ISBN:1858-1382

J M S K

Jurnal Matematika, Statistika, & Komputasi
http://journal.unhas.ac.id/index.php/jmsk
Vol. 2, No.1, 17-30, Juli 2005

Quorum Based Conflict Resolution Algorithms In Distributed Systems

Armin Lawi[†]

Abstract

Mutual exclusion is one of the most fundamental issues in the study of distributed systems. The problem arises when two or more processes are competing to use a mutual exclusive resource concurrently, i.e., the resource can only be used by at most one process at a time. Synchronizations adopting quorum systems are an important class of distributed algorithms since they are gracefully and significantly tolerate process and communication failures that may lead to network partitioning. Coterie based algorithm is a typical quorum based algorithm for mutual exclusion: A process can use the resource only if it obtains permissions from all processes in any quorum ofcoterie, and since each quorum intersects with each other and each process only issues one permission, the mutual exclusion can be guaranteed. Many quorum systems have been defined based on the relaxation of the properties of coterie system. Each of them is designed to resolve its corresponding problem, e.g., *k*-coterie based algorithm to resolve the *k*-mutual exclusion, local coterie for the generalized mutual exclusion, (*h*, *k*)-arbiter for *h*-out of-*k* resource allocation problem, etc. Therefore, design an algorithm for any distributed conflict resolution problem is only meant to define a new quorum system which can be implemented to the corresponding problem. Since most of distributed conflict resolution problems are designed based on the relaxation of the safety property of mutual exclusion, understanding the way to relaxing the safety property and its quorum system is important to study any kind of conflict resolution problem in distributed systems.

Keywords:*Coterie, distributed algorithm, conflictresolution, Mutual exclusion, quorum system.*

1. Introduction

Distributed systems are a computer system that consists of a collection of processes communicated with each other by sending messages over a communication network. Such systems are increasingly available be cause of decrease in prices of computer processors and the high-bandwidth links to connect them. Distributed systems are used for many reasons: to allow a large number of processes together to solve a problem (as the shared problem) to be much faster than any single process can do alone, to allow the distribution of data in several locations, to allow different processes to share resources such as printers, data items, disks or files, or simply to enable users to transfer the shared data. The communication network in a distributed system can be a local area network such as Ethernet, or a wide area network such as the Internet, or even a small home network.

In many distributed systems, mutually exclusive access is often required for accessing shared resource such as printers, data items, files, memory cells, network buses, etc. When the resource can only be accessed in a mutually exclusive way, i.e., at most one process can use the resource at a time, then it is important to synchronize the accession of processes to the resource so their operations are consistent as a result of concurrent executions and the resource are not failed. This can be observed by the following simple example in most of the

[†]*Lecturer at Mathematics Department, Faculty of Mathematics & Natural Sciences, Hasanuddin University*

distributed replicated database systems. Multiple identical copies of a data item are replicated and stored at some distinct places to facilitate system operations so as to increase system reliability and performance. Clearly, processes may continue to access a data item even when some of the copies are unavailable due to failures and is more likely to find the data it needs nearby. Assume that the initial value of a variable in a replicated data item x is 0 and that there are two processes p_0 and p_1 such that each of them increments x by the following statement in some high-level programming language:

$$x := x + 1;$$

The programmer will naturally assume that the final value of x is 2 after both the processes have executed. However, this may not happen if the programmer does not ensure that $x := x + 1$ is executed atomically in the sense that the effect of the operations must appear indivisible to the user. The execution of p_0 and p_1 may get interleaved as follows: At first, process p_0 reads the initial value 0 of the variable x and increments x by 1. Then, process p_1 reads the incremented value 1 of x and increments it by 1. Process p_0 updates the variable value $x := 1$ and p_2 updates it with 2, and thus they result inconsistent values of the variable to the replicated data in the system.

To avoid this problem, the statement $x := x + 1$ should be executed *atomically*. A part of the code that need to be executed atomically is called *critical section* (CS). The problem of ensuring that CS is executed atomically is called the *mutual exclusion problem* (mutex).

There are many conflict resolution problems have been studied by relaxing the safety requirements of mutex, such as *k*-mutex, generalized mutex, writer-readers problem, *h*-out of-*k* resource allocation, group mutex, etc. In these problems, the distributed system is viewed

	Process p_0	Process p_1
1:	read x;	
2:	$x := x + 1$	
3:		read x;
4:	write x;	$x := x + 1$
5:		write x;

Fig. 1. An Interleaved Operations of Write and Read.

as a set of processes that shares a non-empty set of resources. In fact, if the set of shared resources is explicitly used in specifying the safety requirements for a conflict resolution problem, a more general problem which covers almost all previous distributed conflict resolution problems can be defined easily [1]; i.e., to define safety properties in accessing some distinct CSs.

