A Fault-Tolerant *h*-out of-*k* Mutual Exclusion Algorithm Using Cohorts Coteries for Distributed Systems

Jehn-Ruey Jiang

Department of Computer Science and Information Engineering
National Central University, Jhongli 320, Taiwan
jrjiang@csie.ncu.edu.tw

Abstract. In this paper, we propose a distributed algorithm for solving the *h*-out of-*k* mutual exclusion problem with the aid of a specific *k*-coterie — cohorts coterie. The proposed algorithm is resilient to node and/or link failures, and has constant message cost in the best case. Furthermore, it is a candidate to achieve the highest availability among all the algorithms using *k*-coteries. We analyze the algorithm and compare it with other related ones.

1 Introduction

A distributed system consists of interconnected, autonomous nodes which communicate with each other by passing messages. A node in the system may need to enter the *critical section* (*CS*) occasionally to access a shared resource, such as a shared file or a shared table, etc. The problem of controlling the nodes so that the shared resource is accessed by at most one node at a time, is called the *mutual exclusion* problem. If there are *k*, $k \geq 1$, identical copies of shared resources, such as a *k*-user software license, then there can be at most *k* nodes accessing the resources at a time. This raises the *k-mutual exclusion* problem. On some occasions, a node may require to access *h* ($1 \leq h \leq k$) copies out of the *k* shared resources at a time; for example, a node may need *h* disks from a pool of *k* disks to proceed efficiently. How to control the nodes to acquire the desired number of resources with the total number of resources accessed concurrently not exceeding *k* is called the *h-out of k-mutual exclusion* problem or the *h-out of-k resource allocation* problem [10].

There are at least four distributed *h*-out of-*k* mutual exclusion algorithms proposed in the literature. Raynal proposed the first algorithm in [10] and then three algorithms using *k*-arbiters, (*h*, *k*)-arbiters, and *k*-coteries are proposed in [2], [9], and [5], respectively. Among the four algorithms, only Jiang's algorithm using *k*-coteries is fault-tolerant. It can tolerate node and/or network link failures even when the failures lead to network partitioning. Furthermore, it is shown in [5] to have lower message cost than others. The basic idea of Jiang's algorithm is simple: a node should collect enough permissions from some set of nodes to enter CS. However, there raise some problems when a node fails to collect enough permissions repeatedly.

In this paper, we proposed another *h*-out of-*k* mutual exclusion algorithm using a specific *k*-coterie — *cohorts coterie* to eliminate the problems of Jiang's algorithm. A cohorts coterie [4] is a *k*-coterie [3], which is a collection of sets (called *quorums*) satisfying the *intersection*, the *non-intersection* and the *minimality* properties. As we will show, the proposed algorithm has constant message cost in the best case and is a candidate to achieve the highest *availability*, the probability that a node can gather enough permissions to enter CS in an error-prone environment, among all the algorithms using *k*-coteries.

The rest of this paper is organized as follows. In Section 2, we introduce some related work. In Section 3, we propose the *h*-out of-*k* mutual exclusion algorithm using cohorts coteries. In Section 4, we analyze the proposed algorithm and compare it with related ones. And finally, we give some concluding remarks in Section 5.

2 Related Work

In [10], Raynal proposed the first distributed *h*-out of-*k* mutual exclusion algorithm. Raynal's algorithm is extended from Ricart and Agrawala's algorithm [11]. It demands a node *u* to send request messages to all other nodes and wait for replies to estimate the number of unoccupied resources. A node *v* replies that there are *k* unoccupied resources if it is neither using nor requesting shared resources, or if it has lower priority than the requester (in terms of logical clock [7] order). On the other hand, node *v* replies that there are *k*–*h* unoccupied resources if *v* is using *h* resources or if *v* is requesting *h* resources with higher priority. For such a case, node *v* should later reply again that there are *h* resources released after it leaves CS. From all the replies, if node *u* finds that the estimated number of unoccupied resources is larger than the number of requested resources, it can enter CS. Raynal's algorithm has message complexity between 2(*n*–1) and 3(*n*–1), where *n* is the number of nodes. It is not fault-tolerant since a node cannot gather replies from all other nodes if there is any failing node.

