An $O(N^{M/(M+1)})$ distributed algorithm for the k-out of-M resources allocation problem*

Roberto Baldoni

Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza", via Salaria 113 - I-00198, Roma - Italy
e.mail: baldoni@disparcs.dis.uniroma1.it

Abstract

This paper presents a permission-based algorithm to solve the problem of M identical resources shared among N processes in a distributed system. We prove that the number of messages exchanged necessary for a process to acquire k resources is $O(N^{M/(M+1)})$. This result has been obtained (i) investigating the concept of arbiter of conflicting processes and (ii) extending conditions that permit conflict detection and resolution in a system of N competing processes to M shared resources. We show that for M=1 we get Maekawa's algorithm. However, we will also show that Maekawa's results, being based on finite projective plane geometry, do not apply for M>1.

Keywords: k-out of-M resources allocation, distributed mutual exclusion, distributed synchronization.

1. Introduction

The k-out of-M resources allocation problem is a paradigm that includes as particular cases the mutual exclusion [2, 3, 8, 9, 10, 11] and the multiple entries to a critical section [5,12]. The first distributed solution to the k-out of-M problem has been proposed by Raynal [6].

The basic k-out of-M problem can be described as follows [6]: in a distributed system there are M identical resources shared among N processes. At any given time each resource must be used by at most one process, and each process may be allocated any number of resources k ($1 \leq k \leq M$). A process requests resources all at once and, to avoid deadlock, the process is blocked until it gets all of its requested resources. After that, the process starts using the k resources and then releases them all at once. A conflict arises when a process tries to allocate a number of resources greater than those available at that given time.

A k-out of-M distributed algorithm works correctly iff two properties are satisfied:

- **Safety**: *at most M resources may be used at any given time and each resource must be assigned to only one process .*
- **Fairness:** any *request must be granted in a finite time.* [1]

Safety is obtained by ensuring that the following inequality always holds

$$\sum_{i=1}^{N} k_i \leq M \qquad (1)$$

where k_i is the number of resources used by process i. A key point to obtaining *safety* is to detect all conflicts that may arise and resolve them preserving (1). *Safety* reduces to *mutual exclusion* when M is equal to 1 [6].

Fairness is obtained not only by ensuring the detection and resolution of all conflicts, but also by ensuring that conflicts are not always resolved against some particular processes.

In conclusion, to verify if both properties hold we must answer the following questions:

1. which condition permits each conflict to be detected and resolved in a system with M identical resources shared among N processes?
2. which is the right discipline (i.e., policy) for fair conflict resolution among processes in a distributed system?

An exhaustive answer to Question 2 has been given by Chandy and Misra [1] who showed that conflict resolution based on timestamp [2] is fair. Therefore, we adopt the timestamp method. ie., every request is labeled

* This work was supported in part by the *Consiglio Nazionale delle Ricerche* under grant 92.02294.CT12.

with a timestamp and served accordingly. To ensure a total order of requests, timestamp ties are broken by ordering requests according to their process identifiers. Whenever the order is broken a *recovery action* is undertaken to resume a correct serialization in order to avoid deadlocks.

In the case of M=1 the answer to Question 1 has been given by Maekawa [3] using the *finite projective plane theory* [4]. Nevertheless, we will show that this theory does not apply if M is greater than one. Hence, we are going to answer to Question 1 investigating the concept of *arbiter* [3] of conflicting processes. This study shows that there exists a combinatorial proof which yields a general condition of *conflict detection* that encompasses that of Meakawa [3].

To solve the *k-out* of-*M* resources allocation problem, Raynal [6] proposed a very interesting permission-based algorithm [7] based on that of Ricart-Agrawala [9]. However, in doing so, his algorithm exchanges $O(N)$ messages to acquire k resources. In fact, in [6] each request and each release of resource must be sent to *all* processes in the system. An important feature of Raynal's algorithm is that it allows *local recovery* actions. i.e., recovery without sending messages.

In this paper we propose an algorithm that requires $O(N^{M/(M+1)})$ messages per *critical section* (i.e., source code in which a process performs operations with the shared resource) entry. If M is equal to one the complexity of our algorithm is equal to that of Maekawa [3]. Moreover, we will give the condition that permits local recovery actions in our algorithm.