Synchronizations adopting quorum systems are the well-known algorithms to any distributed conflict resolution problem which is generalized from mutex. The class of these solutions gives a significant interest in fault-tolerant of process and communication failures that may lead to network partitioning. Coterie based algorithm is a typical quorum system for mutex: A process can use the resource only if it obtains permissions from all processes in any quorum of a coterie, and since each quorum intersects with each other and each process only issues one permission, the mutex can be guaranteed. Several quorum systems have also been defined based on the relaxation of the properties of coterie system. Each of them is defined to resolve its corresponding problem, e.g., *k*-coterie based algorithm to resolve the *k*-mutex, local coterie for the generalized mutex, bicoterie for readers/writer problem, (h; k)-arbiter coterie for h-out of-k resource allocation problem, etc. Therefore, design an algorithm for any distributed conflict resolution problem is only meant to define a new quorum system which can be implemented to the corresponding systems.

This article discusses the quorum based mutex algorithm using coterie system and presents some simple coterie constructions. The evaluation of the algorithm performance complexities in the sense of the number of messages, availability and load for each construction is also given. We will also show that any kind of distributed conflict resolution problems which is defined by relaxing the safety property of mutex can be resolved using some corresponding quorum systems which are designed by extending the properties of the coterie.

2. Mutual Exclusion: The First Conflict Resolution Problem

2.1 Specification of the Problem

Consider a distributed system consists of a set of fixed number of processes that shares an indivisible resource. The resource thus just consider as a CS henceforth, e.g., the operations performed on the variable of a replicated data introduced Section 1. The mutex algorithm is the problem to synchronize and coordinate access to the CS such that the following three properties are satisfied at any time:

- ***Safety mutex***[1]: At most one process has permission to executing the CS.
- ***Liveness***: All requests for the CS will be grantedeventually.
- ***Fairness***: The CS is granted by different requestin the order they are made.

The abstraction of this problem can be considered as follows. It is assumed that each process is executing a sequence of instructions that alternate accessed repeatedly. The instructions are divided into four continuous sections of code:

1. A possibly nonterminating *non-critical section* (*NCS*), i.e., the part of code which no request to access the resource,
2. A*trying section*, i.e., the protocol which is used to acquire an access *right* to execute the resource,
3. A terminating *CS*, i.e., the part of code when the process has the access right to executing the resource, and
4. An *exit section*, i.e., the protocol to return the access right back to the system.

A process starts by executing the NCS code. At some point the process might need to execute some code in its CS. Thus, the process should firstly execute a *trying* protocol to get an *access right* so as guaranteeing that while it is executing its CS, no other process is allowed to execute its CS. The process can enter its CS whenever in the possession of the access right. When the process leaves its CS, it executes *exit* protocol and thus returns back to the NCS. The structure of a mutex solution may look as depicted in Figure 2. It can easily to observe that the mutual exclusion is just the problem to design a safety synchronization in the

[1]We use the term of *safety mutex*to distinguish the other safety properties of the generalization problems.

form of *trying* and *exit* protocols to be executed, respectively, immediately before and after the CS in such a way that the three properties of mutex are guaranteed.

do loop forever
    non-critical section;
    trying section;
    critical section;
    exit section;
od;

Fig. 2. An Abstraction of AMutex Solution.

2.2 Quorum-Based Mutex Algorithm

In this subsection, we recall the definition a set system of coterie as the building block of the quorum based algorithm for distributed mutual exclusion problem. Let *U* be the universe set of nodes (or processes) in the system. The term of node may refer to a computer in a network or a copy of some data in a replicated data. Henceforth, we use the terms of node and process interchangeably.

Definition 1.*A nonempty collection of sets* $C(\subseteq 2^u)$ *is a* ***coterie*** *under U iff C satisfies*

1. ***Intersection***: $Q_i \cap Q_j \neq \phi, \forall Q_i, Q_j \in C$.
2. ***Minimality:*** $Q_i \not\subset Q_j, \forall Q_i, Q_j \in C$.
3. ***Non-empty***: $Q \neq \phi,\ and\ \forall Q \subseteq U, \forall Q \in U$.

*The elements Q in a coterie are called**quorums**.*

For example, the following quorum set $C_1 = \{\{2,3\},\{2,4\},\{3,4\}\}$ is a coterie under $U_1 = \{1, 2, 3, 4\}$. It should be noted that not all nodes must appear in a coterie; in particular, node 1 does not appear in either quorum of C_1.

Work of the quorum based algorithm (using coterie) for the mutual exclusion can be outlined as follows. A node u wishing to perform an operation (or to access the shared resource) firstly selects a quorum $Q \in C$, and sends request to all members in *Q*. If u can gather permissions (or acknowledgements) from all members of *Q*, then it can perform the operation. Upon finishing the operation, it returns the permission back all members in the selected quorum. Since each member of quorum has only one permission to issue and by the intersection property of the coterie, safety requirement of the mutex can be guaranteed. The Lamport's logical timestamp given in [2] is implemented to handle dead locks and live-locks by requiring low-priority nodes to yield permissions to high-priority nodes. The smaller the timestamp of a node's request, the higher the priority of the request. Thus, the liveness and fairness requirements are guaranteed.

Domination of Coterie

In [3], the concept of domination of coteries has been introduced.

Definition2.*Let C and D,* $C \neq D$, *be two coteries under a universe set of nodes U. Coterie D* ***dominates*** *C iff* $\exists Q' \in D$ *such that* $Q' \subseteq Q, \forall Q \in C$.

A coterie *C* (under *U*) is **dominated** iff there exists another coterie over *U* which dominates *C*. If there is no such a coterie, then *C* is **nondominated** (or, *C* is an ND-coterie).

