A Quorum Based Group k-Mutual Exclusion Algorithm for Open Distributed Environments

Armin Lawi, Kentaro Oda, and Takaichi Yoshida

Porgram of Creation Informatics, Kyushu Institute of Technology,
680-4 Kawazu, Iizuka, Fukuoka 820, Japan

Abstract. This paper presents a quorum-based group k-mutual exclusion algorithm for open distributed computing systems that can evolve their behavior based on membership changes in the environment. The algorithm consists of two main layers; the quorum-consensus and quorum-reconfiguration. The quorum consensus layer is used to handle requests from and to the application layer, and it directly adopts a proposed k-coterie based algorithm of the group k-mutual exclusion in the static environments without any change to its protocol. Thus, the message complexity and quorum availability are the same as in the static environments. The quorum reconfiguration reconstructs information structure of the k-coterie by simply implementing the properties of two quorum input operations called coterie-join and coterie-cross. The reconfiguration layer is simple to use and has a great ability to complete any operation during reconfiguration powerfully thus system does not enter the halt state.

1 Introduction

The *distributed mutual exclusion* is one of the most fundamental issues in the study of distributed control and management problems that arises when multiple computing nodes compete for a shared resource in an uncoordinated way. The problem is to design a safety synchronization such that at most one node is allowed to use the resource at a time. The problem of *k-mutual exclusion* (k-mutex) and *group mutual exclusion* (GME) are the two well studied natural generalizations of the mutual exclusion. The k-mutex guarantees at most k (≥ 1) nodes can be allowed to use a single resource simultaneously, and the GME synchronizes conflicting nodes in sharing m resources such that at most one resource can be used by some concurrent nodes. Recently, Vidyasankar [1] introduced group k-mutex as the generalization of the k-mutex and GME problems in a shared-memory environment. The problem is to design a conflict resolution such that at most k (out of m) resources can be used by some concurrent nodes.

As mentioned, let us consider a distributed system consisting of n nodes, which share undetermined number of resources[1]. The system is said to be *group k-mutual exclusive* if the following requirements hold:

[1] The paper have further relaxed the assumption of the original problem that the nodes have no knowledge about the entire set of the shared resources.

Y. Pan et al. (Eds.): ISPA 2005, LNCS 3758, pp. 119–125, 2005.
© Springer-Verlag Berlin Heidelberg 2005

- **k-mutual exclusion:** at most k resources are allowed to be used by some concurrent nodes at a time.
- **concurrent entering:** nodes which request the allowable resources can use them simultaneously at a time.
- **liveness:** a node requesting a resource will eventually succeed.

Quorum consensus approaches are the well-known solution to any conflict resolution which is generalized from the mutual exclusion. The class of these solutions gives a significant interest in fault-tolerant of node and communication failures that may lead to network partitioning [2, 3]. Coterie based algorithm is a typical quorum consensus for mutual exclusion: A node can use the resource only if it obtains permissions from all nodes in any quorum of a coterie, and since each quorum intersects with each other and each node only issues one permission, the mutual exclusion can be guaranteed. In the GME, Joung [4] have proposed an m-group quorum system for GME quorum consensus, however, construction of such a good quorum system (i.e., a non-dominated m-group quorum system) arises a more difficult problem. Moreover, the coterie based of the mutual exclusion can directly be adopted to this problem; i.e., the conflicting nodes simply use a coterie to manage their mutual exclusive accessions to the requested resources. The k-coterie based algorithms are a particular quorum consensus on the k-mutex problem. There at most k pairwise disjoint quorums in a k-coterie, thus at most k nodes can use it so as to achieve the k-mutex safety requirement. Furthermore, the k-coterie based algorithm can also be used for the group k-mutex in the static environments. In this paper, we firstly present a k-coterie based group k-mutex algorithm in the static environments and adopt it forward to the open distributed environments.

Open distributed computing systems are built on the highly volatile networks in the sense that the rate of membership changes (i.e., nodes joining and leaving the system) is very high. The system consists of a set $\mathcal{P}$ of an undetermined number of nodes which communicate in a message passing manner using a reliable FIFO bidirectional link and share a nonempty set $\mathcal{R}$ of an undetermined number of resources. A node can be created and removed either by user or by another node or even joining and leaving the system by itself. We assume that each node has its own memory and it may fail according to fail-stop failure model in [5]. If a node is created (or join), removed (or leave) or get fails then it can be detected by some other nodes in the system. When a new node is created or joining to the system, it should firstly verify the current configuration of the system.

