Quorum-Based Algorithms for Group Mutual Exclusion*

Yuh-Jzer Joung

Department of Information Management
National Taiwan University
Taipei, Taiwan
joung@ccms.ntu.edu.tw

Abstract. We propose a quorum system, which we referred to as the *surficial* quorum system, for group mutual exclusion. The surficial quorum system is geometrically evident and so is easy to construct. It also has a nice structure based on which a truly distributed algorithm for group mutual exclusion can be obtained, and processes' loads can be minimized. When used with Maekawa's algorithm, the surficial quorum system allows up to $\sqrt{\frac{2n}{m(m-1)}}$ processes to access a resource simultaneously, where n is the total number of processes, and m is the total number of groups. We also present two modifications of Maekawa's algorithm so that the number of processes that can access a resource at a time is not limited to the structure of the underlying quorum system, but to the number that the problem definition allows.

1 Introduction

Group mutual exclusion [16, 20, 4, 13] is a generalization of mutual exclusion that allows a resource to be shared by processes of the same group, but requires processes of different groups to use the resource in a mutually exclusive style. As an application of the problem, assume that data is stored in a CD jukebox where only one disk can be loaded for access at a time. Then when a disk is loaded, users that need data on this disk can concurrently access the disk, while users that need a different disk have to wait until no one is using the currently loaded disk. Group mutual exclusion differs from *l-exclusion* [9] (which is also a generalization of mutual exclusion that allows at most l processes to be in the critical section simultaneously) in that in the latter the conflict in accessing a resource is due to the number of processes that attempt to access the resource, while in the former the conflict is due to the "type" of processes (i.e., the group they belong to).

Solutions for group mutual exclusion in shared memory models have been proposed in [16, 20, 4, 13]. Here we consider message passing networks. Two message-passing solutions for group mutual exclusion have been proposed in [17]. Both are extensions of Ricart and Agrawala's algorithm for mutual exclusion [31]. Basically, they work as follows: a process wishing to enter a critical section (i.e., to use a shared resource) broadcasts a request to all processes in the system, and enters the critical section when all processes have acknowledged its request.

* Part of this research was done when the author was visiting Lab for Computer Science, Massachusetts Institute of Technology (1999-2000). Research supported in part by the National Science Council, Taipei, Taiwan, Grants NSC 89-2213-E-002-110.

Since all processes are involved in determining whether a process can enter the critical section, the algorithms cannot tolerate any single process failure.

In the literature, quorum systems have proven useful in coping with site failures and network partitions for both mutual exclusion (e.g., [11, 26, 2, 21, 8, 1, 30]), and l-exclusion (e.g., [19, 23, 24, 28]). In general, a quorum system (called a *coterie* [11]) consists of a set of quora, each of which is a set of processes. Quora are used to guard the critical section. To enter the critical section, a process must *acquire* a quorum; that is, to obtain permission from every member of the quorum. Suppose that a quorum member gives permission to only one process at a time. Then mutual exclusion can be guaranteed by requiring every two quora in a coterie to intersect, and l-exclusion can be guaranteed by requiring any collection of $l+1$ quora to contain at least two intersecting quora. A quorum usually involves only a subset of the processes in the system. So even if processes may fail or become unreachable due to network partitions, some process may still be able to enter the critical section so long as not all quora are hit (a quorum is *hit* if some of its members fails).

It is easy to see that coteries for l-exclusion cannot be used for group mutual exclusion because two conflicting processes may then both enter the critical section after obtaining permissions from members of two disjoint quora respectively. On the other hand, coteries for mutual exclusion can be used for group mutual exclusion, but it will result in a degenerate solution in which only one process can be in the critical section at a time.

In this paper we present a quorum system, which we refer to as the *surficial* quorum system, for group mutual exclusion. To our knowledge, this is the first quorum system for group mutual exclusion to appear in the literature. The surficial quorum system is geometrically evident and so is easy to construct. It also has a nice structure based on which a truly distributed algorithm for group mutual exclusion can be obtained, and processes' loads can be minimized. When used with Maekawa's algorithm [26], the surficial quorum system allows up to $\sqrt{\frac{2n}{m(m-1)}}$ processes to access a resource simultaneously, where n is the total number of processes, and m is the total number of groups. In contrast, only one process is allowed to access a resource at a time if an ordinary quorum system is used.

We also present a modification of Maekawa's algorithm so that the number of processes that can access a resource at a time is not limited to the structure of the underlying quorum system, but to the number that the problem definition allows. Thus, the modified algorithm can also use ordinary quorum systems to solve group mutual exclusion. Nevertheless, when used with our surficial quorum system, the message complexity of the modified algorithm is still a factor of $O(\sqrt{\frac{2n}{m(m-1)}})$ better than that used with an ordinary quorum system (of the same quorum size). Another modification that trades offs synchronization delay for message complexity is also presented in the paper.

The rest of the paper is organized as follows. Section 2 gives the problem definition for group mutual exclusion. Section 3 presents the surficial quorum system. Section 4 presents quorum-based algorithms for group mutual exclusion. Conclusions and future work are offered in Section 5.

2 The Group Mutual Exclusion Problem

We consider a system of n asynchronous processes $1, \ldots, n$, each of which cycles through the following three states, with NCS being the initial state:

- *NCS*: the process is outside CS (the *Critical Section*), and does not wish to enter CS.
- *trying*: the process wishes to enter CS, but has not yet entered CS.
- *CS*: the process is in CS.

The processes belong to m groups $1, \ldots, m$. To make the problem more general, we do not require groups to be disjoint. When a process may belong to more than one group, the process must identify a unique group to which it belongs when it wishes to enter CS. Since group membership is concerned only at the time a process wishes to enter CS (and at the time the process is in CS), when we say 'process i belongs to group j', we implicitly assume that process i has specified j as its group for entering CS.[1]

The problem is to design an algorithm for the system satisfying the following requirements:

mutual exclusion: At any given time, no two processes of different groups are in CS simultaneously.

lockout freedom: A process wishing to enter CS will eventually succeed.