For example, let $C_2 = \{\{1, 2\}, \{2, 3\}\}$ *and* $C_3 = \{\{1, 2\}, \{1, 3\}\{2, 3\}\}$be coteries over$U_2 = \{1, 2, 3\}$. The coterie C_2 is dominated by C_3. The coterie C_2 is alsodominated by $\{\{2\}\}$. The coterie C_3 is an ND-coterie since we cannot find any coterie dominated it.

Observe that if a system using a dominated coterie is operational in the occurrence of failures then a system using an ND-coterie is also operational, but the opposite is not always true. Hence, reliability of an ND-coterie is better then the dominated one. Another advantage of ND-coteries is the lower cost of message complexity (since every quorums in an ND-coterie are subset of the quorums in the dominated coterie).

Helpful theorems have been presented in [3] to check whether a coterie is dominated or ND-coterie.

Theorem 1*.Let C be a coterie. C is a dominated coterie if f there exists a set* $X \subseteq U$ *satisfies*

1. $X \not\subset Q, \forall Q \in C$, and
2. $X \cap Q \neq \phi, \forall Q \in C$.

Quorum Constructions

Perhaps the two most obvious coteries are the singleton and the set of majorities. Let *n* is the size of theuniverse set of nodes.

Singleton**:** The set system $S = \{\{v\}\}$ for some $v \in U$ is the singleton quorum system.

Majorities: Quorums in a majority coterie *M* are every sets *Q* with the size of$\left\lceil \frac{n+1}{2} \right\rceil$.

Grid: Suppose that $n = k^2$ for some integer *k*. Arrange nodes into a $\sqrt{n} \times \sqrt{n}$ grid, as shown in Figure 3. A quorum in the Grid *G* is the union of all nodes in one full row and column.

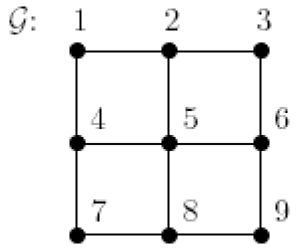

Fig. 3: A 3 x 3 Grid

Tree: Suppose that nodes are arranged into a logical complete *k*-ary tree T with depth *d*, i.e., $\sum_{0 \leq i \leq d} k^i$ for some integer *k*and $d = 0, 1, \ldots$, as depicted in Figure 4. A quorum in the Tree *T* consists of the root, a majority of its children, and a majority oftheir children, and so on.

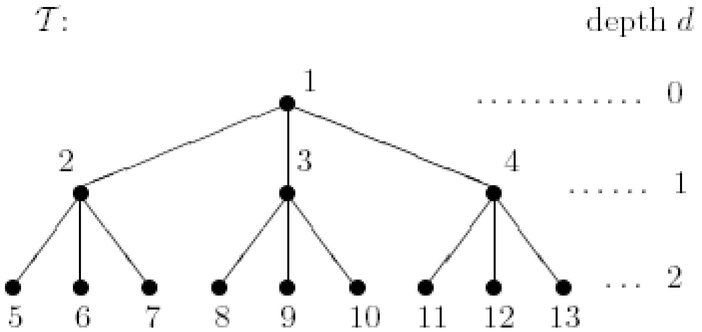

Fig. 4: A Complete 3-ary tree *T* Depth 2

There have been many other algorithms using coteriehas been proposed. In [4], he proposed an algorithmusing coterie constructed from finite projective planes.The size of quorums of the coterie is approximately $\sqrt{n}$ .He showed that coterie based on finite projective planesare the optimal coteries in the sense that each nodehas equal amount of responsibility to the mutex control. Thus, each node requires $O(\sqrt{n})$ messages permutex invocation. Kumar proposed a hierarchical quorum consensus and coterie with multilevel hierarchieswhose quorum size is $n^{0.63}$ [5]. Thus, the size of quorums of a coterie varies from log n to $\left\lceil \frac{n+1}{2} \right\rceil$. In [6], theyinvestigated properties of coteries from the view pointof boolean functions and showed a characterization ofND-coteries.

Measures

The communication cost associated with obtainingmutex using the quorum approach is directly proportional to the quorum size. Other several measures ofquality have also been identified to address the question of which quorum system works best for a given setof nodes; among these, we elaborate on *availability* and*load*.

Availability: A probability that at least one node canbe accessed by the originator operation in the occurrence of node failures. We evaluate the availability of quorumsystems in this article under the assumptions that thereliability of node *v*, i.e., the probability of node *v* beingin operation, is the same value $p \in [0,1]$ for all $v \in U$.

