ELSEVIER

Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®
Theoretical Computer Science 310 (2004) 379–392

Theoretical Computer Science

www.elsevier.com/locate/tcs

(h,k)-Arbiters for h-out-of-k mutual exclusion problem

Yoshifumi Manabe[a,*], Naka Tajima[b,1]

[a] *NTT Communication Science Laboratories, NTT Corporation, 2-4 Hikaridai, Seika-cho, 619-0237 Kyoto, Japan*
[b] *University of Aizu, Tsuruga, Ikki-machi, Aizu-Wakamatsu City, Fukushima 965-8580, Japan*

Received 19 August 2002; received in revised form 24 July 2003; accepted 22 August 2003
Communicated by D. Peleg

Abstract

h-Out-of-k mutual exclusion is a generalization of the 1-mutual exclusion problem, where there are k units of shared resources and each process requests h $(1 \leqslant h \leqslant k)$ units at the same time. Though k-arbiter has been shown to be a quorum-based solution to this problem, quorums in k-arbiter are much larger than those in the 1-coterie for 1-mutual exclusion. Thus, the algorithm based on k-arbiter needs many messages. This paper introduces the new notion that each request uses different quorums depending on the number of units of its request. Based on the notion, this paper defines two (h,k)-arbiters for h-out-of-k mutual exclusion: a uniform (h,k)-arbiter and a $(k+1)$-cube (h,k)-arbiter. The quorums in each (h,k)-arbiter are not larger than the ones in the corresponding k-arbiter; consequently, it is more efficient to use (h,k)-arbiters than the k-arbiters. A uniform (h,k)-arbiter is a generalization of the majority coterie for 1-mutual exclusion. A $(k+1)$-cube (h,k)-arbiter is a generalization of square grid coterie for 1-mutual exclusion.

Keywords: Distributed algorithm; Mutual exclusion; Arbiter; Quorum

1. Introduction

Mutual exclusion is a fundamental problem in distributed systems. When resources, such as files, printers, and communication lines, are shared by multiple processes,

* Corresponding author.
E-mail address: manabe@cslab.kecl.ntt.co.jp (Y. Manabe).
[1] Part of this work was done while the author was staying at NTT Laboratories.

0304-3975/$ - see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2003.08.003

they must be allocated so that each resource is not allocated to multiple processes at the same time. Many algorithms for distributed systems have been proposed with a view to solving this mutual exclusion problem [20,21]. The first case to be considered was one where there is one shared resource, i.e., a 1-*mutual exclusion problem*. More recently considered was the case, where there are k units of an identical shared resource [17], i.e., a k-*mutual exclusion problem*. Many algorithms for the k-mutual exclusion problem have been reported [1–3,6–10,13,15–17,22,24].

In the k-mutual exclusion problem, each process can request one unit of the shared resource at the same time. The h-out-of-k mutual exclusion approach allows every process to request h $(1 \leqslant h \leqslant k)$ units of the shared resource at the same time [18]. One example of this type of mutual exclusion is the sharing of a communication bandwidth. A communication line with a fixed bandwidth k is shared by several processes. Each process effects audio and video communication over the line. Because the bandwidths necessary for audio and video communication differ greatly, the required bandwidth differs from request to request.

Another example is the power consumption of electric appliances. The total power consumption in a room must be less than a fixed number of units to avoid tripping the circuit breaker or overheating the wire. If the appliances are connected to a network, they may be able to keep their total power usage to less than k units at any time. The power consumption of each electric appliance differs from request to request, for example, an air conditioner's power usage varies according to ambient temperature.

While the h-out-of-k mutual exclusion problem can be solved by using a k-mutual exclusion algorithm and executing h requests when a process needs h units of the shared resource, two problems arise. The first problem is poor efficiency. It would be preferable for the requesting process to make one rather than multiple requests. The second problem is potential deadlock. Assume that there are k units and two processes, p_1 and p_2. Process p_1 requests h_1 units and process p_2 requests h_2 units, but $h_1 + h_2 > k$. If process p_1 can only obtain $h'_1 (< h_1)$ units, process p_2 can only obtain $h'_2 (< h_2)$ units, and $h'_1 + h'_2 = k$, a deadlock is created. An additional mechanism is thus needed to avoid this deadlock in order to solve the h-out-of-k mutual exclusion when using a k-mutual exclusion algorithm. Thus, this paper discusses obtaining h units by making one request.

Distributed mutual exclusion algorithms can be classified into three main groups: broadcast-based algorithms, token-based algorithms, and quorum-based algorithms. In outline a broadcast-based algorithm is as follows: A process that wants h units sends a request to every other process. Each process that receives this request replies when at most $k - h$ units are currently used. By receiving a reply from every process, the requesting process can use h units. The requesting process sends a release message to every other process when it finishes using the shared resource. This algorithm uses $3(n - 1)$ messages.

In token-based algorithms, a *permission* is represented by a token. If a process has a token, it can access the shared resource. Token-based algorithms generally perform well with regard to the number of messages exchanged to obtain the shared resource.

In quorum-based algorithms, a permission is divided into pieces that are distributed among the processes. Each process votes its piece to a requesting process. If a process

can obtain enough pieces from the other processes to construct a permission, it can access the shared resource. A set of processes whose pieces can construct a permission is called a *quorum*. The concept of mutual exclusion by voting was introduced by Gifford [5] and Thomas [23] and then generalized as coterie by Garcia-Molina and Barbara [4].

