}(S) = 1$). Since $u_0 \in S$, there is a quorum Q containing u_0 in $\mathcal{C}$, a contradiction since $Q \cap Q_0 = \{u_0\}$. As for only if part, if either $u_0 \notin S$ or $Q_1 \cap S \neq \emptyset$, then one can easily find r quorums $G_1, ..., G_r$ in $\mathcal{C}$ such that $S = \bigcup_{i=1}^{r} G_i$ and $G_i \cap G_j = \emptyset$ for $1 \le i, j \le r, \ \ i \neq j$. Therefore,

$$\begin{aligned} |\Gamma^-| &= \binom{n-W}{rW-1} \\ &= \binom{kW-1}{rW-1}. \end{aligned}$$

By definition,

$$\begin{aligned} \Delta &= R_{k,r}(\mathcal{C}) - R_{k,r}(\mathrm{Maj}_k) \\ &= |\Gamma^+| p^{rW-1}(1-p)^{n-(rW-1)} - |\Gamma^-| p^{rW}(1-p)^{n-rW} \\ &= p^{rW-1}(1-p)^{n-rW} \\ &\quad \times \left\{ \binom{kW-1}{(r-1)W}(1-p) - \binom{kW-1}{rW-1} p \right\}. \end{aligned}$$

Therefore, $\Delta > 0$ if and only if

$$p > \frac{\binom{kW-1}{(r-1)W}}{\binom{kW-1}{(r-1)W} + \binom{kW-1}{rW-1}}.$$

Theorem 6 *Let n be the number of processes, k be an integer such that $(n+1)$ is a multiple of $(k+1)$, and r $(1 \le r \le k)$ be an integer. Then, there is a*

$k,\ W,\ n$	$r=1$	$r=2$	$r=3$	$r=4$	$r=5$	$r=6$
$k=1,\ W=5,\ n=9$	0.5000000	—	—	—	—	—
$k=2,\ W=5,\ n=14$	0.0078740	0.9921260	—	—	—	—
$k=3,\ W=4,\ n=15$	0.0060241	0.5000000	0.9939759	—	—	—
$k=4,\ W=3,\ n=14$	0.0178571	0.2631579	0.7368421	0.9821429	—	—
$k=5,\ W=3,\ n=17$	0.0108696	0.1538462	0.5000000	0.9891304	0.9891304	—
$k=6,\ W=3,\ n=20$	0.0072993	0.0099010	0.3373494	0.6626506	0.9009901	0.9927007

Table 2: $p_l(n, r, k)$ for some n $(k = 1, ..., 6,\ r = 1, ..., k)$.

constant $p_l(n, k, r)$ such that for any process reliability p $(0 < p < p_l(n, k, r))$, Maj$_k$ *does not achieve the maximum (k, r)-availability.Hence,* Maj$_k$ *is not the best k-coterie in terms of (k, r)-availability if $0 < p < p_l(n, k, r)$, where*

$$p_l = \frac{\binom{kW-1}{(r-1)W}}{\binom{kW-1}{(r-1)W} + \binom{kW-1}{rW-1}}.$$

□

Table 2 shows $p_l(n, k, r)$ $(k = 1, \ldots, 6,\ r = 1, \ldots, k)$ for some n.

5.3 Availability of k-Singleton Coterie

This section shows a sufficient condition for the k-singleton coterie to be optimal in terms of the process reliability p.

Theorem 7 *Let n be the number of processes, and k $(\leq n)$ and r $(1 \leq r \leq k)$ be integers. Then, there exists a constant $q(n, k, r) > 0$ such that (any) k-singleton coterie* Sgl$_k$ *is optimal for all process reliability p $(0 \leq p \leq q(n, k, r))$. Hence,* Sgl$_k$ *is the best k-coterie in the sense of (k, r)-availability if $p \leq q(n, k, r)$.*

Proof: Let $\mathcal{C}$ be any k-coterie which is not a k-singleton coterie. We show that there exists a constant $t > 0$ such that for all process reliability p $(0 \leq p \leq t)$,