Let $f_c(Q) : 2^U \rightarrow \{0,1\}, \forall Q \subseteq 2^U$, is a characteristic function of a quorum system *C* such that $f_c(Q) = 1$ if there exists a quorum $Q \in C$ and 0 otherwise. Theavailability of quorum system *C*, *A*(*C*), can be evaluatedusing the following formula

$$A(C) = \sum_{\forall Q \in 2^U} f_c(Q).p^{|Q|}.(1-p)^{n-|Q|}$$

Thus, we have the following results for the availability of Singleton, Majority and Grid coteries, respectively

$$A(S) = 1-(1-p)^n,$$

$$A(M) = \sum_{0 \le i \le n-|Q|} \binom{n}{|Q|+i} .\ p^{|Q|+i}\ .\ (1-p)^{n-|Q|-i}$$

$$A(G) = \left((1-(1-p)^{\sqrt{n}}\right)^{\sqrt{n}} - \left(1-p^{\sqrt{n}} - (1-p)^{\sqrt{n}}\right)^{\sqrt{n}},$$

and for the Tree *T*, let $s = \frac{k+1}{2}$, hence

$$A(T) = p + (1-p)\ .\sum_{i=0}^{s-1} \binom{k}{s+i} .\ A(T)_d^{s+i}.(1-A(T)_d)^{s-1-i},$$

$$A(T)_0 = p$$

Load*:* The load of a quorum system is introduced forevaluating load sharing ability. A strategy is a list ofprobability that represents the frequencies of quorumsbeing selected.

Definition 3. *Let* $C = \{Q_1, ..., Q_m\}$ *be a quorum system over U. If* $p \in [0,1]^m$ *is a probability distributionover the quorums* $Q_i \in C_i$, *i.e.,* $\sum_{i=1}^{m} p_i = 1$, *then p is a strategy for C.*

The *load* on a node u is a strategy *p*of picking quorums induces the frequency of accessing node *u*, $\forall u \in U$ .The *system load* on a quorum system *C*, *L*(*C*), is theload on the *busiest* node induced by the *best* possiblestrategy.

Definition 4.*Let p be a strategy for a quorum system C over the universe set of nodes U. The* ***load*** *induced by p on node u is* $l_p(u) = \sum_{u \in Q_j} p_j$, $\forall u \in U$. *The load induced by a p on a quorum system C is*

$$L_p(C) = \max_{u \in U} l_p(u).$$

The ***system load*** *on a quorum system C is*

$$L(C) = \min_{p} \{L_p(C)\},$$

where the minimum is taken over all strategies p.

Naor and Wool [7] gives an helpful results to achieved he optimal load for a *q*-uniform quorum system.

Theorem 2*.Let C be a q-uniform quorum system. Let p be a strategy and* $M \geq 0$. *Then, the optimal systemload over quorum system C is* $L(C) = M = q/n$.

By Theorem 2, we have the following results for system load on the quorum systems of Singleton, Majority,Grid and Tree, respectively.

$$L(S) = 1, \qquad L(M) = \frac{1}{2},$$

$$L(D) = \frac{2\sqrt{n}-1}{n} \text{ and } L(T) = \frac{1}{2} + \frac{(d+1)(k+1)}{k^{d+1}-1}.$$

3. Conflict Resolution Problems and Their Quorum Systems

In this section, we discuss some conflict resolution problems in the distributed systems. We will show that the problems can easily be defined based on the relaxation of the safety property of mutex and their quorum systems can also be designed by extending the properties of coterie.

3.1.*k*-Mutex and *k*-Coteries

A natural generalization of mutex problem is the *k-mutex*. The problem is defined by relaxing the safetyproperty of mutex (without any change to the otherproperties) as follows.

Safety k-mutex: *At most k nodes has permission to executing the CS simultaneously at a time.*

In a distributed environment, the k-mutex problemarises in several interesting applications. For example,it could be used to monitor the number of nodes in adistributed system that are allowed to perform a certain action, such as issuing broadcast messages. In such a case, the system may restrict the number of broadcasting nodes so as to control level of congestion.

Another application in the context of replicated databasesis the *bounded ignorance* problem given in [8], i.e., whentransactions may specify that they do not need to beaware of the k most recent updates to the database.Here also, instead of the traditional database systemthat uses distributed mutex to ensure one update tothe replicated data at any time, several updates may bepermitted simultaneously.

In *k*-mutex, up to *k* nodes are allowed to access theresource simultaneously. Thus, if we consider *k + 1*quorums that grant permission to execute the CS, thenthere must exist at least two among these *k+1* quorumswith a nonempty intersection. We therefore need to extend the intersection property of the coterie. Note thatquorums constructed to ensure the mutex requirementsalso ensure this property. Hence, inorder to eliminatetrivial solution to the *k*-mutex problem, we add an additional restriction of *non-intersection property* [9, 10].

Definition 5. (k-**coteries**) *A nonempty set* $C \subseteq 2^U$ *isa k-coterieunder Piff C satisfies*

1. ***Intersection****:For any* (*k* + *1*)*-set* $K = \{Q_1,...., Q_{k+1}\} \subseteq C$ *, there exists a pair* $K = \{Q_1, Q_j\} \subseteq K$ *such that* $K = Q_1 \cap Q_j \neq \phi, 1 \leq i \neq j \leq k+1$.
2. ***Non-intersection***: *For any h-set* $H = \{Q_1,..., Q_h \in C | Q_i \cap Q_j = \phi, i \neq j\} \subseteq C, h < k$ *, thereexists* $Q \in C$ *such that* $Q \cap Q_i = \phi, \forall Q_i \in H$.
3. ***Minimality***: $Q_i \not\subset Q_j, \forall Q_i, Q_j \in C, i \neq j$.