Broadcast-based and token-based algorithms suffer from poor failure resiliency. In contrast, quorum-based algorithms tolerate failures of nodes and network partitions. For example, assume that the communication is asynchronous and process might fail in accordance with the fail-stop model [19]. In token-based algorithms, if the current token holder process fails, the other processes need to generate a new token. However, it is impossible to distinguish a token holder's fail-stop and a long delay in the messages from the token holder. Thus, it is very hard to generate a new token without violating the mutual exclusion property. In broadcast-based algorithms, the requesting process cannot receive enough replies even if the number of fail-stop processes is one, because the requesting process cannot distinguish between a fail-stop process and a long delay in a message from the process. On the other hand, since each quorum is a subset of processes, there can be a fault-free quorum. Thus, by sending a request to a set of processes that is a superset of quorums, the requesting process can receive replies from every process in a quorum even if a fail-stop occurs. The set of processes used to send requests is determined by the set of quorums and f, which is the maximum number of fail-stop processes. The details of the fault-tolerance procedure is beyond the scope of this paper and the following discussion assumes that there is no failure or the failure process is detected. We thus focused on using quorum-based algorithms for the h-out-of-k mutual exclusion.

The k-arbiter has been shown to be a quorum-based distributed algorithm for the h-out-of-k mutual exclusion [14]. However, each quorum is larger than that in a 1-coterie for the 1-mutual exclusion algorithm. We thus looked at reducing the size of the quorums. The quorum-based distributed algorithm based on k-arbiter uses the same quorum for any request. We propose a new quorum-based solution that uses an (h,k)-arbiter. This (h,k)-arbiter is a quorum set for each h $(1 \leqslant h \leqslant k)$, $\{Q_{h,k} \mid 1 \leqslant h \leqslant k\}$, where $Q_{h,k}$ is a set of quorums. A process uses a quorum in $Q_{h,k}$ when it wants to use h units of the shared resource.

The effectiveness of selecting different quorum for each h is as follows. A requesting process p whose requesting unit h_1 is close to k, p can detect that its request is blocked when it knows the existence of a small number of other requests. By contrast, another requesting process p' whose requesting unit h_2 is close to 1, p' can detect that its request is blocked when it knows the existence of a larger number of other requests. Thus, the set of processes h_1's quorum intersects to can be small. Therefore, selecting different quorum for each h reduces the size of quorum when h is large.

This paper shows the condition that (h,k)-arbiter must satisfy and presents two (h,k)-arbiters. One is a uniform (h,k)-arbiter, which is a generalization of the uniform k-arbiter. The other one is a $(k+1)$-cube (h,k)-arbiter, which is a generalization of the $(k+1)$-cube k-arbiter. The quorums in each (h,k)-arbiter are not larger than these in the corresponding k-arbiter. Thus, h-out-of-k mutual exclusion can be solved more efficiently by using an (h,k)-arbiter than with a k-arbiter. Moreover, each quorum in

$Q_{k,k}$ of the (h,k)-arbiter can be no smaller than that in the 1-coterie for 1-mutual exclusion because obtaining all units is exactly the same as obtaining the shared resource in 1-mutual exclusion. Each quorum in $Q_{k,k}$ of the uniform (h,k)-arbiter is exactly the same as that in the majority coterie for 1-mutual exclusion. Thus, the uniform (h,k)-arbiter can be considered to be a generalization of the majority coterie. Each quorum in the $(k+1)$-cube (k,k)-arbiter when $k=1$ is about the same size as that in the square grid coterie [12] for 1-mutual exclusion, whose size is minimum. Thus, $(k+1)$-cube (h,k)-arbiter can be considered to be a generalization of the square grid coterie. This paper also shows a distributed h-out-of-k mutual exclusion algorithm using (h,k)-arbiters. Then, its correctness is presented.

Section 2 provides the definition of the problem. Section 3 shows the (h,k)-arbiter and Section 4 gives a distributed algorithm using (h,k)-arbiter. Section 5 concludes the paper and shows further study.

2. Problem definition and previous results

2.1. Distributed system

A distributed system is a set $U=\{p_1, p_2, \ldots, p_n\}$ of n processes. These processes communicate by exchanging messages. Each pair of processes is connected by a logical channel, and the message delay is unpredictable but finite. Moreover, each channel is assumed to have infinite capacity, to be error-free, and first-in, first-out (FIFO). Processes do not share either a common clock or a common memory. No bound exists to the relative speed of processes. The processes fail in accordance with the fail-stop model [19], and a failure can be detected by the other processes.

2.2. h-Out-of-k mutual exclusion problem

The h-out-of-k mutual exclusion problem is defined as follows [18]. There are k identical units of a resource that is shared by the processes in U. No unit must be allocated to more than one process at the same time. A process requests h $(1 \leqslant h \leqslant k)$ units of the resource simultaneously and, to avoid deadlock, the process is blocked until it obtains all its requested units. The process can then start using the units; when finished, it releases them all at once. If $h=1$ for every request, the h-out-of-k mutual exclusion problem corresponds to the k-mutual exclusion problem; moreover, if $k=1$ we obtain 1-mutual exclusion.

The h-out-of-k mutual exclusion algorithm must satisfy two properties:

Safety: Each unit of the resource may be used by at most one process at any given time.

Liveness: All requests must eventually be satisfied.

2.3. Quorums, 1-coteries, and k-arbiters

The following is the definition of coterie [4] making it possible to achieve safety for the 1-mutual exclusion problem.

Definition 1. A quorum q under U is a non-empty subset of U.