Moreover, to avoid degenerate solutions and unnecessary synchronization, we are looking for algorithms that can facilitate "**concurrent entering**", meaning that if a group g of processes wish to enter CS and no other group of processes are interested in entering CS, then the processes in group g can concurrently *enter* CS [16, 20, 13].

3 A Quorum System for Group Mutual Exclusion

In this section we present a quorum system for group mutual exclusion. Let $P = \{1, \ldots, n\}$ be a set of nodes[2], which belong to m groups. An ***m-group quorum system*** $\mathfrak{S} = (C_1, \ldots, C_m)$ over P consists of m sets, where each $C_i \subseteq 2^P$ is a set of subsets of P satisfying the following conditions:

intersection: $\forall\, 1 \leq i, j \leq m, i \neq j, \forall\, Q_1 \in C_i, \forall\, Q_2 \in C_j : Q_1 \cap Q_2 \neq \emptyset$.
minimality: $\forall\, 1 \leq i \leq m, \forall\, Q_1, Q_2 \in C_i, Q_1 \neq Q_2 : Q_1 \not\subseteq Q_2$.

We call each C_i a ***cartel***, and each $Q \in C_i$ a ***quorum***.

Intuitively, $\mathfrak{S}$ can be used to solve group mutual exclusion as follow: each process i of group j, when attempting to enter CS, must acquire permission from every member in a quorum $Q \in C_j$. Upon exiting CS, process i returns the permission to the members of the quorum. Suppose a quorum member gives permission to only one process at a time. Then, by the intersection property, no two processes of different groups can be in CS simultaneously. The minimality property is used rather to enhance efficiency. As is easy to see, if $Q_1 \subseteq Q_2$, then a process that can obtain permission from every member of Q_2 can also obtain permission from every member of Q_1.

Recall that a quorum system over P for mutual exclusion is a set $C \subseteq 2^P$ of quora satisfying the following requirements:

[1] The problem is described in a more anthropomorphous setting as *Congenial Talking Philosophers* in [16].

[2] The terms *processes* and *nodes* will be used interchangeably throughout the paper. For a distinguishing purpose, however, we use "nodes" specifically to denote quorum members, and "processes" to denote group members.

intersection: $\forall\, Q_1, Q_2 \in C : Q_1 \cap Q_2 \neq \emptyset$.
minimality: $\forall\, Q_1, Q_2 \in C, Q_1 \neq Q_2 : Q_1 \not\subseteq Q_2$.

To distinguish quorum systems for mutual exclusion from group quorum systems, we refer to the former as ***ordinary*** quorum systems.

An ordinary quorum system C can be used as an m-group quorum system by a straightforward transformation $\mathfrak{T}_m$:

$$\mathfrak{T}_m(C) = (C, \ldots, C).$$

By the intersection property of C, all quora in a cartel of $\mathfrak{T}_m(C)$ are pairwise intersected. Note that, in general, quora in the same cartel of a group quorum system need not intersect.

We define the ***degree*** of a cartel C, denoted as $\mathsf{deg}(C)$, to be the maximum number of pairwise disjoint quora in C. The ***degree*** of a group quorum system $\mathfrak{S}$, denoted as $\mathsf{deg}(\mathfrak{S})$, is the minimum $\mathsf{deg}(C)$ among the cartels C in $\mathfrak{S}$. Clearly, if a node gives out its permission to at most one process at a time, then the number of processes (of the same group) that can be in CS simultaneously is limited to $\mathsf{deg}(C)$, where C is the cartel associated with the group. Moreover, a group quorum system of degree k immediately implies that every cartel contains at least an unhit quorum even if $k-1$ processes have failed. So high degree group quorum systems also provide a better protection against faults.

In the following we present an m-group quorum system $\mathfrak{S}_m = (C_1, \ldots, C_m)$ with degree $\sqrt{\frac{2n}{m(m-1)}}$. In addition, the quora in the system satisfy the following four extra conditions:

1. $\forall\, 1 \leq i, j \leq m : \left|C_i\right| = \left|C_j\right|$.
2. $\forall\, 1 \leq i, j \leq m, \forall\, Q_1 \in C_i, \forall\, Q_2 \in C_j : \left|Q_1\right| = \left|Q_2\right|$.
3. $\forall\, p, q \in P : \left|n_p\right| = \left|n_q\right|$, where n_p is the multiset $\{Q \mid \exists\, 1 \leq i \leq m : Q \in C_i$ and $p \in Q\}$, and similar for n_q. In other words, $\left|n_p\right|$ is the number of quora involving p.
4. $\forall\, 1 \leq i, j \leq m, i \neq j, \forall\, Q_1 \in C_i, \forall\, Q_2 \in C_j : \left|Q_1 \cap Q_2\right| = 1$.

Intuitively, the first condition ensures that each group has an equal chance in competing CS. The second condition ensures that the number of messages needed per entry to CS is independent of the quorum a process chooses. The third condition means that each node shares the same responsibility in the system. As argued by Maekawa [26], these three conditions are desirable for an algorithm to be truly distributed. The last condition simply minimizes the number of nodes that must be common to any two quora of different cartels, thereby reducing the size of a quorum.

Before presenting the detailed construction of $\mathfrak{S}_m$, we first provide some intuitions. It is easy to see that a 1-group quorum system $\mathfrak{S}_1$ satisfying the above conditions can be obtained as follows:

$$\mathfrak{S}_1 = \left(\{\{p\} \mid p \in P\}\right)$$

The quorum system can be viewed as a line consisting of n nodes, each of which corresponds to a process in P, where $n = |P|$. Each quorum then consists of exactly one node on the line, and the collection of the quora constitutes the only cartel in the system. See Figure 1, top. By extending this line to a two-dimensional plane, we can obtain a 2-group quorum system $\mathfrak{S}_2 = (C_1, C_2)$, where

each quorum in C_1 corresponds to the set of nodes in each row, while each quorum in C_2 corresponds to the set of nodes in each column. By taking one step further, we can construct a 3-group quorum system $\mathfrak{S}_3 = (C_1, C_2, C_3)$ by arranging nodes on the surface of a cube. Notice that a cube can be "wrapped up" by lines (strings) along three different dimensions. Lines along the same dimension are parallel to each other, while any two lines along different dimensions must intersect at two points. If we arrange the nodes on only three sides of the cube as shown in Figure 1, then every two lines along different dimensions intersect at exactly one point.