The second property above is desirable for all values of *k*. When *k* = 1, i.e., in the case of mutex,it is satisfied vacuously. Note that a 1-coterie is justcalled a coterie. As an example, the quorum system $C = \{\{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{3, 4\}\}$ is a 2-coterie under $U = \{1, 2, 3, 4\}$.

The dominance of *k*-coteries can also be defined similarly as in the Definition 2. Let *C* and *D* be two *k*-coteries, and $C \neq D$.

Definition 6. *C* **dominates** *D iff for all* $Q' \in D$ *thereexists* $Q \in C$ *s.t.* $Q \subseteq Q'$. *A k-coterie C is a* **non-dominated***k-***coterie***iffthere is no k-coterie whichdominates C.*

An helpful theorem can also be defined by extendingthe conditions stated in the Theorem 1 as follows.

Theorem 3. *Let C be a k-coterie. C is a dominatedk-coterie if, and only if, there exists a set* $X \subseteq U$ *suchthat the following three conditions are satisfied.*

1. $X \not\subset Q$, *for all* $Q \in C$.
2. *For any k-set* $K = \{Q_1,...Q_k\} \subseteq C$ *, there exists* $Q \subseteq K$ *such that* $Q \cap X \neq \phi$.
3. *Thereexistsh-set* $H = \{Q_1,...Q_h \in C | Q_i \cap Q_j = \phi, i \neq j\} \subseteq C, \quad h < k+1, s.t.$ $X \cap Q_i = \phi, \forall Q_i \in H.$

3.2.Bicoteries and wr-Coteries

The example of conflict resolution problem in thereplicated database systems (which have been introduced in the Section 1) was actually one of the generalization problems of mutex. The execution to the CS,i.e., the operations performed on the same variable ofa same replicated data, consists of two operations with different conditions. Henceforth, this problem is called*writer/readers problem* or *wr-problem* in shortly. Theproblem might be considered into two safety propertiesas follows.

Safety write: *At most one node has permission to executing its write operation into the CS*.

Safety read: *If some nodes are trying to execute theirread operations while no node is executing thewrite operation to the same CS, then they are allowed to executing the CS simultaneously*.

Quorum based algorithm for mutual exclusion can beused for managing wr-problem by having read and writeoperations share the same set of quorums in a coterie.Each copy of a data item is labeled with a *version number* which is initially set to zero and is incremented foreach write operation that has access to it. A read/writeoperation can be proceed only if it obtains permissionsfrom all copies of any quorum. A read operation returnsthe largest version number in the quorum, and a writeoperation updates all of the copies in the quorum. Theintersection property guarantees that at most one operation can be proceed at any time, and at least one copyof a data item has a largest version number in any quorum. However, this mechanism would cause *excessive*operation cost when read operations dominate, which iscommon in many database applications. Thus we needanother type of quorum systems with a more flexibilitycontrolling both operations.

Definition 7. *An ordered pair* $B = \langle W, R \rangle$, *where W and R are sets of subsets of U, is a* ***bicoterie*** *under U if the following two properties hold:*

1. ***wr-intersect***: $W \cap R \neq \phi, \forall W \in W, \forall R \in R$.
2. ***Minimality***: $W_1 \not\subset W_2 \neq \phi, \forall W_1, W_2 \in W$ and $R_1 \not\subset R_2, \forall R_1, R_2 \in R$.

If *W* is a coterie in a bicoterie $B = \langle W, R \rangle$ under*U* (or, bicoterie*B* with an additional **ww-intersection**property), then *B* is called a *writer-readers* coterie (or**wr-coterie**) under *U*. The set of subsets *W* (resp. *R*)of *B* is defined for write (resp. read) operations in thewr-problem.The dominance of bicoteries or wr-coteries can be define as follows.

Definition 8. *Let* $B_1 = \langle W_1, B_1 \rangle$*and* $B_2 = \langle W_2, B_2 \rangle$*be bicoteries over U. Then,* B_1 *is* ***dominated*** *by* B_2 *iff*

1. $\langle W_1, R_1 \rangle \neq \langle W_2, R_2 \rangle$.
2. $\forall Q \in W_1, \exists S \in W_2, S \subseteq Q$.
3. $\forall Q \in R_1, \exists S \in R_2, S \subseteq Q$.

A bicoterie (wr-coterie) *B* is said to be **non-dominated** iff no bicoterie (wr-coterie) dominates *B*. For example, the following $B = \langle W, R \rangle$, where $W = \{ \{1, 2, 3\}, \{1, 2, 4\}, \{1, 3, 4\}, \{2, 3, 4\} \}$ and $R = \{\{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{3, 4\}\}$ is a wr- coterie under the set of nodes $U = \{1, 2, 3, 4\}$.

3.3.Group Mutex andGroup*k*- Mutex

The *group mutex* problem posed in [11] generalizes the classical mutex and wr-problems. In this problemn nodes repeatedly access *m*different resources. Nodesthat have requested to execute the same resource maydo it concurrently. However, nodes that have requested to attend different resources may not execute their resources at the same time. Thus, the group mutex mayalso be defined into two safety properties as follows.

Safety group-mutex: *At most one resource is allowed to being access by some nodes simultaneously*.

Safety concurrent-entering: *If some nodes are trying to execute the same resource while no node isexecuting a different resource, then they are allowed to executing their CS concurrently*.