A set of quorums $Q=\{q_1,q_2,\ldots,q_m\}$ is a 1-coterie under U iff the following properties hold:

- *Intersection*: For any pair of quorums, $q_i,q_j\in Q :: q_i\cap q_j\neq\emptyset$.
- *Minimality*: For any pair of distinct quorums, $q_i,q_j\in Q :: q_i\not\subseteq q_j$.

An outline of the mutual exclusion algorithm using 1-coteries is as follows. A process selects a quorum from a 1-coterie, sends a request to each process in the quorum, and waits to receive permissions from them. After receiving the permissions, the process can use the shared resource. Safety is guaranteed from the intersection property. Minimality is introduced to remove unnecessary elements. If $q_i\subseteq q_j$, q_j is unnecessary because using q_j needs more communication than using q_i.

We briefly mention two examples of 1-coteries that will be used in the following:

(1) *Majority coteries*: $Q=\{q\subset U\,|\,|q|=\lfloor n/2\rfloor+1\}$, where $n=|U|$.

(2) *Square grid coteries* [12]: Assume that the processes in U are structured into a square grid $(1\ldots\sqrt{n},1\ldots\sqrt{n})$.

$$Q=\{q_{i,j}\,|\,1\leqslant i\leqslant\sqrt{n},1\leqslant j\leqslant\sqrt{n}\},$$
$$\text{where } q_{i,j}=\{(x,y)\,|\,x=i\}\cup\{(x,y)\,|\,y=j\}.$$

The size of each quorum is $O(\sqrt{n})$.

In order to solve k-mutual exclusion, a k-arbiter has been defined [14]. Its intersection property differs from that for 1-mutual exclusion as follows.

Definition 2. A set of quorums $Q=\{q_1,q_2,\ldots,q_m\}$ is a k-arbiter under U iff the following properties hold:

- *Intersection*: For any $(k+1)$ quorums, $q_{i_1},q_{i_2},\ldots,q_{i_{k+1}}\in Q :: \bigcap_{1\leqslant j\leqslant k+1} q_{i_j}\neq\emptyset$.
- *Minimality*: For any pair of distinct quorums, $q_i,q_j\in Q :: q_i\not\subseteq q_j$.

When $k=1$, the k-arbiter becomes a 1-coterie. Two examples of k-arbiters are as follows [14]:

(1) *Uniform k-arbiter*: $Q=\{q\subset U\,|\,|q|=\lfloor k\cdot n/(k+1)\rfloor+1\}$.

(2) $(k+1)$-*cube k-arbiter*: Suppose that $n=a^{k+1}$ for some integer a. The processes are structured into a $(k+1)$-dimensional hypercube. Each process is represented by $(x_1,x_2,\ldots,x_{k+1})$ $(0\leqslant x_i\leqslant a-1, 1\leqslant i\leqslant k+1)$.

$$Q=\{q^{b_1,\ldots,b_{k+1}}\,|\,0\leqslant b_j\leqslant a-1,1\leqslant j\leqslant k+1\},$$
$$\text{where } q^{b_1,\ldots,b_{k+1}}=\{(x_1,x_2,\ldots,x_{k+1})\,|\,x_1=b_1\}\cup\{(x_1,x_2,\ldots,x_{k+1})\,|\,x_2=b_2\}$$
$$\cup\cdots\cup\{(x_1,x_2,\ldots,x_k,x_{k+1})\,|\,x_k=b_k\}\cup\{(x_1,x_2,\ldots,x_{k+1})\,|\,x_{k+1}=b_{k+1}\}.$$

The size of each quorum is at most $(k+1)n^{k/(k+1)}$.

When $n\neq a^{k+1}$, the above construction is easily modified to obtain a k-arbiter [14].

The uniform k-arbiter becomes the majority coterie when $k=1$. The $(k+1)$-cube k-arbiter becomes the square grid coterie when $k=1$. Note that the lower bound on the size of each quorum of the symmetric k-arbiter is $\Omega(n^{k/(k+1)})$ [14]. Thus, a $(k+1)$-cube

k-arbiter satisfies the lower bound when k is a constant. An outline of the h-out-of-k mutual exclusion algorithm using a k-arbiter is as follows. When a process wants to use h units of the shared resource, it selects a quorum from a k-arbiter, sends a request to each process in the quorum, and waits to receive permissions from them. After receiving the permissions, the process can use the shared resource. When a process receives a request to use h units from a process p, it sends permission to p if the number of units in the requests it sends permission for is not greater than $k-h$. Safety is guaranteed from the intersection property. The minimality is exactly the same as that for the 1-coterie.

3. (h,k)-Arbiter

3.1. Definition of (h,k)-arbiter

We assume that each process uses a different arbiter for each h $(1 \leqslant h \leqslant k)$. An (h,k)-arbiter, $\mathcal{Q}_{h,k}$, is a set of arbiters $\{Q_{h,k} \mid 1 \leqslant h \leqslant k\}$. If a process p wants to use h units of the shared resource, p selects a quorum $q_{h,k} \in Q_{h,k}$ and executes the same procedure as that for using a k-arbiter. The algorithm to send a permission is exactly the same as that for a k-arbiter. The intersection property of the (h,k)-arbiter differs from that for a k-arbiter, as shown in the following.

A *set* has no repeated elements, while a *bag* may have repeated elements: $\{1,2,2,3\}$ is a bag of four elements.

Definition 3. A request pattern of h-out-of-k mutual exclusion, r_k, is a bag of positive integers up to k.

The h-out-of-k mutual exclusion is omitted if it is obvious.

Definition 4. A request pattern r_k is conflicting iff $\sum_{h \in r_k} h \geqslant k+1$. A conflicting request pattern r_k is critical iff $\forall h \in r_k$, $r_k - \{h\}$ is not conflicting.