The rest of this paper is structured as follows: In Section 2 Maekawa's approach and the finite projective planes are briefly explained. The arbiter set concept is introduced in Section 3. The *k*-out of-*M* algorithm and the inadequacy of Maekawa's approach to manage more than one resource are shown in Section 4. *Safety* and *fairness* for the proposed algorithm are proved in Section 5 and 6 respectively. In Section 7 we compute the complexity of the *k*-out of-*M* algorithm. Finally, in Section 8 the number of messages exchanged per critical section entry is discussed and compared with that of Raynal.

2. Maekawa's Algorithm

We assume the following assumptions on the communication network:

- *each process is connected to each other by means of reliable channels*;
- *communication is asynchronous* and *transmission times over each channel are unpredictable but finite*;
- *messages over each channel are delivered in FIFO order.*

Let U={1,..,N} be a set of concurrent processes, in which the term *concurrent processes* may also refer to computers in a network. In Maekawa's algorithm [3] M is equal to one and a request set (R) is associated with each concurrent process. These request sets verify the following conditions:

(i) $i \in R_i \qquad \forall i \in U$

(ii) $R_i \cap R_j \neq \emptyset \qquad \forall i, j \in U$ ***and*** $i \neq j$

The skeleton code executed by a process i to enter its critical section is the following: i sends a REQUEST message to each member of its R_i with an attached timestamp, waits for a GRANT message from each member of R_i, and enters its critical section. Upon exiting, i sends a RELEASE message to each member of R_i .

Maekawa claims that the problem of computing the *minimal* cardinality of request sets that satisfy Conditions (i) and (ii) is equivalent to considering a finite projective plane of N points whose order is K-1. *In that plane, lines are equivalent to request sets and points to processes.* A finite projective plane satisfies the following axioms [4]:

Axiom 1. *There is one and only line containing two points.*

Axiom 2. *There is one and only point common to two distinct lines.*

Axiom 3. *There exist four lines, no three of which go through the same point.*

Moreover, among other things, a finite projective plane of order K-1 satisfies the following properties [4]:

1. Each line contains K points,

2. Each point is on K lines,

3. There are exactly $((K-1)^2+(K-1)+1)$ points and lines in a plane.

The latter property implies that

$$N=K(K-1)+1 \qquad (2)$$

Then, we see that the *minimal* cardinality of a request set K is $O(\sqrt{N})$. A *minimal* configuration of the request sets obviously minimizes, in turn, the number of messages exchanged per critical section entry.

3. The Arbiter Set

In Maekawa's algorithm a member of $R_i \cap R_j$ is called "arbiter" of the requests of i and j. The arbiter *detects* a conflict between i and j and *resolves* it in order to ensure:

- *safety* (i.e., only one request is granted by the arbiter at a time) and,
- *fair conflict resolution* (conflicts are resolved by ordering requests by *timestamp*).

Using the arbiter concept, Condition (ii) of Maekawa's algorithm may be reformulated as follows:

(ii') ***there exists at least one arbiter*** $\forall i, j \in U$ **and** $i \neq j$

Let us introduce an *arbiter set* (A) associated with each process. An arbiter set is deduced by request sets, as follows

$A_i = \{ h \mid i \in R_h \}$

Hence, in Maekawa's algorithm, process *i arbitrates* (i.e., detects and resolves) all distinct pairs of processes that can be formed with A_i 's members.

4. The *k*-out of-*M* Algorithm

In figure 2 the activities of process *i* are shown. Let Λ be a set of requests, that represents, for an arbiter process, the best knowledge about the resources allocation. Each member of Λ includes the *process name*, the *number of required resources* and the *timestamp* of the request.

An arbiter *y* inserts *l*'s request in Λ_y and sends a GRANT message to a process *l* that requires *k* resources iff

$$\sum_{i \in \Lambda_y} resources(\Lambda_y[i]) + k \leq M \qquad (3)$$

Otherwise there is a conflict and the request will be inserted in a waiting queue (Q_y) ordered by timestamp.