In the group mutex, [11] have proposed an *m-groupquorum system* for quorum based group mutex algorithm. However, construction such a good quorum system (i.e., an ND *m*-group quorum system) arises amore difficult problem. Moreover, since the problemonly relaxing the safety property of mutex as in thewr-problem, the coterie based algorithm for mutex candirectly be adopting to resolving this problem; i.e., theconflicting nodes simply use a coterie to manage theirmutual exclusive accessions to the requested resources.

Definition 9. *An m-group quorum system* $G = (C_1, ..., C_m)$ *over a set of nodes U consists of m sets,where each* $C_i \subseteq 2^U$ *is a set of subsets U satisfying thefollowing two conditions*:

1. *For all* $Q_i \in C_i$ *and for all* $Q_j \in C_j, 1 \le i, j \le m, i \ne j,$ *then* $Q_i \in Q_j \ne \phi$.
2. *For all* $Q_1, Q_2 \in C_i, 1 \le i \le m,$ *and* $Q_i \ne Q_2$, *then* $Q_1 \subseteq Q_2$.

An example of *m*-group quorum system is given in[11] using the sacrificial quorum system, i.e., a mappingof nodes from the surface of cubic space. However, theresults of this method always give dominated *m*-groupquorum system. As an example, the following $G = \{C_1, C_2, C_3, C_4\}$ is a 4-quorum system under the set $U = \{1, 2, ..., 9\}$, where

$$C_1 = \{\{1, 2, 3\}, \{4, 5, 6\}, \{7, 8, 9\}\},$$
$$C_2 = \{\{1, 6, 8\}, \{2, 4, 9\}, \{3, 5, 7\}\},$$
$$C_3 = \{\{1, 5, 9\}, \{2, 6, 7\}, \{3, 4, 8\}\}, \text{ and}$$
$$C_4 = \{\{1, 4, 7\}, \{2, 5, 8\}, \{3, 6, 9\}\}.$$

The group *k*-mutex, i.e., a combined problem of *k*-mutex and group mutex is considered in [12, 13] for aparallel shared-memory environment. The problem just relaxing the safety group-mutex property to allow forat most *k* resources might be accessed by some nodessimultaneously.

Safety group k-mutex: *At most k resources are allowed to being accessed by some nodes simultaneously*.

The *k*-coterie based algorithm for k-mutex is directlyadopted to resolve this problem. The only different isthat when a node *u* wish to use the resource r_v, then *u*chooses a quorum in the coterie C_v, and the rest worksthe same as for k-coterie. The quorum based algorithmfor *m* resources can be presented as in Figure 5.

3.4.The (*m*, *h*, *k*)-Resource Allocation Problem

Recently, Lawi *et al.* [1, 14, 15] and Joung [16] independently introduced and defined (*m*, *h*, *k*)-*resource allocation* as a general conflict resolution problem which relaxes the safety requirement of the k-mutex and GME problems. The problem models and designs a conflict resolution in a distributed system consisting of n nodes which share *m* resources. The system is said to be (*m*, *h*, *k*)-resource allocated if the following safety properties are hold.

Group h-mutex: *At most h* (*out of m*) *resources can be used by some nodes simultaneously at a time*.

***k*-concurrent entering**: *At most k (out of n) concurrent nodes can use the same resource at a time*.

This problem can cover all the conflict resolution problems mentioned before. If the system only consisting of a single shared resource ($m = 1$), the problem corresponds to the mutex when $k = 1$, and it corresponds to the k-mutex when k is constantly determined. If $m > 1$, the problem corresponds to the GME when $h = 1$ and k is undetermined, it corresponds to the *generalized mutex* given in [17], when $k = 1$ and h is undetermined, and it corresponds to the group k-exclusion [12, 18], when $h = 1$ and k is constantly determined. The problem also covers some generalized problems that have not yet been studied such as when $k \geq 1$ and h is constantly determined, and when k is constantly determined and h is undetermined (and conversely). Moreover, the problem also corresponds to some new generalizations of the wr-problem [19, 20], when its requirements are applied after relaxing or leaving strained.

A simple approach to (m, h, k)-resource allocation can use an l-coterie based mutex algorithm. The two requirements of the group h-exclusion and the k-concurrent entering are independently solved using the h- and k-mutex algorithms respectively, and a node can use a critical resource only if it gets the access right from both of the h- and k-coterie based algorithms. This algorithm is a natural one, however, the number of messages required per entry to the resource will be doubled to the original algorithm. Therefore, it is inefficient in terms of the message complexity. Intuitively, the number of messages can be reduced if we can find a new quorum system which combines the h- and k-coteries into a single quorum system.