The existing distributed quorum consensus can run correctly on top of network layer of the open distributed environments, since they are designed as a resilient solution against node and communication failures. However, the membership changes by the leaving and joining nodes will adversely decrease availability of the quorum system. The contention is the reliability that can be gained by developing a core set of distributed algorithms that are aware of the underlying volatility in the network. Lawi *et al.*[6] have proposed a wait-avoidance mechanism in reconfiguring quorum system for mutual exclusion so as to prevent this

drawback. Their algorithm mainly consists of two layers that separately works; the quorum-consensus and -reconfiguration. The quorum consensus layer is used to handle requests from and to the application layer, and it directly adopts the coterie based algorithm for group mutual exclusion in the static environments. The quorum reconfiguration layer reconstructs information structure of the coterie by implementing the two quorum input operations called coterie-join and -cross operations. The coterie join operation is used when a set of nodes have leaved from the system while some others are joining, and the coterie cross is implemented to the algorithm when there is only a set of joining nodes enter the system. In this paper, we extend the results in [6] by showing that; the k-coterie based algorithm of the group k-mutex can also be used in their quorum consensus layer, and the quorum reconfiguration layer can also be adopted in reconfiguring k-coteries.

2 The Quorum Consensus Layer

2.1 k-Coteries

Definition 1. [7] A nonempty set of sets, $\mathcal{C}$, is a *k-coterie* under a set of nodes $\mathcal{P}$ iff $\mathcal{C}$ satisfies the following properties:

1. **Non-intersection:** For any h-set $\mathcal{H} = \{Q_1, \ldots, Q_h \in \mathcal{C} \mid Q_i \cap Q_j = \emptyset, i \neq j\}$, $h < k$, there exists $Q \in \mathcal{C}$ such that $Q \cap Q_i = \emptyset$, $1 \leq i \leq h$.
2. **Intersection:** For any $(k+1)$-set $\mathcal{K} = \{Q_1, \ldots, Q_{k+1}\} \subseteq \mathcal{C}$, there exists a pair Q_i, $Q_j \in \mathcal{K}$ such that $Q_i \cap Q_j \neq \emptyset$, $1 \leq i, j \leq k+1$, $i \neq j$.
3. **Minimality:** $Q_i \nsubseteq Q_j$, $\forall Q_i, Q_j \in \mathcal{C}$, $i \neq j$. □

The quorum consensus layer has two sections that alternate accessed repeatedly: a possibly nonterminating *noncritical section* (NCS) and a terminating *critical section* (CS). The layer stays in the NCS when there is no request to use a resource from the application layer and enters the CS whenever it has an *access right* to a requested resource. The CS is a specified part of the code in which node uses the resource. A node executes a *trying* protocol to entreat an access right so as to enter the CS, and executes an *exit* protocol after leaved the CS and thus returns back to the NCS again. Therefore, the problem in this layer is to design a safety synchronization in the form of *trying* and *exit* protocols to be executed, respectively, immediately before and after the CS which satisfies the safety requirements of group k-mutex (as mentioned in Section 1).

Let $\mathcal{C}$ be a k-coterie. Each node in $\mathcal{P}$ has local variables `AGREE`, `DISAGREE`, `PERM` and `QUEUE`, respectively, keeps the set of nodes which have agreed (by message `ack`), the set of nodes which have not yet agreed (by message `wait`), the set of requests in which p_i has give its permission but has not yet received a message `reclaim`, and the ordered set of requests in which p_i has replied `wait` messages. For conciseness, we roughly give a curt description how the k-group mutex algorithm works for this layer in Figure 1.

Trying Section { // *When node p_i wishes to access a resource r_i*
1: Selects a quorum Q from $\mathcal{C}$;
2: **send** req(t_i, p_i, r_i) **to** $\forall p \in Q$; // *t_i is the p_i's current logical time*
3: Inserts $p_j (\in Q)$ answering ack into AGREE;
4: **if** $(\exists Q \in \mathcal{C}, Q \subseteq$ AGREE$)$ **then** *state* := **Critical Section**;
5: **else-if** { // *If there exists $p_j (\in Q)$ answers* wait
6: Inserts p_j answering wait into DISAGREE;
7: Selects another quorum $Q' \in \mathcal{C}$ such that
$(Q' \cap$ DISAGREE $= \emptyset)$ *and* $(Q' = \max\{|Q \cap$ AGREE$|\})$;
8: **if** (*there is no quorum satisfy*) **then** *state* := **Wait**;
9: $Q := (Q' - Q)$ *and* **goto** line 2; } }

Exit Section { // *When node p_i leaves resource r_i*
1: **send** exit **to** $\forall p_j \in$ (AGREE $\cap$ DISAGREE)}