Then, at any given time the following condition holds in each arbiter y

(a) $(ts_i < ts_j)$ ***or*** $(ts_i = ts_j$ ***and*** $i < j)$ $\forall i \in \Lambda_y, \forall j \in Q_y$

Upon receiving a GRANT message process *l* set $GRANT_l[y]$ to TRUE. Hence, according to (a) in each *requesting* process of A_y we have

(b) $GRANT_i[y]$=TRUE $\forall i \in \Lambda_y$

(c) $GRANT_j[y]$=FALSE $\forall j \in Q_y$

When all components of GRANT array are TRUE, *l* enters its critical section and upon exiting it sends a RELEASE message to each member of R_l.

Upon receiving the RELEASE message, *y* deletes *l*'request from Λ_y and the request on top of Q_y will enter Λ_y iff (3) is verified, where, in this case, k is the number of resources required by the process on top of Q_y.

When *y* receives a request that conflicts and has a lower timestamp than some requests in Λ_y a *recovery action*

R1={1, 2, 3}
R2={2, 4, 6}
R3={3, 5, 6}
R4={1, 4, 5}
R5={2, 5, 7}
R6={1, 6, 7}
R7={3, 4, 7}

Figure 1. Request sets in case of 7 processes

must be carried out since Condition (a) is no longer verified. i.e.,

$$\exists i \in Q_y, \exists l \in \Lambda_y : (ts_i < ts_l)$$ ***or*** $(ts_i = ts_l$ ***and*** $i < l)$

The *recovery action*, works as follows: an INQUIRE message is sent by the arbiter *y* to process *l* and waits for a RELEASE or YIELD message from *l*.

Upon receiving the INQUIRE message, if *l* is outside its critical section but is *requesting* it sends back a YIELD message to *y* and returns to wait for a new GRANT message of *y*. In case *l* is in its critical section, after a finite time, it will send a RELEASE message. Finally, if *l* is no longer *requesting*, no message is sent, since *l* exited from its critical section and, then, has already sent the RELEASE message. In all the cases $GRANT_l[y]$=FALSE.

If *y* receives *l*'s YIELD message, it removes *l*'s request from Λ_y and reinserts the request in Q_y. Otherwise, if *y* receives a RELEASE message, it removes *l*'s request from L_y. In both cases Condition (a) holds and it is consistent with Conditions (b) and (c).

Now we show that in the algorithm of figure 2, even though the Condition (ii) on the request sets holds it is not enough strong to preserve safety if M is greater than one.

Let us suppose that there are seven processes that compete for two resources (i.e., M=2). The request sets of the processes are shown in figure 1 and they verify Conditions (i) and (ii). Moreover, suppose that all processes raise a request for one resource (i.e., k_i=1 $\forall i$), no process is in its critical section (i.e., each Λ is empty) and, that the request timestamps have the value of the label of the process. Given (3), each process sends two GRANT messages. Therefore, four processes (1, 2, 3 and 4) obtain all the necessary GRANT messages to enter their critical sections violating the inequality (1).

The problem is that, if there are two resources, condition (ii) does not guarantee one arbiter for each triple of processes in conflict.

```
When requesting k_i resources
  begin     requesting ← TRUE; for each w ∈ R_i do GRANT[w] ← FALSE od;
            for each w ∈ R_i do send REQUEST(i, k_i, get_timestamp()) to w; od;
            repeat
                  receive GRANT from w;
                  GRANT[w] ← TRUE;
            until (GRANT[w]=TRUE for each w∈R_i );
            <critical section>
            for each w∈R_i do send RELEASE(i, k_i, ts_i) to w; od;
            requesting ← FALSE;
  end

Upon receiving a REQUEST (j, k_j, ts_j) from j ;
    begin  Q ← Q + [j k_j, ts_j];
           for each w∈Λ do
                  if ((ts_j< timestamp (Λ[w]) or ((ts_j=timestamp(Λ[w])) and ( j<w )) then
                        if (w=i) and (<i is outside the critical section>) then
                              GRANT[i] ← FALSE;
                              Q ← Q + Λ[i];
                              Λ ← Λ - Λ[i];
                        else
                              if (w≠i) then
                                    send INQUIRE to w;
                                    select
                                          when receive a RELEASE (w, k_w, ts_w) from w;
                                          when receive an YIELD from w
                                                Q ← Q + Λ[w];
                                    end select;
                                    Λ ← Λ - Λ[w];
                              fi;
                        fi;
                  fi;
           od;
(XX)       while ( Σ_{w∈Λ} resources(Λ[w]) + resources(head(Q))≤ M) do
                  send GRANT to process(head(Q));
                  Λ ← Λ + head (Q) ;
                  Q ← Q - head(Q);
           od;
  end

Upon receiving a RELEASE (j, k_j, ts_j) from j
      begin  Λ ← Λ - [j, k_j, ts_j];
             if Q≠ empty then (act like step XX) fi;
       end

Upon receiving an INQUIRE from j
      begin
            if (<i is outside the critical section> and requesting) then
                  GRANT[j] ← FALSE;
                  send YIELD to j;
            fi;
      end
```