Trying Section{ // *When* u_i *wishes to access a resource* r_v
ts_i++; // ts_i is u_i's current logical time
Select a quorum Q *in* $\mathcal{C}_v$;
send $\mathtt{req}(ts_i, u_i)$ **to** u_j, $\forall u_j \in Q$;
Add $u_j (\in Q)$ *answering* `ack` *into* $\mathtt{AGREE}_i$;
if (*there is a* $Q(\in \mathcal{C}_v) \subseteq \mathtt{AGREE}_1$) {
state := **Critical Section**; }
else-if { // *If there exists* $u_j(\in Q)$ *answers* `wait`
Add u_j *answering* `wait` *into* $\mathtt{DISAGREE}_i$;
Select another quorum $Q' \in \mathcal{C}_v$ *such that*
$Q' \cap \mathtt{DISAGREE}_1 = \emptyset \bigwedge Q' = \max\{|Q \cap \mathtt{AGREE}_1|\}$;
if (*there is no quorum satisfy*) {
state := **Wait**; }
$Q := (Q' - Q)$ *and* **goto** line 3; } }

Exit Section { // *When user* u_i *leaves resource* r_v
send `exit` **to** $\forall u_j \in (\mathtt{AGREE}_1 \cup \mathtt{DISAGREE}_1)$; }

When u_i **receives** $\mathtt{req}(ts_j, u_j)$ **message:**
if $(\mathtt{PERM}_1 = \emptyset)$ {
send `ack` **to** u_j and *add* $\mathtt{req}(ts_j, u_j)$ *to* $\mathtt{PERM}_i$; }
else-if { // If there exists $(ts_x, u_x) \in \mathtt{PERM}_1$
// Let (ts_y, u_y) is the highest priority in $\mathtt{QUEUE}_i$;
Insert $\mathtt{req}(ts_j, u_j)$ *into* $\mathtt{QUEUE}_i$;
if $((ts_j, u_j) > \max\{(ts_x, u_x), (ts_y, u_y)\})$ {
send `reclaim` **to** u_x; }
else-if {
send `wait` **to** u_j; }}

When u_i **receives** `exit` **message from** u_j**:**
Remove request u_j *from* $\mathtt{PERM}_i$;
if $(\mathtt{QUEUE}_1 \neq \emptyset)$ {
// Let (ts_y, u_y) is the highest priority in $\mathtt{QUEUE}_i$;
Move $\mathtt{req}(ts_y, u_y)$ *from* $\mathtt{QUEUE}_i$ *to* $\mathtt{PERM}_i$;
send `ack` **to** u_y; }

When u_i **receives** `reclaim` **message from** u_j**:**
if (u_i *not in critical section* and $u_j \in \mathtt{AGREE}_1$) {
Move u_j *from* $\mathtt{AGREE}_i$ *to* $\mathtt{DISAGREE}_j$;
send `relinquish` **to** u_j; }

When u_i **receives** `relinquish` **message from** u_j**:**
// Let (ts_x, u_x) is the highest priority in $\mathtt{QUEUE}_i$;
if $(ts_x, u_x) > (ts_j, u_j)$ {
Move $\mathtt{req}(ts_j, u_j)$ *from* $\mathtt{PERM}_i$ *to* $\mathtt{QUEUE}_i$;
send `ack` **to** u_x;
Move $\mathtt{req}(ts_x, u_x)$ *from* $\mathtt{QUEUE}_i$ *to* $\mathtt{PERM}_i$; }

Fig. 5. The (m, h, k)-coterie Based Algorithm.

Let C and C'' be two k-coteries under U and P', respectively. We say that they are *disjoint* if $Q \cap Q' = \phi, \forall Q \in C, \forall Q' \in C'$. Clearly they are disjoint if U and P' are disjoint.

The new quorum system, (m,h, k)-coterie, is defined as follows.

Definition 10.((m, h, k)-***coteries***) *A collection ofsets* $B = \{C_1, ..., C_m\}$, *where* C_i *is a k-coterie underU,* $\forall C_i \in B$ *is an*(m, h, k)-***coterie****under U iff thefollowing conditions hold:*

1. ***Disjoint****: For any* $l(< k)$ *mutually disjoint elements* $C_1, ..., C_l \in B$, *there is another elements* $C \in B$ *such that C and* $C_i^{'}$ *are disjoint for all* $1 \le i \le l$.
2. ***Bicoteries****: For any (h+1)-set* $\{C_i^{'}, ..., C_{h+1}^{'}\} \subseteq B$, *there exists a pair* $\{C_i^{'}, C_j^{'}\}$ *form bicoteries,* $\forall 1 \le i \ne j \le h+1$.

For example, the quorum system $B_1 = \{C_1, C_2, C_3, C_4\}$ is a (4,2,2)-coterie on a set $U = \{1, 2, ...,16\}$ where

$$C_1 = \{\{1, 2, 5, 7\}, \{3, 4, 6, 8\}\},$$
$$C_2 = \{\{5,6, 9, 11\}, \{7,8, 10, 12\}\},$$
$$C_3 = \{\{9, 10, 13, 15\}, \{11,12, 14, 16\}\}, \text{ and}$$
$$C_2 = \{\{1, 3, 13, 14\}, \{2, 4, 15, 16\}\}.$$

4. Conclusions

In this article, we have discussed some quorum based distributed conflict resolution algorithms in distributedsystems. We discuss the coterie based mutex algorithmfirstly and present some simple constructions of coteriesystem. The evaluation of the algorithm performancecomplexities in the sense of the number of messages,availability and load for each construction have alsobeen given to measure which quorum system works bestfor a given set on nodes.