When p_i **receives** req(t_j, p_j, r_j) **message** {
1: // Let $\langle t_y, p_y \rangle$ is the highest priority in QUEUE;
2: **if** (PERM $= \emptyset$ or $r_j = r_y$) **then**
3: **send** ack **to** p_j and inserts req(t_j, p_j, r_j) to PERM;
4: **else-if** { // *If there exists* req(t_x, p_x, r_x) *in* PERM and $r_j \neq r_y$
5: Inserts req(t_j, p_j, r_j) into QUEUE;
6: **if** $\langle t_j, p_j \rangle > \min\{\langle t_x, p_x \rangle, \langle t_y, p_y \rangle\}$ **then send wait to** p_j;
7: **else-if** // *If $\langle t_j, p_j \rangle$ is the highest priority in* QUEUE
8: **send** reclaim **to** p_y; } }

When p_i **receives** exit **message from** p_j {
1: Removes req(t_j, p_j, r_j) from PERM;
2: **if** (PERM $= \emptyset$ and QUEUE $\neq \emptyset$) **then** {
3: // *Let $\langle t_y, p_y \rangle$ is the highest priority in* QUEUE;
4: **for each** (req$(t_j, p_j, r_j) \in$ QUEUE and $r_j = r_y$) {
5: Moves req(t_j, p_j, r_j) from QUEUE to PERM;
6: **send** ack **to** p_j; } } }

When p_i **receives** reclaim **message from** p_j {
1: **if** (p_i *not in CS* and $p_j \in$ AGREE) **then** {
2: Moves p_j from AGREE to DISAGREE;
3: **send** relinquish **to** p_j; } }

When p_i **receives** relinquish **message from** p_j**:** {
1: // *Let $\langle t_y, p_y \rangle$ is the highest priority in* QUEUE;
2: **send** ack **to** p_y;
3: Inserts req(t_y, p_y, r_y) into PERM }

Fig. 1. A distributed group k-mutex algorithm for static environments

2.2 Non-dominated k-Coteries

Definition 2. [3] $\mathcal{C}$ is a *dominated* k-coterie under $\mathcal{P}$ iff there exists a k-coterie $\mathcal{D}$ (under $\mathcal{P}$) such that

1. $\mathcal{C} \neq \mathcal{D}$,
2. $\forall Q \in \mathcal{C}$, $\exists S \in \mathcal{D}$, $S \subseteq Q$.

If there is no such $\mathcal{D}$, then $\mathcal{C}$ is *non-dominated* (or, an ND k-coterie). □

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

Neilsen [8] have proposed a helpful theorem to check whether a coterie is dominated or not. The theorem can be relaxed to further the k-coteries as well.

Theorem 1. $\mathcal{C}$ is a dominated k-coterie under a set of node $\mathcal{P}$ iff there exists a set $X \subseteq \mathcal{P}$ such that the following conditions hold.

1. **Non-intersection:** There exists h-set $\mathcal{H} = \{Q_1, \ldots, Q_h \in \mathcal{C} \mid Q_i \cap Q_j = \emptyset, i \neq j\}$, $h < k-1$, such that $X \cap Q_i = \emptyset$.
2. **Intersection:** For any k-set $\mathcal{K} = \{Q_1, \ldots, Q_k\} \subseteq \mathcal{C}$, there exist $Q_i \in \mathcal{K}$ such that $Q_i \cap X \neq \emptyset$.
3. **Minimality:** $\forall Q \in \mathcal{C}$, $Q \not\subseteq X$. ∎

3 Quorum Reconfiguration

The quorum reconfiguration layer mainly based on the reconfiguration algorithm posed by Lawi *et al.*[6] which uses two quorum input operations in reconfiguring the quorum system of the mutual exclusion; i.e., coterie-join and -cross. We have extended their results for k-coteries and directly adopt them in this layer.

For the following subsections, let $\mathcal{C}_1$ and $\mathcal{C}_2$ be k-coteries under $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively, and $\mathcal{P}_1 \cap \mathcal{P}_2 = \emptyset$.

3.1 Coterie Join Operation

Definition 3. [8] Let x be a node in $\mathcal{P}_1$. A *coterie join* operation for inputs $\mathcal{C}_1$ and $\mathcal{C}_2$ produces a quorum set $(\mathcal{C}_1 \odot_x \mathcal{C}_2)$ defined by

$$(\mathcal{C}_1 \odot_x \mathcal{C}_2) = \{(Q_1 - \{x\}) \cup Q_2 \mid Q_1 \in \mathcal{C}_1, Q_2 \in \mathcal{C}_2 \text{ and } x \in Q_1\} \cup \{Q_1 \mid Q_1 \in \mathcal{C}_1 \text{ and } x \notin Q_1\}.$$ □

Jiang and Huang [9] have proved the following results.