For example, $r_4^1 = \{2,2,3\}$ and $r_4^2 = \{1,1,1,2\}$ are conflicting request patterns for $k=4$; r_4^1 is not critical because $\{2,3\}$ is also a conflicting request pattern, while r_4^2 is critical.

Definition 5. For a request pattern r_k, a bag $\{q \in Q_{h,k} \mid h \in r_k\}$ is called a quorum assignment for r_k. Let $QA(r_k)$ be the set of all quorum assignments for r_k.

A quorum assignment means a case where each process selects a quorum in $Q_{h,k}$ and sends a request. Because $|Q_{h,k}| \geqslant 1$, there can be multiple quorum assignments for a given request pattern.

Theorem 1 (Intersection property). *The safety property is guaranteed iff*

$$\forall \Gamma \in QA(cr_k), \quad \bigcap_{q \in \Gamma} q \neq \emptyset \tag{1}$$

is satisfied for any critical conflicting request pattern cr_k.

Note that the minimality property is exactly the same as that for the 1-coterie.

Proof of Theorem 1. First, assume that $\exists \Gamma \in QA(cr_k)$, $\bigcap_{q\in\Gamma} q = \emptyset$ is satisfied for a conflicting request pattern cr_k. Since $\bigcap_{q\in\Gamma} q = \emptyset$, no process receives every request in cr_k. Since cr_k is critical, for any $h \in cr_k$, $cr_k - \{h\}$ is not conflicting. Thus, since $\sum_{h'\in cr_k-\{h\}} h' \leqslant k$, every process sends a permission to every request. Therefore, every request in cr_k receives enough permissions to use the shared resource at the same time. Therefore, safety is not guaranteed.

Next, assume that $\forall \Gamma \in QA(cr_k)$, $\bigcap_{q\in\Gamma} q \neq \emptyset$ is satisfied for any critical conflicting request pattern cr_k. We show that the processes cannot use the shared resource at the same time for any conflicting request pattern r_k. For any conflicting request pattern r_k, there is a critical conflicting request pattern cr_k satisfying $cr_k \subseteq r_k$. Such a cr_k can be obtained as follows. If r_k is critical, set $cr_k := r_k$ and terminate the procedure. If not, there exists $h \in r_k$ satisfying $r_k - \{h\}$ is conflicting. Remove h from r_k. New r_k is also a conflicting request pattern and returns to the top of the procedure.

The above procedure terminates for every r_k since r_k is a finite bag. Thus, there is a critical conflicting request pattern cr_k satisfying $cr_k \subseteq r_k$. From the assumption, there is a process p that receives every request in cr_k. Since p can send permissions for up to k requested units at the same time, p does not send a permission to every request in cr_k. Thus, the requests in cr_k cannot all use the shared resource at the same time, meaning that the requests in $r_k(\supseteq cr_k)$ do not use the shared resource at the same time, which guarantees the safety property. □

Note that the condition in Theorem 1 becomes the intersection property for k-coterie when $cr_k = \{1, 1, \ldots, 1\}$. Thus, the intersection property for (h,k)-arbiter includes that for k-arbiter.

The following property is satisfied for critical conflicting request patterns.

Theorem 2. *Critical conflicting request pattern cr_k satisfies $\sum_{h\in cr_k} h \leqslant k + \alpha$, where $\alpha = \min_{h\in cr_k} h$.*

Proof. If a critical conflicting requesting pattern cr_k satisfies $\sum_{h\in cr_k} h \geqslant k + 1 + \alpha$, $cr_k - \{\alpha\}$ satisfies $\sum_{h\in cr_k-\{\alpha\}} h \geqslant k + 1$ and $cr_k - \{\alpha\}$ is conflicting. This implies that cr_k is not critical. □

3.2. Uniform (h,k)-arbiter

The uniform (h,k)-arbiter is defined as follows:

$$Q_{h,k} = \left\{ q \subset U \mid |q| = \left\lfloor \frac{k \cdot n}{k+h} \right\rfloor + 1 \right\}.$$

Theorem 3. *The uniform (h,k)-arbiter satisfies the conditions of the (h,k)-arbiter.*

Proof. In order to prove that the condition of Theorem 1 is satisfied, we show that there is a process that receives every request. If a process requesting h units selects $q_h \in Q_{h,k}$, $n - |q_h|$ processes do not receive the request from the process. Thus, we must show that $\forall \Gamma \in QA(cr_k)$, $\sum_{q\in\Gamma}(n - |q|) < n$ for any critical conflicting request pattern cr_k. From the definition of $Q_{h,k}$, this inequality can be written as

$$\sum_{h\in cr_k}\left(n - \left\lfloor \frac{k\cdot n}{k+h}\right\rfloor - 1\right) < n.$$

Since $\lfloor k\cdot n/(k+h)\rfloor \geqslant k\cdot n/(k+h) - (k+h-1)/(k+h)$,

$$\sum_{h\in cr_k}\left(n - \left\lfloor \frac{k\cdot n}{k+h}\right\rfloor - 1\right) \leqslant \sum_{h\in cr_k}\frac{h\cdot n - 1}{k+h}.$$

When $\min_{h\in cr_k} h = \alpha$, $\sum_{h\in cr_k} h \leqslant k+\alpha$ (from Theorem 2). Thus,

$$\sum_{h\in cr_k}\frac{h\cdot n-1}{k+h} \leqslant \sum_{h\in cr_k}\frac{h\cdot n-1}{k+\alpha} \leqslant \frac{(k+\alpha)n - |cr_k|}{k+\alpha} < n.$$

Therefore, Eq. (1) holds. Minimality holds since $|q| = |q'|$ for any two $q, q' \in Q_{h,k}$. □

The quorums in the uniform (h,k)-arbiter are no larger than those in the uniform k-arbiter. Thus, the uniform (h,k)-arbiter is superior to the uniform k-arbiter.