We can unfold the three sides of the cube on the plane as shown on the left of Figure 2. Each quorum in C_1 then corresponds to a vertical line across the first (the right most) column of squares. Each quorum in C_2 corresponds to a horizontal line across the top square, and a vertical line across the left square on the bottom. Finally, each quorum in C_3 corresponds to a horizontal line across the two squares on the bottom.

By appending another three squares to the bottom of the above pile of squares and extending the lines to these extra squares, we can construct $\mathfrak{S}_4$ as shown on the right of Figure 2. Each quorum in C_1 corresponds to a vertical line across the first column of squares. Each quorum in C_2 corresponds to a horizontal line across the square on the first level (starting from the top), and then a vertical line across the second column of squares. Each quorum in C_3 corresponds to a horizontal line across the squares on the second level, and then a vertical line across the third column of squares. Finally, each quorum in C_4 corresponds to a horizontal line across the squares on the third level. Notice that on the right staircase of Figure 2, the first level of squares constitutes $\mathfrak{S}_2$, and the first two levels of squares constitutes $\mathfrak{S}_3$.

This procedure can be extended to $\mathfrak{S}_m$. In general, each C_i in $\mathfrak{S}_m$ needs $m-1$ squares, each of which is to be shared with one of the other $m-1$ cartels so that the corresponding lines of the two cartels intersect at exactly one node on the square. Overall, there are $m(m-1)/2$ squares. Let k be the width of each square (i.e., the number of quora in each cartel). Then each square consists of k^2 nodes. So the total number of nodes on the $m(m-1)/2$ squares is $k^2 m(m-1)/2$. A simple way to map nodes on the squares to the processes in P is to let each node correspond to a unique process. In this case $k^2 m(m-1)/2 = n$, where $n = |P|$. So

$$k = \sqrt{\frac{2n}{m(m-1)}}, \quad m > 1.$$

So the quorum size is

$$(m-1) \cdot k = \sqrt{\frac{2n(m-1)}{m}}$$

and each cartel consists of $\sqrt{\frac{2n}{m(m-1)}}$ quora.

In the following we present the "staircase" construction of the surficial quorum system.

Inputs. $P \subseteq N$, $n, m \in N$, where $n = |P|$.
Outputs. An m-group quorum system $\mathfrak{S}_m = (C_1, \ldots, C_m)$ over P.
Assumption. $m > 1$ and $\sqrt{\frac{2n}{m(m-1)}} = k$ for some integer k.

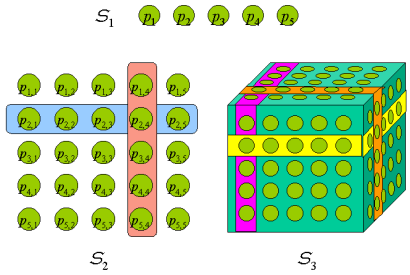


Fig. 1. A surficial quorum systems.

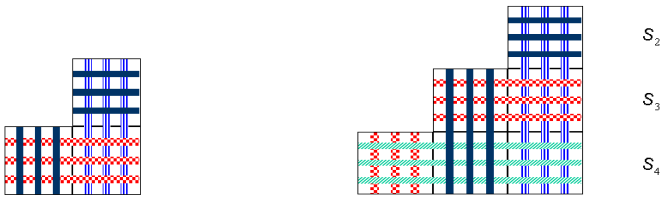


Fig. 2. The unfolded surficial quorum system for $\mathfrak{S}_3$ (left), and $\mathfrak{S}_4$ (right).

1. Partition P into $\frac{m(m-1)}{2}$ subsets, each of which consists of k^2 nodes. Let $P^{i,j}$ denote these subsets, $1 \leq i \leq m-1$, $i \leq j \leq m-1$.
   For each $P^{i,j}$, let $p^{i,j}_{r,s}$ denote the nodes in the set, where $1 \leq r, s \leq k$. (See Figure 3.)
2. For each cartel C_i in $\mathfrak{S}_m$, denote the quora in the cartel by $Q_{i,j}$, $1 \leq i \leq m$, $1 \leq j \leq k$. Then, $Q_{i,j}$ is defined by

   $$Q_{i,j} \;=\; \{\, p^{s,i-1}_{r,j} \,|\, 1 \leq s \leq i-1, 1 \leq r \leq k \,\} \bigcup \{\, p^{i,s}_{j,r} \,|\, i \leq s \leq m-1, 1 \leq r \leq k \,\}$$

   Note that when $i = 1$, the first set in the formula is empty, and when $i = m$, the second part is empty.