Theorem 2. Let $\mathcal{C}_3 = (\mathcal{C}_1 \odot_x \mathcal{C}_2)$, then

1. $\mathcal{C}_3$ is a k-coterie under $\mathcal{P}_3 \subseteq \mathcal{P}_1 \cup \mathcal{P}_2$.
2. $\mathcal{C}_3$ is an ND k-coterie only if $\mathcal{C}_1$ and $\mathcal{C}_2$ are both ND k-coteries.
3. $\mathcal{C}_3$ is dominated, if either $\mathcal{C}_1$ or $\mathcal{C}_2$ is dominated. ∎

The following can easily be proved using mathematical induction.

Corollary 3. Let $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_m$ be k-coteries under $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_m$, respectively. For any $X = \{x_1, x_2, \ldots, x_{m-1} \mid x_i \in \mathcal{P}_i\}$, then $\mathcal{C} = (\mathcal{C}_1 \odot_{x_1} \cdots \odot_{x_{m-1}} \mathcal{C}_m)$ is a k-coterie under $\mathcal{P} \subseteq \cup_{i=1}^{m} \mathcal{P}_i$. ∎

3.2 Coterie Cross Operation

Definition 4. [6] A *coterie cross* operation for inputs $\mathcal{C}_1$ and $\mathcal{C}_2$ produces a quorum set defined by, $(\mathcal{C}_1 \otimes \mathcal{C}_2) = \{Q_1 \cup Q_2 \mid Q_1 \in \mathcal{C}_1 \text{ and } Q_2 \in \mathcal{C}_2\}$. □

Theorem 4. [6] Let $\mathcal{C}'$ and $\mathcal{C}''$ be *coteries* under $\mathcal{P}'$ and $\mathcal{P}''$, respectively, and $\mathcal{P}' \cap \mathcal{P}'' = \emptyset$. If $\mathcal{C} = (\mathcal{C}' \otimes \mathcal{C}'')$, then

1. $\mathcal{C}$ is a coterie under $\mathcal{P} \subseteq \mathcal{P}' \cup \mathcal{P}''$.
2. $\mathcal{C}$ is an ND-coterie only if $\mathcal{C}'$ and $\mathcal{C}''$ are both ND-coteries.
3. $\mathcal{C}$ is dominated, if either $\mathcal{C}'$ or $\mathcal{C}''$ is dominated. ∎

We have extended results in Theorem 4 for k-coteries as follows.

Theorem 5. Let $\mathcal{C}_4 = (\mathcal{C}_1 \otimes \mathcal{C}_2)$. Then,

1. $\mathcal{C}_4$ is a k-coterie under $\mathcal{P}_4 \subseteq \mathcal{P}_1 \cup \mathcal{P}_2$.
2. $\mathcal{C}_4$ is an ND k-coterie only if both $\mathcal{C}_1$ and $\mathcal{C}_2$ are ND k-coteries.
3. $\mathcal{C}_4$ is dominated only if either $\mathcal{C}_1$ or $\mathcal{C}_2$ is dominated k-coterie.. ∎

3.3 The Reconfiguration Algorithm

The quorum reconfiguration layer simply implements the two operations introduced in the previous two subsections, but for the conciseness, we roughly outline how it works as follows. Let $\mathcal{C}$ be the the current k-coterie of the system.

1. *When there are sets joining nodes X and leaving nodes Y*: The algorithm firstly partitions the set X into m ($\leq |Y|$) disjoint sets and constructs m independent k-coteries $\mathcal{C}_1, \ldots, \mathcal{C}_m$ under $X_1, \ldots, X_m$, respectively, and creates a new coterie $\mathcal{C}_{\text{temp}} = \mathcal{C}$. Each node $y_i \in Y$ is replaced by $\mathcal{C}_i$ iteratively using coterie cross operation, $\mathcal{C}_{\text{temp}} = \mathcal{C}_{\text{temp}} \odot_{y_i} \mathcal{C}_i$, $i = 1 \ldots, m$. The iterated result of $\mathcal{C}_{\text{temp}}$ is stored to $\mathcal{C}$ as the new quorum configuration.
2. *When there is only a set X of joining nodes*: The algorithm simply creates a k-coterie $\mathcal{C}'$ under X and restores $\mathcal{C}$ with $(\mathcal{C} \otimes \mathcal{C}')$ as the new configuration.