Assume that the number of units h in every request satisfies $h = k$. In this case, h-out-of-k mutual exclusion becomes 1-mutual exclusion, meaning that every process tries to access a unique imaginary resource consisting of k units. Thus, the quorums in $Q_{k,k}$ cannot be smaller than these in a 1-coterie. Since the uniform (h,k)-arbiter is defined so that the requesting process can use the shared resource if the number of permissions exceeds a certain threshold, i.e., its definition is exactly the same as the majority coterie for 1-mutual exclusion, the quorums in $Q_{k,k}$ cannot be smaller than those in the majority coterie. The size of the quorums in $Q_{k,k}$ is $\lfloor n/2\rfloor + 1$, which is exactly the same as that of those in the majority coterie. Thus the uniform (h,k)-arbiter is a generalization of the majority coterie for 1-mutual exclusion.

3.3. $(k+1)$-*Cube* (h,k)-*arbiter*

The $(k+1)$-cube (h,k)-arbiter is defined as follows. First, assume that $n = a^{k+1}$ for integer a. Each process corresponds to a node on a $(k+1)$-dimensional hypercube, which can be represented by $\{(x_1,x_2,\ldots,x_{k+1}) \mid 0 \leqslant x_i \leqslant a-1,\ 1 \leqslant i \leqslant k+1\}$.

Let

$$z_h = \left\lfloor \frac{1+k}{1+k/h}\right\rfloor.$$

$$Q_{h,k} = \{q_{h,k}^{b_1,\ldots,b_{k+1}} \mid 0 \leqslant b_i \leqslant a-1,\ 1 \leqslant i \leqslant k+1\},$$

where

$$q_{h,k}^{b_1,\ldots,b_{k+1}} = \bigcup_{0 \leqslant j \leqslant k+1-z_h} \{(x_1, x_2, \ldots, x_{k+1}) \mid x_{i+j} = b_{i+j}, 1 \leqslant i \leqslant z_h\}.$$

$q_{h,k}^{b_1,\ldots,b_{k+1}}$ consists of $(k+2-z_h)$ subcubes whose dimensions are $(k+1-z_h)$. Therefore, the size of each quorum in $Q_{h,k}$, $|q_{h,k}^{b_1,\ldots,b_{k+1}}|$, is no larger than $(k+2-z_h)a^{k+1-z_h}$.

Theorem 4. *The $(k+1)$-cube (h,k)-arbiter satisfies the conditions of the (h,k)-arbiter.*

Proof. In order to prove that the condition of Theorem 1 is satisfied, we show that there is a process that receives every request. Let $\{q_{h_1,k}^{b_1^1,b_2^1,\ldots,b_{k+1}^1}, q_{h_2,k}^{b_1^2,b_2^2,\ldots,b_{k+1}^2}, \ldots, q_{h_\ell,k}^{b_1^\ell,b_2^\ell,\ldots,b_{k+1}^\ell}\}$ be a quorum assignment for a critical conflicting request pattern. From Theorem 2, $\sum_{i=1}^{\ell} h_i \leqslant k+\alpha$, where $\alpha = \min_{1\leqslant i\leqslant \ell} h_i$. Let $y_0 = 0$ and $y_i = \sum_{j=1}^{i} z_{h_j}$ $(1 \leqslant i \leqslant \ell)$. From the definition of z_h, $y_\ell = \sum_{j=1}^{\ell} \lfloor (1+k)/(1+k/h_j) \rfloor \leqslant \sum_{j=1}^{\ell} (1+k)/(1+k/h_j) = (1+k)\sum_{j=1}^{\ell} h_j/(h_j+k) \leqslant (1+k)\sum_{j=1}^{\ell} h_j/(\alpha+k) \leqslant (1+k)(k+\alpha)/(\alpha+k) = (1+k)$.

Now consider a process $v = (b_1^1, b_2^1, \ldots, b_{y_1}^1, b_{y_1+1}^2, b_{y_1+2}^2, \ldots, b_{y_2}^2, \ldots, b_{y_{i-1}+1}^i, b_{y_{i-1}+2}^i, \ldots, b_{y_i}^i, \ldots, b_{y_{\ell-1}+1}^\ell, b_{y_{\ell-1}+2}^\ell, \ldots, b_{y_\ell}^\ell, c_{y_{\ell+1}}, c_{y_{\ell+2}}, \ldots, c_{k+1})$, where $c_j (y_\ell + 1 \leqslant j \leqslant k+1)$ is an arbitrary number. If $y_\ell = k+1$, c_j does not exist.

$v \in q_{h_i,k}^{b_1^i,b_2^i,\ldots,b_{k+1}^i}$ is satisfied for every i $(1 \leqslant i \leqslant \ell)$, since subcube $\{(x_1, x_2, \ldots, x_{k+1}) \mid x_{y_{i-1}+j} = b_{y_{i-1}+j}^i, 1 \leqslant j \leqslant z_h\}$ is contained in $q_{h_i,k}^{b_1^i,b_2^i,\ldots,b_{k+1}^i}$.