Remarks

Some comments on the surficial quorum system are given in order. First, the degree of $\mathfrak{S}_m$ is $\sqrt{\frac{2n}{m(m-1)}}$. In terms of degree, the construction is not optimal, as we can construct an m-group quorum system with degree $\sqrt{n}$ [18]. It can be seen that $\sqrt{n}$ is the upper bound as no m-group quorum system over a set of n nodes can have degree greater than $\sqrt{n}$ for any $m > 1$. (This is because every quorum in an m-group quorum system of degree k must contain at least k elements. So every cartel must involve at least $k \cdot k \leq n$ different nodes.) However, to reach such degree, n must be equal to some p^{2c}, where p is a prime and c is a positive integer. Besides, the construction is difficult to visualize as it works on an affine plane of order p^c. In contrast, the surficial quorum system is easy to visualize and so is easy to construct. Moreover, the surficial quorum system

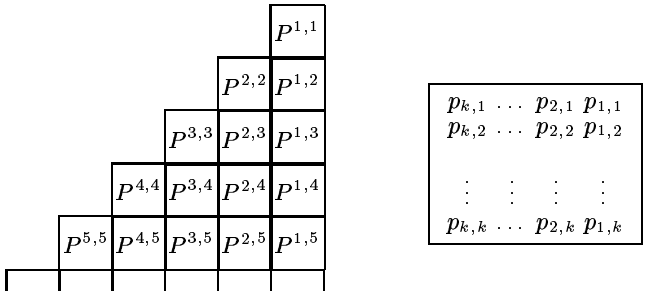


Fig. 3. Arrangement of nodes in $\mathfrak{S}_m$. On the left is the indices (superscripts) of squares, and on the right is the indices (subscripts) of nodes in each square.

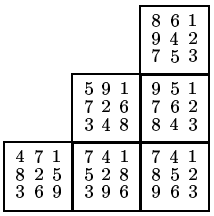
minimizes processes' loads by letting every node be included in at most two quora. Under this condition, the maximum degree an m-group quorum system can achieve is $\sqrt{\frac{2n}{m(m-1)}}$, which is exactly what $\mathfrak{S}_m$ has achieved.[3]

Second, in the construction we have chosen the mapping between nodes and processes to be one-to-one. The advantage of this mapping is that it is simple and geometrically evident. The quorum size may be reduced (and therefore the increase of $\mathsf{deg}(\mathfrak{S}_m)$) by choosing a more sophisticated mapping to allow nodes to map to the same process. For example, the mapping shown in Figure 4 for $m = 4$ and $n = 9$ results in a 4-group quorum system that is optimal in degree (i.e., with maximum possible degree).[4] However, finding a mapping that optimizes the degree for arbitrary n and m is considerably more difficult and remains a challenging future work.

Third, any nonempty mapping φ from the nodes to the processes results in an m-group quorum system, although it may not satisfy the four extra conditions discussed at the beginning of this section. For example, the mapping $\varphi(p_{r,s}^{i,j}) = q$ for any fixed $q \in P$, $1 \le i \le m-1, i \le j \le m-1, 1 \le r, s \le k$ results in a singleton-based m-group quorum system. Moreover, assume instead that $k = \sqrt{n}$. Let the processes be arranged as a grid so that $P = \{q_{r,s} \mid 1 \le r, s \le k\}$.

[3] To see this, let $\mathfrak{C} = (C_1, \ldots, C_m)$ be a group quorum system of degree k over an n-set. Let $P_{r,s}^{i,j}$ be the intersection of the rth quorum in C_i and the sth quorum in C_j, $i \neq j$. By definition, $P_{r,s}^{i,j} \neq \emptyset$. Since every node is included in at most two quora, $P_{r,s}^{i,j} \cap P_{r',s'}^{i',j'} = \emptyset$ if $\{i,j\} \neq \{i',j'\}$ (i.e., the two *unordered* pairs are not equal) or $(r,s) \neq (r',s')$ (i.e., the two *ordered* pairs are not equal). Given that $1 \le i \neq j \le m$ and that each cartel contains at least k quora, there are $\binom{m}{2}$ unordered pairs of i, j, and for each of them, at least k^2 ordered pairs of (r,s) that can constitute a $P_{r,s}^{i,j}$. So $\left|\bigcup_{i,j,r,s} P_{r,s}^{i,j}\right| \ge \binom{m}{2}k^2$. Since $\binom{m}{2}k^2 \le n$, we have $k \le \sqrt{\frac{2n}{m(m-1)}}$.

[4] Alternatively, the quorum system can be visualized as follows: take any one of the six squares. Then, any vertical line of three nodes constitutes a quorum, and the three vertical lines on the square constitute a cartel. Similarly, the three horizontal lines constitute a second cartel. The other two cartels correspond to the three "rounded" lines with slope 1 and -1, respectively. For example, the three "rounded" lines with slope 1 on the top square are: $\{3,6,9\}$, $\{5,2,8\}$, and $\{7,4,1\}$.



$\mathfrak{S} = (C_1, C_2, C_3, C_4)$ where
$C_1 = \{\{1,2,3\},\ \{4,5,6\},\ \{7,8,9\}\}$
$C_2 = \{\{1,6,8\},\ \{2,4,9\},\ \{3,5,7\}\}$
$C_3 = \{\{3,5,9\},\ \{2,6,7\},\ \{3,4,8\}\}$
$C_4 = \{\{1,4,7\},\ \{2,5,8\},\ \{3,6,9\}\}$

Fig. 4. A mapping between processes and nodes and the resulting 4-group quorum system.

Then the mapping $\varphi(p_{r,s}^{i,j}) = q_{r,s}$ results in a quorum system $\mathfrak{C} = (C_1, ..., C_m)$ such that each quorum in C_1 corresponds to a column in the grid, each quorum in C_m corresponds to a row in the grid, and each quorum in every other cartel C_i corresponds to a row and a column.[5] By experimenting different mappings and by adjusting the dimension of the squares (i.e., by replacing the squares with rectangles), we can fine-tune the quorum system to fit into different applications in which the demand of the shared resource by different groups of processes is off-balanced.

Finally, it is easy to see that $\mathfrak{S}_m$ is in general "dominated." Dominance between group quorum systems is defined as follows [18]:

Definition 1. *Let* $\mathfrak{C} = (C_1, \ldots, C_m)$ *and* $\mathfrak{D} = (D_1, \ldots, D_m)$ *be two m-group quorum systems over* P. *Then* $\mathfrak{C}$ *dominates* $\mathfrak{D}$ *if*

1. $\mathfrak{C} \neq \mathfrak{D}$,
2. $\forall Q \in D_i, 1 \leq i \leq m, \exists R \in C_i : R \subseteq Q$.