In [1], Baldoni proposed the concept of *arbiter sets* to solve the *h*-out of-*k* mutual exclusion problem. Every node is associated with an arbiter set (request set), and any *k* arbiter sets should have at least one common member. A node should send request messages with parameter *h* (*h*≤*k*) to all members of its arbiter set to gather permissions to access *h* resources. Every node keeps the number of unoccupied resources, which is initially *k* and is decreased by *h* after granting a request for accessing *h* resources. The total number of resources concurrently being accessed is guaranteed to be no more than *k* because the common member of any *k*+1 arbiter sets can serve as the arbiter, which grants its permission only when the number of unoccupied resources is no less than the number of requested resources. The algorithm using arbiter sets has the message complexity $O(q)$, where *q* is the size of arbiter sets. Baldoni proved that arbiter sets have the size lower bound of $O(n^{k/k+1})$ if all arbiter sets have the same size and every node appears in the same number of arbiter sets.

The concept of the arbiter set was further formalized as the *k*-arbiter by Baldoni et al. in [2]. A *k-arbiter* is a collection of minimal arbiter sets (called *quorums*) where any *k*+1 quorums have at least one common member. Two *k*-arbiters were proposed in [2]:

(k+1)-cube and uniform k-arbiters, with quorum sizes $(k+1)\cdot n^{k/(k+1)}$ and $\lfloor k \cdot n/(k+1) \rfloor + 1$, repectively. In [9], Manabe and Tajima further generalized k-arbiters with (h,k)-arbiters and proposed (k+1)-cube and uniform (h,k)-arbiters with quorum size $(k+2-h)\cdot n^{(k+1-h)/(k+1)}$ and $\lfloor k \cdot n/(k+h) \rfloor + 1$, respectively.

The h-out of-k mutual exclusion algorithms [1, 2, 9] using arbiter sets are not fault-tolerant in the sense that a node just selects a quorum and waits for all the members of the quorum to reply. If any member of the selected quorum fails, a node may fail to gather permissions to enter CS. In [5], Jiang proposed a fault-tolerant distributed h-out of-k mutual exclusion algorithm using k-coterie. A *k-coterie* [3] is a collection of sets (called *quorums*) satisfying the following properties:

1. *Intersection Property*: There are at most k pairwise disjoint quorums.
2. *Non-intersection Property*: For any h (< k) pairwise disjoint quorums $Q_1,...,Q_h$, there exists a quorum Q_{h+1} such that $Q_1,...,Q_{h+1}$ are pairwise disjoint.
3. *Minimality Property*: Any quorum is not a super set of another quorum.

In Jiang's algorithm, a node u requesting to access h resources should randomly select h pairwise disjoint quorums and send request messages to the members of the h quorums. On receiving a request message, a node v grants its permission by replying a grant message. Node u can enter CS after it gathers permissions from members of h pairwise disjoint quorums. The correctness of h-out of-k mutual exclusion is guaranteed since every node grants its permission to only one at a time and there are at most k pairwise disjoint quorums. Jiang's algorithm is fault-tolerant in the sense that a node can reselect h pairwise disjoint quorums for sending incremental request messages when the node does not gather enough permissions after a timeout period. However, Jiang's algorithm has the following problems: First, it does not explicitly specify how to efficiently select and reselect h pairwise disjoint quorums. Second, it is difficult to determine the timeout value.

3 The proposed algorithm

In this section, we propose an h-out of-k mutual exclusion algorithm using a specific k-coterie — cohorts coterie [4]. The proposed algorithm does not use timeout mechanism and does not require a node to reselect h pairwise disjoint quorums. As we will show, the proposed algorithm has constant message complexity in the best case and is a candidate to achieve the highest availability among those using k-coteries.

Before presenting the proposed algorithm, we first introduce the cohorts k-coterie [6], which is constructed with the aid of *cohorts structures*. A *cohorts structure* $Coh(k, m) \equiv (C_1,...,C_m)$, $m \geq k$, is a list of sets, where each set C_i is called a *cohort*. The cohorts structure $Coh(k, m)$ should observe the following three properties:

P1. $|C_1| = k$.
P2. $\forall i$: $1 < i \leq m : |C_i| > 2k-2$, for $k>1$ ($|C_i| > 1$, for $k=1$).

P3. $\forall i, j$: $1 \le i, j \le m, i \ne j$: $C_i \cap C_j = \varnothing$.