Therefore, Eq. (1) holds. Minimality holds since $|q| = |q'|$ for any two $q, q' \in Q_{h,k}$. □

If $n \neq a^{k+1}$, we can modify the above quorum definition as follows. Assume that $(a-1)^{k+1} < n < a^{k+1}$. The $(k+1)$-tuple $(b_1, b_2, \ldots, b_i, \ldots, b_{k+1})$ $(0 \leqslant b_i \leqslant a-1, 1 \leqslant i \leqslant k+1)$ represents process $\sum_{i=1}^{k+1} b_i \cdot a^{i-1} \pmod n$. Construct the $(k+1)$-cube (h,k)-arbiter for these a^{k+1} tuples. Based on this relation between $(k+1)$-tuples and processes, it is obvious that the above quorum definition satisfies the intersection property. Minimality is satisfied by removing q' if $q \subset q'$ for some $q, q' \in Q_{h,k}$.

The quorums in the $(k+1)$-cube (h,k)-arbiter are not larger than those in the $(k+1)$-cube k-arbiter. Thus, the $(k+1)$-cube (h,k)-arbiter is superior to the $(k+1)$-cube k-arbiter.

Assume that $h = k$ and consider 1-mutual exclusion. The size of the quorums in $Q_{k,k}$ is $(\lceil (k+1)/2 \rceil + 1)a^{\lceil (k+1)/2 \rceil}$, which is close to the minimum value $O(\sqrt{n})$ of the square grid coterie. In addition, if $k = 1$, $Q_{1,1}$ is exactly the same as the square grid coterie. Thus, the $(k+1)$-cube (h,k)-arbiter is a generalization of the square grid coterie for 1-mutual exclusion.

4. Distributed mutual exclusion algorithm

4.1. Algorithm description

In this section we present a distributed algorithm for h-out-of-k mutual exclusion that uses an (h,k)-arbiter. The intersection property guarantees safety. The distributed

algorithm used to achieve liveness is the same as that for the k-arbiter. Its outline is as follows.

Each process maintains a Lamport's logical clock [11]. Let the value of process p's current clock be c_p. When p sends message m, the current value of c_p is piggybacked on m. Let c_m be the value of the clock piggybacked on m when p receives message m. p updates $c_p := \max(c_p, c_m) + 1$. This clock update mechanism is omitted in the algorithm shown in Fig. 1.

Each request is a tuple (h, p, c), where h is the number of units, p is the requesting process, and c is the value of the logical clock when p initiated the request. A priority is assigned to every request. The priority of request (h, p, c) is higher than that of (h', p', c') if $c < c'$ or $(c = c'$ and $p < p')$. A total order is thus given to the requests.

When p initiates request (h, p, c), it selects a quorum $q \in Q_{h,k}$ and sends a "request (h, p, c)" message to every process in q. If p receives an "OK" response from every process in q, it can use h units of the shared resource. When p finishes using the units, it sends a "release" message to every process in q. Note that a "cancel" message might be received during the wait for "OK"s. The procedure for this case is shown below.

Each process initially has k permissions. Let x_p be the permissions p currently has. Each process also maintains a priority queue of requests. It tracks the status of each request: "wait", "ok", or "cancel". The priority of requests is defined as described above. Initially, the queue is empty. When p receives "request (h', p', c')", it inserts the request in its priority queue and sets the status to "wait".

If the request (h', p', c') satisfies the condition that the total number of units requested from the higher-priority requests in the queue is no more than $k - h'$, and $x_p \geqslant h'$, p sends an "OK" message to p', changes the request's status to "ok", and executes $x_p := x_p - h'$.

There may be a request (h'', p'', c'') in the priority queue whose status is "ok" and the total number of units requested from higher-priority requests is now more than $k - h''$ as a result of insertion of a new request in the priority queue. Note that multiple requests might satisfy this condition as a result of inserting one request. The "OK" to these requests must be cancelled, otherwise a deadlock might occur. Thus, p sends a "cancel" message to p'' to cancel the "OK" and changes the status of the request to "cancel".

When p'' receives the "cancel" message from p, it sends a "cancelled" message back to p if it has not yet received "OK" from every process in q. It then waits for another "OK" message from p.

When p receives a "cancelled" message from p'', it changes the status of the request (h'', p'', c'') to wait. p executes $x_p := x_p + h''$ and tries to send an "OK" message to the higher-priority requests based on the above condition for sending an "OK" message.

When p receives a "release" message from p'', it removes the request (h'', p'', c'') from its priority queue, executes $x_p := x_p + h''$, and tries to send an "OK" message to the other requests based on the above condition for sending an "OK" message. The algorithm is shown in detail in Fig. 1.