$\mathfrak{D}$ *is* ***nondominated*** *if there is no* $\mathfrak{C}$ *such that* $\mathfrak{C}$ *dominates* $\mathfrak{D}$.

To illustrate dominance, the group quorum system $\mathfrak{C} = \big(\{\{1,2\},\{3,4\}\}, \{\{1,3\}, \{2,4\}\}\big)$ over $P = \{1,2,3,4\}$ is dominated by $\mathfrak{D} = \big(\{\{1,2\},\{3,4\},\{1,4\},\{2,3\}\}, \{\{1,3\},\{2,4\}\}\big)$. By a simple enumeration, it can be seen than $\mathfrak{D}$ is nondominated.

Intuitively, a group quorum system C *dominates* D if whenever a quorum of a cartel in D can survive some failures, then some quorum in the corresponding cartel of C can certainly survive as well. Thus, C is said to be superior to D because C provides more protection against failures.

"Fully distributedness" and "nondominance" appear to be two conflicting notions, as for example, the "fully distributed" ordinary quorum system proposed by Maekawa [26] is also dominated [10, 29]. However, the construction of $\mathfrak{S}_m = (C_1, \ldots, C_m)$ is such that every quorum Q in the cartels is a minimal set that intersects every other quorum in a different cartel. As proved in [18], this property implies that $\mathfrak{S}_m$ can be extended to a nondominated group quorum system $\mathfrak{U} = (D_1, \ldots, D_m)$ such that $C_i \subseteq D_i$ for all $1 \leq i \leq m$. In other words,

[5] Note that because in the construction a horizontal line of nodes is "connected" by a unique vertical line of nodes, not any union of a row and a column in the grid represents a quorum in C_i. However, by relaxing the "connection" condition, we can obtain a C_i that is composed of any union of a row and a column in the grid. Note further that C_1 and C_m have degree $\sqrt{n}$, while the other cartels have degree 1.

we can build upon $\mathfrak{S}_m$ a nondominated group quorum system $\mathfrak{U}$ such that $\mathfrak{U}$ "contains" $\mathfrak{S}_m$. An important meaning of this "containing" relation is that: $\mathfrak{S}_m$ can be used to realize a truly distributed algorithm when failures do not occur, while $\mathfrak{U}$ can be used to "backup" $\mathfrak{S}_m$ when failures do occur to increase fault tolerance.

4 Quorum-Based Algorithms for Group Mutual Exclusion

In this section we present two algorithms that use quorum systems to solve group mutual exclusion. The network is assumed to be complete and the communication channel between each pair of processes is reliable and FIFO. We begin with Maekawa's algorithm [26]. Then we discuss some variations involving trade-offs between concurrency and message complexity, and between concurrency and synchronization delay.

4.1 The Basic Framework

Most quorum-based algorithms for mutual exclusion are based on Maekawa's algorithm [26], which works as follows:

1. A process i wishing to enter CS sends a request message to each member of a quorum Q, and enters CS only after it has received a permission message from every member of Q.
   Upon leaving CS, a process sends a release message to every member of Q to release its permission.
2. A node gives away one permission at a time. So upon receipt of a request by i, a node j checks if it has given away its permission.
   (a) If j has given a permission message to another process k, and has not yet received a release message from k, then some arbitration mechanism is used to determine whether to let i wait, or to let i preempt k's possession of the permission.
   (b) Otherwise, j sends a permission message to i.

In general, request messages by process i to the members of Q are sent simultaneously so as to minimize the *synchronization delay*, which is the delay from the time a process invokes a mutual exclusion request until the time it enters CS. The delay is measured in terms of message transmission time. It is clear that the minimum synchronization delay is 2 if request messages are sent simultaneously. However, due to the asynchrony of the system, a process may hold a permission while waiting for another process to release a permission. This in turn may incur a deadlock.

Maekawa's algorithm handles deadlocks by requiring low-priority processes to yield permissions to high-priority processes. Priorities are usually implemented by Lamport's logical timestamps [25]. The smaller the timestamp of a request, the higher the priority of the request. Specifically, if a node i receives a request by j after giving a permission to k and j's priority is higher than k's, then i issues an inquiry message to k. Process k then releases its permission to i (by sending a release message) if it determines that it cannot successfully acquire permissions from all members of the quorum it chooses. After receiving the release message, node i gives its permission to j by sending j a grant message. When j exits CS and releases i's permission, i then returns the permission to k (presumably no

other process with a priority higher than k's is also waiting for i's permission). So Maekawa's algorithm needs $3c$ to $6c$ messages per entry to CS, where c is the (maximum) size of a quorum.

Maekawa's algorithm works straightforwardly for group mutual exclusion: Let $\mathfrak{C} = (C_1, \ldots, C_m)$ be an m-group quorum system over P. A process $i \in P$ that wishes to enter CS as a member of group j chooses a quorum from the cartel C_j, and enters CS only when it has obtained a permission from every member of the quorum. By the mutual exclusion property of $\mathfrak{C}$ and by the conflict resolution scheme used in the algorithms, mutual exclusion and lockout freedom are easily guaranteed.

4.2 A Tradeoff Between Concurrency and Message Complexity

In Maekawa's algorithm, since a node gives permission to only one process at a time, the maximum number of processes of a group that can be in CS simultaneously is limited to the degree of the cartel associated with the group. So no concurrency is offered using group quorum systems $\mathfrak{T}_m(C)$ derived from ordinary quorum systems C (which have degree one).

For group quorum systems with degree greater than one, they may still not be able to offer a satisfactory degree of concurrency. This is because the size of a group can be greater than $\sqrt{|P|}$. However, as discussed in Section 3, no m-group quorum system over P can have degree more than $\sqrt{|P|}$, unless $m = 1$.