We have also showed the relaxation of the safety property of mutex in defining other conflict resolution problems in distributed systems, and some of their corresponding quorum systems which are designed by extending the properties of the coterie have also beenpresented. We may conclude that almost all distributedconflict resolution problem can be defined based on therelaxation of the safety mutex property with an additional concurrent entering property. Some interesting future works for the generalization problems areto explore the performance measurements of the extended quorum systems and to investigate their properties which may differ from their superior coterie system.

References

[1] Lawi, A., Yamashita, M., 2003, “A quorum based *m*-group (*h*,*k*)-exclusion algorithm”, *Proc. International Symposium on Information Science andElectrical Engineering* (*ISEE2003*), pp : 405-408.

[2] Lamport, L., 1978, “Time, clocks and the ordering ofevents in a distributed system”, *Communicationsof The ACM 21*, pp : 558-565.

[3] Garcia, H., Barbara, D., 1985, “How to assignvotes in a distributed system”, *Journal of The ACM32*, pp : 841-860.

[4] Maekawa, M., 1985, “A √*N*algorithm for mutual exclusion in decentralized systems”, *ACM Transactionon Computer Systems 3*, pp : 145-159.

[5] Kumar, A., 1991, “Hierarchical quorum consensus: anew algorithm for managing replicated data”, *IEEETransactions on Computers 4*, pp : 996-1004.

[6] Ibaraki, T., Kameda, T., 1993, “A theory of coteries:Mutual exclusion in distributed systems”, *IEEETransaction on Parallel and Distributed Computing 4*, pp : 779-794.

[7] Naor, M., Wieder, U., 2003, “Scalable and dynamic quorum systems”, *Proc. Principles of DistributedComputing* (*PODC*), pp : 114-122.

[8] Khrisnakumar, N., Bernstein, A., 1991, “Bounded ignorance in replicated systems”, *Proc. 10th ACMSymp. Principles of Database Systems,* pp : 63-74.

[9] Fujita, S., Yamashita, M., Ae, T., 1991, “Distributed*k*-mutual exclusion problem and *k*-coteries”, *Proc. 2nd International Symposium on Algorithms*(*LNCS 557*), pp:22-31.

[10] Huang, S., Jiang, J., Kuo, Y., 1993, “*k*-coteries for fault-tolerant *k* entries to a critical section”, *Proc.13th International Conf.Dist. ComputingSystems* (*DISC*), pp:74-81.

[11] Joung, Y.J., 2003, “Quorum-based algorithms for groupmutual exclusion”, *IEEE Transaction on Paralleland Distributed Systems 14*, pp : 463-476.

[12] Vidyasankar, V., 2003, “A simple group mutual-exclusion algorithm”, *Information Processing Letters 85*, pp : 79-85.

[13] Takamura, M., Altman, T., Igarashi, Y., 2004, “Speedupof vidyasankar's algorithm for the group *k*-exclusion problem”, *Information Processing Letters91*, pp : 85-91.

[14] Lawi, A., Oda, K., Yoshida, T., 2006, “A quorum based(*m*,*h*,*k*)-resource allocation algorithm”, *Proc.International Conference on Parallel and Distributed Application and Techniques* (*PDPTA*), pp : 399-405.

[15] Lawi, A., Oda, K., Yoshida, T., 2006, “Quorumbased distributed conflict resolution algorithm forbounded capacity resources”, *Parallel & Distributed Processing& Applications* (*ISPA-06*):*Lecture Notes in Comp. Science* (*LNCS*) *4331*,pp : 135-144.

[16] Joung, Y.J, 2004, “On quorum systems for group resources with bounded capacity”, *Proc.18th International Conf. on Distributed Computing*(*LNCS 3274*), pp : 86-101.

[17] Kakugawa, H., Yamashita, M., 1996, “Local coteries anda distributed resource allocation algorithm”,*Trans. Information Processing Society Japan37*, pp:1487-1498.

[18] Lawi, A., Oda, K., Yoshida, T., 2005, “A quorum basedgroup *k*-mutual exclusion algorithm for open distributed environments”, *Parallel and DistributedProcessing and Applications: Lecture Notes inComputer Science (LNCS) 3758*, pp : 119-125.

[19] Manabe, Y., Tajima, N., 2004, “ (*h*,*k*)-arbiters for *h*-out-of-*k*-mutual exclusion problem”, *Theoretical Computer Science 310*, pp : 379-392.

[20] Datta, A.K., Hadid, R., Villain, V., 2003, “A new self-stabilizing *k*-out-of-exclusion algorithm on rings”, *Self-Stabilizing Systems* (*LNCS 2704*), pp : 113-128.

[21] Agrawal, D., Abbadi, A.E., 1991, “An efficient and fault tolerant algorithm for distributed mutual exclusion”, *ACM Trans. Computer Systems 9*, pp : 1-20.

[22] Bernstein, P., Goodman, N., 1983, “The failure and recovery problem for replicated databases”, *Proc.Principles of Distributed Computing* (*PODC*), pp : 114-122.

[23] Carvalho, O., Roucairol, G., 1983, “On mutual exclusion in computer networks”, *Communication of TheACM 26*, pp : 146-147.

[24] Dijkstra, E., 1965, “Solution of a problem in concurrent programming control”, *Communications of TheACM 8*, pp : 569.