4.2. Correctness of algorithm

The proof of the correctness of this algorithm is shown.

```
program MutualExclusion(p:process);
var c = 0 : integer; /* logical clock. */
  h = 0 : integer; /* units. */
  In = false : boolean;
  Q, K = φ : set of process;
  x = k : integer; /* current tokens. */
  Que = null : priority queue;
  ql = 0 : integer; /* Que's length. */

When p wants to use h units:
begin
    Q :=(one quorum in Q_{h,k});
    K := φ;
    send "request(h, p, c)" to all r ∈ Q;
end; /* end of request initiation. */

At arrival of "OK" from p':
begin
  K := K ⋃ {p'};
  if K ≠ Q then return; /* need more */
  In := true;
.... /* in the critical section */
  In := false;
  send "release" to all r ∈ Q
end;

At arrival of "request(h', p', c')" from p':
begin
  Insert it to Que by priority (p', c');
    /* Suppose this request be j-th. */
  Que[j].c := c';
  Que[j].p := p';
  Que[j].h := h';
  Que[j].st :='wait';
  ql := ql + 1;
  if (∑_{i=1}^{j} Que[i].h ≤ k) and (x ≥ h') then
    begin /* permission */
      send "OK" to p';
      Que[j].st :='ok';
      x := x − h'
    end;
    /* cancel "OK" to lower priorities. */
  i :=maximum integer (≤ ql)
    that satisfies ∑_{j=1}^{i} Que[j].h ≤ k;
  if (i = ql) then return;
  for j := i + 1 to ql do
    if (Que[i].st ='ok') then begin
      send "cancel" to Que[i].p;
      Que[i].st :='cancel'
    end
end;

At arrival of "cancel" from p':
begin
  if In = false then begin
    send "cancelled" to p';
    K := K − {p'}
  end
end;

At arrival of "release" from p':
begin
  /* Suppose Que[j].p = p' */ ;
  x := x + Que[j].h;
  Delete entry Que[j] from Que;
  ql := ql − 1;
  NewChance
end;

At arrival of "cancelled" from p':
begin
  /* Suppose Que[j].p = p' */ ;
  Que[j].st :='wait';
  x := x + Que[j].h;
  NewChance
end;

procedure NewChance; /* tokens released. */
begin
  for i := 1 to ql do begin
    if (Que[i].st ='wait') then begin
      if (x < Que[i].h) then return;
      send "OK" to Que[i].p;
      Que[i].st :='ok';
      x := x − Que[i].h
    end
  end
end;
```

Fig. 1. Distributed h-out-of-k mutual exclusion algorithm.

Theorem 5. *No deadlock occurs in the above h-out-of-k mutual exclusion algorithm.*

Proof. Let us consider the wait-for graph G. Each node in G represents a process. A directed edge (p, p') exists in G iff both the following conditions hold for a process r:

- p has sent "request" to r and is waiting for "OK" from r;
- r has sent "OK" to p' and it is not cancelled by "cancel".

First, we show that when a deadlock occurs, there is a permanent directed cycle in G. Without loss of generality, we assume that after a deadlock occurs, all processes that can enter the critical section have exited from the critical section and no new request occurs. In this case, no node without an edge in G is involved in the deadlock. Thus G has at least one edge. If there is no directed cycle in G, then there is a process p with an incoming edge and no outgoing edge. Since p has no outgoing edge, every process that receives "request" from p has not sent "OK" to any other process. Thus p receives "OK" and p can enter the critical section. This contradicts the assumption that a deadlock occurs.

Next, we show that there is no permanent directed cycle in wait-for graph G during the execution of the algorithm. Let us assume that there is a permanent directed cycle $p_0, p_1, \ldots, p_{m-1}, p_0$ in G, that is, there are directed edges $(p_i, p_{i+1 \,(\mathrm{mod}\, m)})$ $(0 \leqslant i \leqslant m-1)$. Let r_i $(0 \leqslant i \leqslant m-1)$ be the process that has sent "OK" to $p_{i+1 \,(\mathrm{mod}\, m)}$ and received "request" from p_i.

The sum of the units of the requests whose priority is higher than or equal to that of p_i is more than k at r_i. Otherwise, r_i would have sent "cancel" to lower priority requests, waited until enough tokens were available, and sent "OK" to p_i. Since the "OK" to $p_{i+1 \,(\mathrm{mod}\, m)}$ is not cancelled by r_i, the sum of the units of the requests whose priority is higher than or equal to that of $p_{i+1 \,(\mathrm{mod}\, m)}$ is no more than k at r_i. Thus, the priority of p_i is lower than that of $p_{i+1 \,(\mathrm{mod}\, m)}$. Since there is a directed cycle, p_0's priority would have to be lower than p_0's priority. This contradicts the definition of the priority. Therefore, there is no permanent directed cycle in G and no deadlock occurs. □

Theorem 6. *No starvation occurs in the above h-out-of-k mutual exclusion algorithm.*

Proof. Assume that there is a requesting process p that can never enter the critical section. The request has a pair (p, c_p) as its priority. Let q be the (h,k)-arbiter selected by p. Since p can never enter the critical section, at least one process $r \in q$ receives an infinite number of requests whose priority is higher than p, after receipt of p's request. Note that after receiving the request from p, the value of r's logical clock c_r satisfies $c_r > c_p$. Thus, when some requesting process u receives "OK" from r, u's logical clock value is greater than c_p and u cannot send any more requests whose priority is higher than (p, c_p). Thus, there cannot be an infinite number of requests with a priority higher than (p, c_p). Therefore, p can enter the critical section. □

4.3. Message complexity

Let us enumerate the number of messages sent per request. The best case is that there is no conflict.

(1) p sends "request" to every process in some $q \in Q_{h,k}$.
(2) Every process in q sends "OK" to p.
(3) p enters and exits the critical section. p sends "release" to every process in q.

Thus, the message complexity is $3|q_{h,k}|$ per request, where $|q_{h,k}|$ is the size of the largest quorum in $Q_{h,k}$.

Next consider the worst case. The worst case is that there is conflict and lower priority requests must be cancelled.

(1) p sends "request" to every process in some $q \in Q_{h,k}$.
(2) Each process $u \in q$ has sent k "OK"s to other requests whose requesting units are 1. The request from p has a priority higher than these requests. Thus, u sends "cancel" to h requesting processes to cancel the "OK"s.
(3) Each process replies "cancelled" to u. Thus, u sends "OK" to p.
(4) p enters and exits the critical section. p sends "release" to every process in q.
(5) u receives "release" and sends "OK" again to the cancelled processes.