To overcome this, we modify Maekawa's algorithm so that a node can give permission to more than one process. So even if quora in the same cartel may intersect, two or more processes may still enter CS simultaneously. The rule for a node k to determine whether to grant i's request for permission is as follows:

> k grants i's request so long as there is no *conflict*—i.e., no other process of a different group is also requesting/possessing k's permission. Otherwise, conflicts are resolved as follows: a process i of group g yields k's permission to another process j of group h if j has priority higher than all processes of group g that currently request/possess j's permissions.

Note that because of the rule, requests are not processed strictly in First-Come-First-Served (FCFS) order. In particular, a node k may grant i's request even if some other j of a different group is already waiting for k's permission (regardless of whether j's priority is higher than i or not.) The reason for this is to increase system performance. We shall explain this in more detail in the following section.

We refer to the algorithm as Maekawa_M. Because the algorithm is fairly easy to design, we omit the detailed code of the algorithm in this extended abstract.

4.3 A Tradeoff Between Concurrency and Synchronization Delay

In the above algorithm, after a node i has given permission to k processes, i may have to withdraw its permission if it receives a higher priority request from a different group. Withdrawing a permission from a process results in three messages: an inquiry message, a relinquish message, and eventually the return of the permission to the process. So in the worst case, a request to i may generate $3k$ messages. So the maximum number of messages per entry to CS is $3c + 3c \cdot r$, where c is the quorum size, and r is the maximum number of permissions a node may give away at a time. To facilitate a maximum concurrency while minimizing message complexity, for each cartel C, r should be limited to $s/\mathsf{deg}(C)$, where s is the size of the group that uses C. In this case, an entry to CS requires at

```
A.1  *[ wish to enter CS as a member of group g ⟶
A.2      state := trying;
A.3      randomly select a quorum Q from C_g;
A.4      send REQUEST(i, g, Q) to first(Q);
B.1  □ receive GRANT(j) ⟶
B.2      state := CS; /* acquires quorum Q */
C.1  □ exit CS ⟶
C.2      state := NCS;
C.3      for j ∈ Q do send RELEASE(i) to j;
C.4  ]
```

Variables:

- *state*: the state of process i.
- Q: the quorum i selects. We assume the following three functions on quora:
  - $first(Q)$: return the smallest id in Q.
  - $next(k, Q)$: return the smallest id in Q that is greater than k.
  - $last(Q)$: return the largest id in Q.
- C_g: the cartel associated with group g.

Messages:

- REQUEST(i, g, Q): a request by i to obtain the recipient's permission to enter CS as a member of group g. Q is the (id of the) quorum i chooses.
- GRANT(j): a permission given by node j to the recipient. In particular, j is the last node in Q.
- RELEASE(i): a message by i to release the recipient's permission.

Fig. 5. Algorithm Maekawa_S executed by process i.

most $3c + 3c \cdot s/\mathsf{deg}(C)$. Given that s is determined by the problem definition, the message complexity is roughly in proportion to $c/\mathsf{deg}(C)$. So the higher the degree of the underlying group quorum system, the lower the message complexity.

If message complexity needs to be bounded in $O(c)$, deadlocks must be resolved in a different way. A well-known technique in resource allocation is to let each process request permissions from quorum members in some fixed order [14], say, with increasing node IDs. So if a process i is waiting for j's permission, then every permission i holds must be from some f such that $f < j$. Moreover, every process k that currently holds j's permission has either obtained permissions from all members of its quorum, or is waiting for a permission from some h such that $h > j$. So deadlocks are not possible because there is no circular waiting.

Note that the above deadlock free argument does not depend on how many permissions a node may give out at a time. That is, a node can still give out multiple permissions. The number of messages required per entry to CS is $3c$, and the minimum synchronization delay is $2c$. The message complexity and the minimum synchronization delay can be reduced further to $2c + 1$ and $c + 1$, respectively, by letting quorum members circulate request messages. The complete code of the algorithm is given in Figures 5 and 6. We refer to the algorithm as Maekawa_S. For ease of understanding, the algorithm is presented as two CSP-like repetitive commands consisting of guarded commands [15]: Figure 5 describes the behavior of a process that acts as a group member, and Figure 6 describes the behavior of a node that acts as a quorum member. If one wishes, the two repetitive commands can be combined into a single one.

```
D.1   *[ receive REQUEST(i, g, Q) ⟶
D.2      if grant_ps = ∅ ⋁ (g = current_group ⋀ reference ≠ ⊥ ⋀ i ∉ grant_ps) then [
           /* grant the request */
D.3        if j = last(Q) then send GRANT(j) to i;
D.4        else send REQUEST(i, g, Q) to next(j, Q);
D.5        if grant_ps = ∅ then [
D.6          current_group := g;
D.7          reference := i; ]
D.8        grant_ps := grant_ps + {i}; ]
D.9      else if i ∈ grant_ps /* i's request arrives before its previous release message */
D.10       early_requests := early_requests + {⟨i, g, Q⟩};
D.11     else request_ps := request_ps + {⟨i, g, Q⟩}; ]
E.1   □ receive RELEASE(i) ⟶
E.2      grant_ps := grant_ps − {i};
E.3      if reference = i then
E.4        if grant_ps ≠ ∅ ∧ request_ps = ∅ then /* choose a new reference */
E.5          reference := k for some arbitrary k ∈ grant_ps;
E.6        else reference := ⊥;
E.7      if grant_ps = ∅ ⋀ request_ps ≠ ∅ then [
           /* grant the earliest request in the queue, as well as the requests from
           the same group of processes */
E.8        ⟨k, h, R⟩ := first(request_ps);
E.9        reference := k;
E.10       current_group := h;
E.11       for ⟨k', h', R'⟩ ∈ request_ps, h = h', do [
E.12         if j = last(Q) then send GRANT(j) to k';
E.13           else send REQUEST(k', h', R') to next(j, R');
E.14         request_ps := request_ps − {⟨k', h', R'⟩};
E.15         grant_ps := grant_ps + {k'}; ]
E.16     if ⟨i, g, S⟩ ∈ early_requests for some g and S, then [
           /* process i's early request */
E.17       send REQUEST(i, g, S) to j;
E.18       early_requests := early_requests − {⟨i, g, S⟩}; ]
E.19  ]
```