Note that the coterie cross operation can also be implemented in case 1, however, the result k-coterie will be dominated. Let $\mathcal{C}'$ be k-coterie under the set X of joining nodes and $\mathcal{P} \cap X = \emptyset$. Let $x \in \mathcal{P}$ is the leaving node, then

$$(\mathcal{C} \otimes \mathcal{C}') = (\mathcal{C} \odot_x \mathcal{C}') \setminus \{Q \mid Q \in \mathcal{C} \text{ and } x \notin Q\}$$

Thus, there exists a set $Z \in \{Q \mid Q \in \mathcal{C} \text{ and } x \notin Q\}$ satisfies the Theorem 1.

4 Performance Analysis

The number of messages required per entry to the CS is the same as for the mutual exclusion [10] and hence for the k-mutex algorithm [7] in the static environments. The message complexity of the algorithm in the best case is 3ϵ and can be bounded from above by 6ϵ in the worst case, where $\epsilon = \max\{|Q| \mid Q \in \mathcal{C}\}$.

Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be k-coteries under $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively, and $\mathcal{C} = \mathcal{C}_1 \odot_x \mathcal{C}_2$, $x \in \mathcal{P}_1$, or $\mathcal{C} = \mathcal{C}_1 \otimes \mathcal{C}_2$.

Theorem 6. $|Q| \leq 2 \max\{|Q'| \mid Q' \in \mathcal{C}_1 \text{ or } Q' \in \mathcal{C}_2\}$, $\forall Q \in \mathcal{C}$. ∎

Now, let $\|\mathcal{C}\|$ (resp., $\|\mathcal{C}_1\|$ and $\|\mathcal{C}_2\|$) defines rank of coterie $\mathcal{C}$ (resp., $\mathcal{C}_1$ and $\mathcal{C}_2$); i.e., the number of quorums in coterie $\mathcal{C}$ (resp., $\mathcal{C}$ and $\mathcal{C}$).

Theorem 7. If $\mathcal{C}_1$ and $\mathcal{C}_2$ are majority ND k-coteries, then

1. $\|\mathcal{C}\| = \|\mathcal{C}_1\| \times \|\mathcal{C}_2\|$, when $\mathcal{C} = \mathcal{C}_1 \otimes \mathcal{C}_2$, and
2. $\|\mathcal{C}\| \geq \|\mathcal{C}_2\| \times \binom{|\mathcal{P}_1|-1}{q-1}, q = \lceil \frac{|\mathcal{P}_1|+1}{k+1} \rceil$, when $\mathcal{C} = \mathcal{C}_1 \odot_x \mathcal{C}_2$. ∎

5 Conclusions

We have proposed a quorum based group k-mutex algorithm for open distributed environments in this paper. The algorithm consists of two main parts, i.e., the quorum-consensus and quorum-reconfiguration, each of which placed in different layers and work separately. The quorum consensus layer directly adopts a k-coterie based algorithm for group k-mutex in the static environments which is also proposed in this paper. Thus, its message complexity and quorum availability performances are the same as in the static environments.

References

1. Vidyasankar, K.: A simple group mutual ℓ-exclusion algorithm. Information Processing Letters **85** (2003) 79–85
2. Agrawal, D., Abbadi, A.E.: An efficient and fault-tolerant algorithm for distributed mutual exclusion. ACM Trans. Computer Systems **9** (1991) 1–20
3. Garcia-Molina, H., Barbara, D.: How to assign votes in a distributed system. Journal of the ACM **32** (1985) 841–860
4. Joung, Y.J.: Quorum-based algorithms for group mutual exclusion. IEEE Transaction on Parallel and Distributed Systems **14** (2003) 463–476
5. Schlichting, R., Schneider, F.: Fail-stop processors: an approach to designing fault-tolerant computing systems. ACM Trans. Computer Systems **1** (1983) 222–238
6. Lawi, A., Oda, K., Yoshida, T.: A simple quorum reconfiguration for open distributed environments. In: Proc. International Conference on Parallel and Distributed Systems (ICPADS). Volume II. (2005) 664–668
7. Fujita, S., Yamashita, M., Ae, T.: Distributed k-mutual exclusion problem and k-coteries. In: Proc. 2nd International Symposium on Algorithms, Lecture Notes in Computer Science 557. (1991) 22–31
8. Neilsen, M.L., Mizuno, M.: Coterie join algorithm. IEEE Trans. Parallel and Dist. Systems **3** (1992) 582–590
9. Jiang, J.R., Huang, S.T.: Obtaining nondominated k-coteries for fault-tolerant distributed k-mutual exlusion. In: Proc. International Conference on Parallel and Distributed Systems (ICPADS). (1994) 582–587
10. Maekawa, M.: A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems. ACM Trans. Computer Systems **3** (1985) 145–159