$k,\ r$		p										
		0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
4, 1	Maj$_k$	0.0000	0.1584	0.5519	0.8392	0.9602	0.9935	0.9994	1.0000	1.0000	1.0000	1.0000
	Sgl$_k$	0.0000	0.3439	0.5904	0.7599	0.8704	0.9375	0.9744	0.9919	0.9984	0.9999	1.0000
4, 2	Maj$_k$	0.0000	0.0015	0.0439	0.2195	0.5141	0.7880	0.9417	0.9917	0.9996	1.0000	1.0000
	Sgl$_k$	0.0000	0.0523	0.1808	0.3483	0.5248	0.6875	0.8208	0.9163	0.9728	0.9963	1.0000
4, 3	Maj$_k$	0.0000	0.0000	0.0004	0.0083	0.0583	0.2120	0.3373	0.6405	0.8883	0.9985	1.0000
	Sgl$_k$	0.0000	0.0037	0.0272	0.0837	0.1792	0.3125	0.4752	0.6517	0.8192	0.9477	1.0000
4, 4	Maj$_k$	0.0000	0.0000	0.0000	0.0000	0.0006	0.0065	0.0398	0.1608	0.4481	0.8416	1.0000
	Sgl$_k$	0.0000	0.0001	0.0016	0.0081	0.0256	0.0625	0.1296	0.2401	0.3164	0.5220	1.0000

Table 3: (k, r)-availabilities of Maj$_k$ and Sgl$_k$ $(k = 4,\ n = 14)$.

the (k,r)-availability of Sgl_k is larger than or equal to that of $\mathcal{C}$. The proof here is similar to that of Theorem 1.

Let $\Delta = R_{k,r}(\mathrm{Sgl}_k) - R_{k,r}(\mathcal{C})$. By definition, for all S with size at most $r-1$, $F_{\mathrm{Sgl}_k,k,r}(S) = F_{\mathcal{C},k,r}(S) = 0$. Define

$$\begin{aligned} m_0 &= \Big|\{S \mid F_{\mathrm{Sgl}_k,k,r}(S) = 1, |S| = r\}\Big|, \text{ and} \\ m_1 &= \Big|\{S \mid F_{\mathcal{C},k,r}(S) = 1, |S| = r\}\Big|. \end{aligned}$$

Then, clearly, $m_0 > m_1$, since $\mathcal{C}$ is not a k-singleton coterie. Therefore, by definition,

$$\Delta \geq p^r(1-p)^{n-r} - \sum_{i=r+1}^{n} \binom{n}{i} p^i (1-p)^{n-i}.$$

It is easy to see that there is a constant t such that $\Delta \geq 0$ for all p $(0 \leq p \leq t)$.

Since the number of different k-coteries are finite, the theorem follows. □

6 Conclusion

In this paper, we ingestigated the distributed k-mutual exclusion problem. To solve the problem, we introduced the k-coterie. An algorithm which is based on the k-coterie was proposed and its correctness (k-mutual exclusion, deadlock-free, starvation-free) was shown. An experiment was done in distributed environment to show the advantages of our algorithm by compareing ours with Raymond's algorithm. We concluded that ours is more efficient. From a view point of reliability of distributed systems, a necessary and a sufficient conditions such that the k-majority coterie is optomal in the sense of (k,r)-availability.

The scheme we discussed in this paper, the k-coterie scheme, does not provide *full* identical entrances of a critical section. The reason is that a process which is failed to get permissions from all processes in a quorum must find other quorum; this may correspond to finding another entrance, in a sense. (But entrances are not fully different.) So, it is an interesting problem finding a new scheme which provides fully identical entrances of a critical section. That is, a process which wishes to enter a critical section does not have to retry. This is left as a future task.

Acknowledgements

The author wishes to express his deep gratitude to Professor T. Ae, Associate Professor M. Yamashita, Assistant S. Fujita in Computer Systems Laboratory, Associate Professor R. Aibara in Research Center for Integrated Systems of Hiroshima University, for their patience and insightful guidance.

He would like to express sincere thanks to Professor K. Onaga in Hiroshima University for his helpful advice and discussions of this thesis. He is deeply indebted to Professor T. Ichikawa, Professor N. Yoshida, Associate Professor K. Ito, Associate Professor S. Wakabayashi, and Associate Professor T. Watanabe in Hiroshima University.

He also would like to express thanks to the members in Computer Systems Laboratory of Hiroshima University, especially, Mr. Y. Chikamatsu, Mr. K. Agusa, and Mr. Y. Nishioka for their friendly help and valuable discussions.

References

[1] Raymond, K.: A Tree-Based Algorithm for Distributed Mutual Exclusion, *ACM Trans. Comput. Syst.*, Vol. 7, No. 1, pp. 61–77 (1989).

[2] Lamport, L.: Time, Clocks, and the Ordering of Events in a Distributed System, *Commun. ACM*, Vol. 21, No. 7, pp. 558–565 (1978).

[3] R.H.Thomas, : A mojority consencus approach to concurrency controle for multiple copy databeses, *ACM Transactions on Database Systems*, Vol. 4, No. 2 (1979).