Note that after p receives "OK", it may be cancelled by another process r's request. Messages for this procedure are counted in the procedure for r. Thus, the message complexity of this case is $(3h+3)|q_{h,k}|$ per request.

5. Concluding remarks

We have defined two (h,k)-arbiters for h-out-of-k mutual exclusion: a uniform (h,k)-arbiter and a $(k+1)$-cube (h,k)-arbiter. The quorums in each (h,k)-arbiter are not larger than those in the corresponding k-arbiters; consequently using the (h,k)-arbiters is better than using the k-arbiters.

An outstanding problem involves obtaining an (h,k)-arbiter with minimum size quorums, since the (h,k)-arbiters we have developed are not optimal. Another problem is obtaining a non-dominated (h,k)-arbiter, where the non-domination of (h,k)-arbiter can be defined in exactly the same way as for the k-arbiter [14].

Acknowledgements

We thank Dr. Hirofumi Katsuno of NTT (currently Professor of Tokyo Denki University) and Mr. Yoshifumi Ooyama of NTT for their encouragement and suggestions. We also thank the anonymous referees for their useful comments on ways to improve this paper.

References

[1] D. Agrawal, Ö. Eğecioğlu, A.E. Abbadi, Analysis of quorum-based protocols for distributed $(k+1)$-exclusion, Proc. of First COCOON, Lecture Notes in Computer Science, Vol. 959, Springer, Berlin, 1995, pp. 161–170.

[2] R. Baldoni, B. Ciciani, Distributed algorithms for multiple entries to a critical section with priority, Inform. Process. Lett. 50 (1994) 165–172.
[3] S. Bulgannawar, N.H. Vaidya, A distributed K-mutual exclusion algorithm, Proc. 15th Internat. Symp. on Distributed Computing Systems, Vancouver, Canada, 1995, pp. 153–160.
[4] H. Garcia-Molina, D. Barbara, How to assign votes in a distributed system, J. ACM 32 (4) (1985) 841–860.
[5] D.K. Gifford, Weighted voting for replicated data, Proc. Seventh ACM Symp. on Operating Systems Principles (SOSP), California, USA, 1979, pp. 150–162.
[6] J.-R. Jiang, S.-T. Huang, Y.-C. Kuo, Cohorts structures for fault-tolerant k entries to a critical section, IEEE Trans. Comput. 48 (2) (1997) 222–228.
[7] H. Kakugawa, S. Fujita, M. Yamashita, T. Ae, Availability of k-coterie, IEEE Trans. Comput. 42 (5) (1993) 553–558.
[8] H. Kakugawa, S. Fujita, M. Yamashita, T. Ae, A distributed k-mutual exclusion algorithm using k-coterie, Inform. Process. Lett. 49 (1994) 213–238.
[9] Y.-C. Kuo, S.-T. Huang, A simple scheme to construct k-coteries with $O(\sqrt{N})$ uniform quorum sizes, Inform. Process. Lett. 59 (1996) 31–36.
[10] Y.-C. Kuo, S.-T. Huang, A geometric approach for constructing coteries and k-coteries, IEEE Trans. Parallel Distributed Systems 8 (4) (1997) 402–411.
[11] L. Lamport, Time, clocks, and the ordering of events in a distributed system, Comm. ACM 21 (7) (1978) 558–565.
[12] M. Maekawa, A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems, ACM Trans. Comput. Systems 3 (2) (1985) 145–159.
[13] K. Makki, P. Banta, K. Been, N. Pissinou, E.K. Park, A token based distributed k mutual exclusion algorithm, Proc. Fourth IEEE Symp. on Parallel and Distributed Processing, Arlington, USA, 1992, pp. 408–411.
[14] Y. Manabe, R. Baldoni, M. Raynal, S. Aoyagi, k-Arbiter: a safe and general scheme for h-out-of-k mutual exclusion, Theoret. Comput. Sci. 193 (1–2) (1998) 97–112.
[15] M. Naimi, Distributed algorithm for k-entries to critical section based on the directed graphs, ACM Oper. Systems Rev. 27 (4) (1993) 67–75.
[16] M.L. Nilsen, M. Mizuno, Nondominated k-coteries for multiple mutual exclusion, Inform. Process. Lett. 50 (1994) 247–252 (Erratum 60 (1996) 319).
[17] K. Raymond, A distributed algorithm for multiple entries to a critical section, Inform. Process. Lett. 30 (4) (1989) 189–193.
[18] M. Raynal, A distributed solution for the k-out-of-m resources allocation problem, Proc. First Conf. on Computing and Information, Lecture Notes in Computer Sciences, Vol. 497, Springer, Berlin, 1991, pp. 599–609.
[19] R.D. Schlichting, F.B. Schneider, Fail-stop processors: an approach to designing fault-tolerant computing systems, ACM Trans. Comput. Systems 1 (3) (1983) 222–238.
[20] M. Singhal, A taxonomy of distributed mutual exclusion, J. Parallel Distributed Comput. 18 (1) (1993) 94–101.
[21] R.K. Srimani, S.R. Das (Eds.), Distributed Mutual Exclusion Algorithms, IEEE Computer Society Press, Silver Spring, MD, 1992.
[22] P.K. Srimani, R.L.N. Reddy, Another distributed algorithm for multiple entries to a critical section, Inform. Process. Lett. 41 (1) (1992) 51–57.
[23] R.H. Thomas, A majority consensus approach to concurrency control for multiple copy databases, ACM TODS 4 (2) (1979) 180–209.
[24] S.M. Yuan, H.K. Chang, Comments on: availability of k-coterie, IEEE Trans. Comput. 43 (12) (1994) 1457.