Variables:

- *request_ps*: queue of requests received by j. The requests are represented as tuples $\langle i, g, Q\rangle$, where i is the requester, g is the group of the requester, and Q is the quorum i chooses. The queue is initialized to $\emptyset$. Requests in the queue are ordered by the time they are inserted into the queue. Function *first*(*request_ps*) returns the earliest request in the queue.
- *early_requests*: queue of "early" requests received by j. A request $\langle i, g, Q\rangle$ is said "early" and is temporarily stored in *early_requests* if i has released j's permission before it issued the request, but the request arrives at j before the release message.
- *grant_ps*: set of processes to which j has given a permission. It is initialized to $\emptyset$.
- *current_group*: the group of the processes to which j has given permissions. It is initialized to $\perp$.
- *reference*: a reference process used to determine whether subsequent processes of *current_group* can enter CS.

Fig. 6. Algorithm Maekawa_S executed by node j.

Although we have assumed FIFO communication channels, in the algorithm a node j may receive a process i's request before it receives i's release message for i's previous request (lines D.9-10). This occurs because request messages hop through quorum members. So when i issues a release message msg_1 to j and then issues a new request msg_2, msg_2 may arrive at j (indirectly through members of a different quorum) before msg_1 does. To simplify the algorithm, j defers the process of msg_2 until it has received msg_1 (lines E.16-18).

Like Maekawa_M, requests are not processed strictly in FCFS order. Instead, when a node j receives a request from a process i of group g, if j has no outstanding permission, then j sends i a permission and chooses i as a reference (line D.7). A reference process is used such that subsequent requests from the same group are also granted so long as the reference remains in CS. When a reference process leaves CS, if no other process of a different group is waiting for j's permission, then a new reference is chosen from those processes that currently hold j's permissions (lines E.3-6). Otherwise, the reference is reset to $\perp$, meaning that the "door" to CS for the group is closed to yield the opportunity to another group.

There are two reasons for choosing this "entry policy". First, by the mutual exclusion property, while some reference process i is in CS, no other group of processes can be in CS. So maximal resource utilization can be achieved by allowing more processes of the same group to share CS with i, regardless of whether some other group of processes are waiting for CS or not. Furthermore, because while i is in CS, some fast process may enter and exit CS any number of times, the algorithm facilitates an unbounded degree of concurrency [16]. Note that lockout freedom can still be guaranteed because a reference process will eventually leave CS and close the "door" to CS for its group.

The other reason is to minimize the number of "context switches" [17]. (A *context switch* occurs when the next entry to CS is by a different group of process.) As analyzed in [17], in group mutual exclusion requests to CS cannot be processed in a strictly FCFS order, or else the system could degenerate to the case in which nearly only one process can be in CS at a time when m is large. As can be seen, this phenomenon will also appear in both Maekawa_M and Maekawa_S. So in both algorithms we allow some late requests to "jump over" existing requests to allow more processes to concurrently enter CS. Due to the space limitation, more detailed performance analysis will be presented in the full paper.

4.4 Correctness of the Algorithms

Maekawa's algorithm has been well studied in [26, 32, 33, 6, 7]. It is easy to see that the algorithm can be directly adapted to group mutual exclusion using group quorum systems. For the two modified algorithms Maekawa_M and Maekawa_S we have presented, their correctness can also be easily seen. So to save space, we will leave the formal analysis in the full paper.

5 Conclusions and Future Work

We have presented a quorum system, the surficial quorum system, for group mutual exclusion. The surficial quorum system generalizes existing quorum systems for mutual exclusion in that quora for processes of the same group need not intersect with one another. This generalization allows processes to acquire quora simultaneously, and so allows them to enter critical section concurrently. The surficial quorum system has a very simple geometrical structure, based on

which a truly distributed algorithm for group mutual exclusion can be obtained, and based on which processes' loads can be minimized.

The surficial quorum system has degree $\sqrt{\frac{2n}{m(m-1)}}$, where n is the total number of processes, and m is the total number of groups. So when used with Maekawa's algorithm, it allows a maximum of $\sqrt{\frac{2n}{m(m-1)}}$ processes to be in the critical section simultaneously. The message complexity per entry to the critical section is $6 \cdot \sqrt{\frac{2n(m-1)}{m}}$. Furthermore, it can tolerate up to $\sqrt{\frac{2n}{m(m-1)}} - 1$ process failures. For comparison, the two message-passing algorithms RA1 and RA2 presented in [17] have message complexity $2n$ and $3n$, respectively, and both allow all group members to be in the critical section simultaneously. However, they cannot tolerate any single process failure. In terms of minimum synchronization delay, all three algorithms have the same measure—2.

As we have noted earlier, the degree of a group quorum system is bounded by $\sqrt{n}$. So Maekawa's algorithm must be generalized if group size is greater than $\sqrt{n}$ and we wish to allow all group members to be in the critical section simultaneously. Two generalizations Maekawa_M and Maekawa_S were presented in the paper. Both allow all group members to be in the critical section simultaneously, regardless of the degree of the underlying quorum systems. Maekawa_M preserves Maekawa's minimum synchronization delay, but needs $3c + 3c \cdot s/d$ messages per entry to the critical section (when a node gives away bounded number of permissions), where c is the quorum size, s is the group size, and d is the degree of the underlying group quorum system. So when used with the surficial quorum system, the message complexity is $O(n \cdot \min\{m, n\})$. The other algorithm Maekawa_S trades off minimum synchronization delay for message complexity. It reduces the message complexity to $2c + 1$, but needs a minimum synchronization delay of $c + 1$.