To sum up, a cohorts structure *Coh*(k, m) has m pairwise disjoint cohorts with the first cohort having k members and the other cohorts having more than $2k-2$ members (or more than one member when k=1). For example, ({1, 2}, {3, 4, 5}, {6, 7, 8, 9, 10}) is *Coh*(2,3) since it has three pairwise disjoint cohorts with the first cohort and the other cohorts having 2 (=k) and more than 2 (=$2k-2$) members, respectively.

A set Q is said to be a *quorum under Coh(k, m)* if some cohort C_i in *Coh*(k, m) is Q's *primary cohort*, and each cohort C_j, $j > i$, is Q's *supporting cohort*, where a cohort C is Q's primary cohort if $|Q \cap C| = |C| - (k-1)$ (i.e., Q contains exactly all except $k-1$ members of C), and a cohort C is Q's supporting cohort if $|Q \cap C| = 1$ (i.e., Q contains exactly one member of C).

The family of all quorums under *Coh*(k, m) is called a *cohorts coterie*, which has been shown to be a k-coterie in [4]. For example, the following sets are quorums under *Coh*(2, 2)≡({1, 2}, {3, 4, 5}): Q_1={3, 4}, Q_2={3, 5}, Q_3={4, 5}, Q_4={1, 3}, Q_5={1, 4}, Q_6={1, 5}, Q_7={2, 3}, Q_8={2, 4} and Q_9={2, 5}. Quorums Q_1,...,Q_3 take {3, 4, 5} as their primary cohort and no supporting cohort is needed, and quorums Q_4,...,Q_9 take {1, 2} as their primary cohort and {3, 4, 5} as their supporting cohort. It is easy to check that these nine sets constitute a 2-coterie.

In [6], the cohorts coterie is shown to be *nondominated* (*ND*). Let $\mathcal{C}$ and $\mathcal{D}$ be two distinct k-coteries. $\mathcal{C}$ is said to *dominate* $\mathcal{D}$ if and only if every quorum in $\mathcal{D}$ is a super set of some quorum in $\mathcal{C}$ (i.e.,$\forall Q$, $\exists Q'$: $Q \in \mathcal{D}$, $Q' \in \mathcal{C}$: $Q' \subseteq Q$). Obviously, the dominating one ($\mathcal{C}$) has more chances than the dominated one ($\mathcal{D}$) to have *available quorums* in an error-prone environment, where a quorum is said to be *available* if all of its members (nodes) are *up*. Since an available quorum implies an available entry to CS, we should always concentrate on ND k-coteries that no other k-coterie can dominate. The algorithm using ND k-coteries, for example the proposed algorithm, is a candidate to achieve the highest availability.

The core of the proposed algorithm is a permission gathering procedure, which is named *Get_Quorum* shown in Figure 1. For a distributed system with n nodes organized as a *cohorts structure Coh*(k, m)≡(C_1,...,C_m), a node requesting h out of k resources should invoke *Get_Quorum*(h, k, (C_1,...,C_m)). The node can access h resources after *Get_Quorum* returns.

The function *Probe*(C_i, g) evoked in *Get_Quorum* performs the task of probing all the nodes in set C_i for their permissions. It returns a set of nodes of C_i that reply grant messages for the following three cases. (It will not return if none of the cases stands.)

Case 1: If i>1 and there are more than $|C_i|-(k-1)+(g-1)$ nodes replying grant messages, the returning set will be a set of $|C_i|-(k-1)+(g-1)$ replying nodes.

Case 2: If i>1 and there are more than g but less than $|C_i|-(k-1)+(g-1)$ nodes replying grant messages, the returning set will be a set of g replying nodes. (Note that $|C_i|-(k-1)+(g-1)>g$ because $|C_i| > 2k-2$ for k>1, or $|C_i| > 1$ for k=1.)