[4] Barbara, D. and Garcia-Molina, H.: The Reliability of Voting Mechanisms, *IEEE Trans. Comput.*, Vol. C-36, No. 10, pp. 1197–1208 (1987).

[5] Fujita, S., Yamashita, M. and Ae, T.: A non trivial solution of the distributed k-mutual exclusion problem, IEICE Japan, SIG Computation Record COMP90–99, p.p. 87–95 (in Japanese) (1991).

[6] Raymond, K.: A Distributed Algorithm For Multiple Entries To A Critical Section, *Inf. Process. Lett.*, Vol. 30, pp. 189–193 (1989).

[7] Ricart, G. and Agrawala, A. K.: An Optimal Algorithm for Mutual Exclusion in Computer Network, *Commun. ACM*, Vol. 24, No. 1, pp. 9–17 (1981).

[8] Garcia-Molina, H. and Barbara, D.: How to Assign Votes in a Distributed System, *J. ACM*, Vol. 32, No. 4, pp. 841–860 (1985).

[9] Agrawal, D. and Abbadi, A. E.: An Efficient solution to the Distributed Mutual Exclusion Problem, in *Principles of Distributed Computing*, pp. 193–200 (1989).

[10] Ricart, G. and Agrawala, A. K.: Author's response to 'On mutual exclusion in computer networks' by Carvalho and Roucairol, *Commun. ACM*, Vol. 26, No. 2, pp. 147–148 (1983).

[11] Suzuki, I. and Kasami, T.: A Distributed Mutual Exclusion Algorithm, *ACM Trans. Comput. Syst.*, Vol. 3, No. 4, pp. 344–349 (1985).

[12] Maekawa, M.: A $\sqrt{N}$ Algorithm for Mutual Exclusion in Decentralized Systems, *ACM Trans. Comput. Syst.*, Vol. 3, No. 2, pp. 145–159 (1985).

[13] Mizuno, M., L.Neilsen, M. and Rao, R.: A Token Based Distributed Mutual Exclusion Algorithm based on Quorum Agreements, in *Proc. of 11th International Conference on Distributed Computing Systems*, pp. 361–368 (1991).

[14] Singhal, M.: A class of deadlock-free Maekawa-type algorithms for mutual exclusion in distributed systems, *Distributed Computing*, pp. 131–138 (1991).

[15] J.L.Peterson, and A.Silberchatz, : *Operating Systems Concepts 2nd Ed.*, Addison-Wesley, Reading, MA (1987).

[16] Maekawa, M., E.Oldehoft, A. and R.Oldehoft, R.: *Operating Systems – Advanced Concept*, The Benjamin/Cummings Publishing Company (1987).

[17] Misra, J.: Distributed Discrete-Event Simulation, *ACM Computing Surveys*, Vol. 18, No. 1, pp. 39–65 (1986).

[18] Bal, H. E.: Programming Languages for Distributed Simulation Computing Systems, *ACM Computing Surveys*, Vol. 21, No. 3, pp. 261–322 (1989).

[19] Bagrodia, R. L. and Shen, C.-C.: MIDAS: Integrated Design and Simulation of Distributed Systems, *IEEE Trans. Softw. Eng.*, Vol. 17, No. 18, pp. 1042–1058 (1991).

[20] Ono, M.: Distributed Algorithm Simulator running in Distributed Systems (Graduate thesis, Hiroshima University) (1991).

[21] Ikegawa, Y., Yamashita, M. and Ae, T.: An Experimental Study on Leader Election Algorithms for Ring Networks, *Transactions on the IEICE Japan*, Vol. J73–D–I,3 p.p. 261–268, (1989).

[22] Nagamatsu, S.: A Dynamic Load Balancing Algorithms for Tree-structured Network (Graduate thesis, Hiroshima University) (1988).

[23] Tokura, N.: A Tool for Distributed Algorithm Simulation, *Information Processing*, Vol. 30, No. 4, pp. 380–386 (1989).

[24] Ibaraki, T. and Kameda, T.: Theory of Coteries, CSS/LCCR TR90-09, Simon Fraser University, Burnaby, B.C. Canada (1990).

[25] Raynal, M.: *Algorithms for Mutual Exclusion*, North Oxford Academic (1986), (Translated by D. Beeson).

[26] Cheung, S. Y., Ahamad, M. and Ammar, M. H.: Multi-Dimentional Voting: A General Method for Implementing Syncronization in Distributed Systems, in *Proceedings of 10th International Conference of Distributed Computing Systems*, pp. 362–369 (1990).