There is a considerable literature on quorum systems. Many structures have been explored to construct ordinary quorum systems, including *finite projective planes* [26], *weighted voting* [12, 11, 3], *grids* [22, 8, 1], *trees* [2], *wheels* [27], *walls* [30], and *planar graphs* [5]. The structure we used in the surficial quorum system can be viewed as a "multidimensional" grid. For future work, it is interesting to investigate how the other structures can be used for group quorum systems.

Acknowledgments. I would like to thank Nancy Lynch for giving me a valuable opportunity to visit her group, and to discuss this research with her, as well as with the other members of the TDS group, including Idit Keidar, Alex Shvartsman, and Victor Luchangco. I would also like to thank Michel Raynal for stimulating this research, and the anonymous referees for their useful comments.

References

1. D. Agrawal, Ö. Eğecioğlu, and A. El Abbadi. Billard quorums on the grid. *IPL*, 64(1):9–16, 14 October 1997.
2. D. Agrawal and A. El Abbadi. An efficient and fault-tolerant solution for distributed mutual exclusion. *ACM TOCS*, 9(1):1–20, February 1991.
3. M. Ahamad and M. H. Ammar. Multidimensional voting. *ACM TOCS*, 9(4):399–431, November 1991.
4. K. Alagarsamy and K. Vidyasankar. Elegant solutions for group mutual exclusion problem. Technical report, Dept. of Computer Science, Memorial University of Newfoundland, Canada, 1999.

5. R. A. Bazzi. Planar quorums. *TCS*, 243(1–2):243–268, July 2000.
6. Y.-I. Chang. A correct $O(\sqrt{N})$ distributed mutual exclusion algorithm. In *Proc. 5th Int'l. Conf. on Parallel and Distributed Computing and Systems*, pages 56–61, 1992.
7. Y.-I. Chang. Notes on Maekawa's $O(\sqrt{N})$ distributed mutual exclusion algorithm. In *Proc. SPDP '93*, pages 352–359, 1994.
8. S. Y. Cheung, M. H. Ammar, and M. Ahamad. The grid protocol: A high performance scheme for maintaining replicated data. *IEEE TKDE*, 4(6):582–59, December 1992.
9. M. J. Fischer, N. A. Lynch, J. E. Burns, and A. Borodin. Resource allocation with immunity to limited process failure (preliminary report). In *Proc. 20th FOCS*, pages 234–254, 1979.
10. W. C. Ada Fu. *Enhancing concurrency and availability for database systems.* PhD thesis, Simon Fraser University, Burnaby, British Columbia, Canada, 1990.
11. H. Garcia-Molina and D. Barbara. How to assign votes in a distributed system. *JACM*, 32(4):841–860, October 1985.
12. David K. Gifford. Weighted voting for replicated data. In *Proc. 7th ACM SOSP*, pages 150–162, 1979.
13. V. Hadzilacos. A note on group mutual exclusion. In *Proc. 20th PODC*, 2001.
14. J. W. Havender. Avoiding deadlock in multiasking systems. *IBM Systems Journal*, 7(2):74–84, 1968.
15. C. A. R. Hoare. Communicating sequential processes. *CACM*, 21(8):666–677, August 1978.
16. Y.-J. Joung. Asynchronous group mutual exclusion (extended abstract). In *Proc. 17th PODC*, pages 51–60, 1998. Full paper in *Distributed Computing*, 13(4):189-206, 2000.
17. Y.-J. Joung. The congenial talking philosophers problem in computer networks (extended abstract). In *Proc, 13th DISC*, LNCS 1693, pages 195–209, 1999.
18. Y.-J. Joung. On generalized quorum systems. Technical report, Department of Information Management, National Taiwan University, Taipei, Taiwan, 2000.
19. H. Kakugawa, S. Fujita, M. Yamashita, and T. Ae. Availability of k-coterie. *IEEE Trans. on Computers*, 42(5):553–558, May 1993.
20. P. Keane and M. Moir. A simple local-spin group mutual exclusion algorithm. In *Proc. 18th PODC*, pages 23–32, 1999.
21. A. Kumar. Hierarchical quorum consensus: A new algorithm for managing replicated data. *IEEE Trans. on Computers*, 40(9):996–1004, September 1991.
22. A. Kumar and S. Y. Cheung. A high availability $\sqrt{N}$ hierarchical grid algorithm for replicated data. *IPL*, 40(6):311–316, 30 December 1991.
23. Y.-C. Kuo and S.-T. Huang. A simple scheme to construct k-coteries with $O(\sqrt{N})$ uniform quorum sizes. *IPL*, 59(1):31–36, 8 July 1996.
24. Y.-C. Kuo and S.-T. Huang. A geometric approach for constructing coteries and k-coteries. *IEEE TPDS*, 8(4):402–411, April 1997.
25. L. Lamport. Time, clocks and the ordering of events in a distributed system. *CACM*, 21(7):558–565, July 1978.
26. M. Maekawa. A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems. *ACM TOCS*, 3(2):145–159, May 1985.
27. Y. Marcus and D. Peleg. Construction methods for quorum systems. Technical Report CS92-33, The Weizmann Institute of Science, Rehovot, Israel, 1992.
28. M. L. Neilsen. Properties of nondominated K-coteries. *The J. of Systems and Software*, 37(1):91–96, April 1997.
29. D. Peleg and A. Wool. The availability of quorum systems. *I&C*, 123(2):210–223, December 1995.
30. D. Peleg and A. Wool. Crumbling walls: A class of practical and efficient quorum systems. *DC*, 10(2):87–97, 1997.
31. G. Ricart and A. K. Agrawala. An optimal algorithm for mutual exclusion in computer networks. *CACM*, 24(1):9–17, January 1981.
32. B. A. Sanders. The information structure of distributed mutual exclusion algorithms. *ACM TOCS*, 5(3):284–299, August 1987.
33. M. Singhal. A class of deadlock-free Maekawa-type algorithms for mutual exclusion in distributed systems. *DC*, 4(3):131–138, 1991.