Case 3: If i=1 and there are more than g nodes replying grant messages, the returning set will be a set of g replying nodes. (Note that because $|C_1|=k$, only one node can make C_1 the primary cohort of a quorum. Thus, g replying nodes can make C_1 be the primary cohorts of g quorums.)

```
Function Get_Quorum( h, k: Integer; (C1,...,Cm): Cohorts Structure):Set;
  Var R, S: Set;
  Var g: Integer;
   g = h;    //g: Storing the number of primary cohorts needed
   R = ∅;  //R: The set of replying nodes that will be returned
   For (i =m,...,2 ) Do
     S=Probe(Ci, g);
     If |S| = |Ci|–(k–1)+(g–1)
     Then {R=R∪S; g=g–1; If g=0 Then Return R;}
     /*Ci can be the primary cohort of one quorum,
       and be the supporting cohorts of g–1 quorums          */
     Else If |S|=g Then R=R∪S;
     /*Ci can only be the supporting cohorts of g quorums */
   EndFor
   S=Probe(C1, g);
   Return R∪S;   //C1 is the primary cohort of g quorums
End Get_Quorum
```

Figure 1. The permission gathering procedure – *Get_Quorum*

The procedure *Get_Quorum* uses no timeout mechanism and can return a set of nodes of h pairwise disjoint quorums efficiently. It is clear that no two nodes can simultaneously gather permissions from h_1 and h_2 pairwise disjoint quorums by invoking *Get_Quorum* if $h_1+h_2>k$. Thus, the proposed algorithm guarantees the safety property of h-out of-k mutual exclusion that there are no more than k resources being accessed concurrently. To ensure the liveness (i.e., deadlock and starvation-free) property, we could rely on the well known conflict resolution mechanism of Meakawa's algorithm [8]. However, we omit the details for simplicity.

4 Analysis and Comparison

The reader can check that if a node calls the function *Get_Quorum* when there is no failing node and no request conflict, then *Get_Quorum* will return a set of the union of h pairwise disjoint quorums which take $C_m,\ldots,C_{m-h-1}$ respectively as their primary cohorts, with C_m being the supporting cohorts of h–1 quorums,…, and C_{m-h} being the supporting cohort of one quorum. For such a case, the node has to send request message to $c \cdot h$ nodes if we assume all cohorts (including $C_m,\ldots,C_{m-h-1}$) are of the same size c, $c > 2k$–2. However, if there are failures and/or request conflicts, then some of the h quorums may take $C_{m-h-2},\ldots,C_1$ as their primary cohorts. In an extreme case, some quorums may take C_1 as their primary cohorts. In such a case, the node has to send request messages to all the n system nodes.

Like other quorum-based algorithms [1, 2, 5, 9], the proposed algorithm is a Meakawa-type algorithm [8], which relies Lamport's logical clock concept [7] and five

types of messages, namely *request*, *grant*, *release*, *inquire* and *relinquish* messages, to avoid deadlock and starvation. In the best case, a node u needs $3c \cdot h$ messages to access h, $h \le k$, resources. The best case occurs when u sends request messages to all members of $C_m,\ldots,C_{m-h-1}$, receives grant messages from all members of $C_m,\ldots,C_{m-h-1}$, and at last sends release message to all members of $C_m,\ldots,C_{m-h-1}$ on leaving CS. In the worst case, the message complexity is $6n$. It occurs when u sends request messages to each node u, u sends inquire message to some node w, w sends relinquish message to u, u sends grant message to v, v sends release message to u (after v leaves CS), and at last u sends grant message to w. The worst case message complexity can be reduced by the following mechanism. We can set a probability p for a node to decide whether or not to further probe nodes in cohorts $C_{m-h-2},\ldots,C_1$ after it has probed nodes in $C_m,\ldots,C_{m-h-1}$. The probability p makes the worst case message complexity to be $6f \cdot n$, where f is a constant between 0 and 1. With p, we can trade fault-tolerance for message-efficiency. The reader can check that larger p will lead to higher message complexity and higher degree of fault-tolerance.

As shown in [10], Raynal's algorithm has message complexity between $2(n-1)$ and $3(n-1)$. The message complexities of the algorithms using k-arbiters and (h,k)-arbiters have been analyzed in [5]. Both the algorithms have the message complexity $3q$ in the best case and $(3h+3)q$ in the worst case, where q is the quorum size of the k-arbiter or the (h,k)-arbiter. As shown in [5], Jiang's algorithm has message complexity $3h \cdot q$ in the best case and $6e \cdot n$ in the worst case, where q is the quorum size of the k-coterie and $0 < e \le 1$. Table 1 shows the comparison of Raynal's algorithm [10], the algorithms using k-arbiters [2] and (h,k)-arbiters [9], Jiang's algorithm [5], and the proposed algorithm.

Table 1. The comparison of various distributed h-out of-k mutual exclusion algorithms

Algorithm	Message complexity	Quorum reselection	Timeout mechanism	Fault-Tolerance
Raynal's algorithm [10]	between $2(n-1)$ and $3(n-1)$	no	no	no
The algorithm using k-arbiters [2]	between $3q$ to $(3h+3)q$, where $q=(k+1)\cdot n^{k/(k+1)}$ for the $(k+1)$-cube arbiter and $q=\lfloor k \cdot n/(k+1) \rfloor + 1$ for the uniform k-arbiter	no	no	no
The algorithm using (h,k)-arbiters [9]	between $3q$ to $(3h+3)q$, where $q=(k+2-h)\cdot n^{(k+1-h)/(k+1)}$ for the $(k+1)$-cube (h,k)-arbiter and $q=\lfloor k \cdot n/(k+h) \rfloor + 1$ for the uniform (h,k)-arbiter	no	no	no
Jiang's algorithm [5]	between $3h \cdot q$ and $6e \cdot n$, where q is the quorum size of the k-coterie used, and $0 < e \le 1$	yes	yes	yes
The proposed algorithm	between $3c \cdot h$ and $6f \cdot n$, where $c > 2k-2$ and $0 < f \le 1$	no	no	yes (maybe of the highest availability)

*n stands for the number of nodes, and h stands for the number of requested resources.

5 Conclusion

In this paper, we have proposed a distributed *h*-out of-*k* mutual exclusion algorithm using a specific *k*-coterie — cohorts coterie. The proposed algorithm becomes a *k*-mutual exclusion algorithm for $k>h=1$, and becomes a mutual exclusion algorithm for $k=h=1$. It is resilient to node and/or link failures and has constant message cost in the best case. Furthermore, it is a candidate to achieve the highest availability among all the algorithms using *k*-coteries since the cohorts coterie is *ND*. We have compared the proposed algorithm with Raynal's algorithm [10], the algorithms using *k*-arbiters [2] and (*h*,*k*)-arbiters [9], and Jiang's algorithm [5] to show its superiority.

References

1. Baldoni, R.: An $O(N^{M/(M+1)})$ Distributed Algorithm for the *k*-out of-*M* Resources Allocation Problem. 14th IEEE International Conference on Distributed Computing Systems, (1994) 81-88
2. Baldoni, R., Manabe, Y., Raynal M., Aoyagy, S.: *k*-Arbiter: A Safe and General Scheme for *h*-out of-*k* Mutual Exclusion. Theoretical Computer Science, 193 (1998) 97-112
3. Huang, S.-T., Jiang, J.-R., Kuo, Y.-C.: *k*-Coteries for Fault-Tolerant *k* Entries to a Critical Section. 13th IEEE International Conference on Distributed Computing Systems, (1993) 74-81
4. Jiang, J.-R., Huang, S.-T., Kuo, Y.-C.: Cohorts Structures for Fault-Tolerant *k* Entries to a Critical Section. IEEE Trans. on Computers, 48 (1997) 222-228
5. Jiang, J.-R.: Distributed *h*-out of-*k* Mutual Exclusion Using *k*-Coteries. 3rd International Conference on Parallel and Distributed Computing, Application and Technologies (PDCAT'02), (2002) 218-226
6. Jiang, J.-R.: On the Nondomination of Cohorts Coteries. IEEE Trans. on Computers, 53 (2004) 922-923
7. Lamport, L.: Time, Clocks, and the Ordering of Events in a Distributed System. Communications of ACM, 21 (1978) 558-565
8. Meakawa, M.: A $\sqrt{N}$ Algorithm for Mutual Exclusion in Decentralized Systems. ACM Trans. Comp. Sys., 3 (1985) 145-159
9. Manabe, Y., Tajima, N.: (*h*-*k*)-Arbiter for *h*-out of-*k* Mutual Exclusion Problem. Theoretical Computer Science, 310 (2004) 379-392
10. Raynal, M.: A Distributed Solution for the *k*-out of-*m* Resources Allocation Problem. Lecture Notes in Computer Sciences, Vol. 497. Springer Verlag (1991) 599-609
11. Ricart, G., Agrawala, A. K.: An Optimal Algorithm for Mutual Exclusion in Computer Networks. Communications of ACM, 24 (1981) 9-17