Notice that initially all Qs and Λs are empty. Each procedure of receiving a message is executed as an atomic action.

Figure 2. The *k*-out of-M resources allocation algorithm code

5. Safety

In order to preserve *safety* inequality (1) must hold at any given time. Then, *each possible conflict must be detected and resolved.*

Given (1), the number of processes simultaneously in their critical section falls between 0 and M. Then, in the worst case we must arbitrate (M+1)-tuples of processes in conflict. We can now deduce the following rules:

Conflict detection rule. For each one of distinct (M+1)-tuples of conflicting processes there must exist at least one arbiter, so that every time M+1 processes try to allocate resources, there is someone that detects this event.

Conflict resolution rule. Given an (M+1)-tuple of conflicting processes, according to (3) the relative arbiter permits at most M processes to enter their critical sections.

It is clear that if the *conflict detection rule* is satisfied, any conflict of (h)-tuples (with 1<h≤M) of processes can be detected, whereas a conflict of h processes (with M+1<h≤N) will be split in $\binom{h}{M+1}$ conflicts of distinct (M+1)-tuples. The *conflict detection* rule leads to the following extension of Condition (ii'):

(iii') there exists at least one arbiter

$\forall <h_1,..., h_{M+1}> \in U$ ***and*** $h_i \neq h_j$

i.e.,

(iii) $R_{h_1} \cap .. R_{h_M} \cap R_{h_{M+1}} \neq \varnothing$

$\forall <h_1,..., h_{M+1}> \in U$ ***and*** $h_i \neq h_j$

The latter condition is necessary to meet *safety* in a *k*-out of *M*-resources permission-based algorithm. In particular, for the proposed algorithm we have

Theorem 1: *Condition (iii) is necessary and sufficient to guarantee Safety in the algorithm of Section 4.*

Proof: *Sufficiency*: Let Θ be an (M+1)-tuple ($<h_1,..., h_{M+1}>$) of requesting processes ordered by timestamp (then, $\sum_{i \in \Theta} k_i > M$), and let w be Θ's arbiter. Given the algorithm of Section 4, w sends a number of GRANT messages that is equal to the maximum integer Z that verifies

$$\sum_{i=1}^{Z} resources(\Lambda_w[h_i]) \leq M \quad i \in \Theta$$

Therefore, only Z members (with $1 \leq Z \leq M$) of Q may enter simultaneously their critical sections while (M+1-Z) requests are inserted in Θ_w. □

Necessity: We will prove the necessity showing that *safety* is violated, when Condition (iii) is not verified. i.e.,

$$\exists\, \Theta \in U : R_{h_1} \cap .. R_{h_M} \cap R_{h_{M+1}} = \varnothing \qquad (4)$$

Let us suppose $<h_1,..,h_M>$ to be in their critical sections (of course they have one resource each), w to be their arbiter (i.e., $R_{h_1} \cap .. \cap R_{h_M} = \{w\}$) and no request pending. Then for process w we have

$$\sum_{i=1}^{M} resources(\Lambda_w[h_i]) = M \qquad (5)$$

for all other processes we have

$$\sum_{i=1}^{M} resources(\Lambda_j[h_i]) \leq M-1,\ \forall j \in \{1,..N\} \textbf{\textit{and}}\ j \neq w \quad (6)$$

If process h_{M+1} requests one resource, given (4) w is not a member of $R_{h_{M+1}}$, then it will not receive h_{M+1}'s request. Given (5) and (6) w is the only process such that

$$\sum_{i=1}^{M} resources(\Lambda_w[h_i]) + 1 > M \qquad (7)$$

then it is the only one able to block h_{M+1} from entering its critical section. Then, h_{M+1} will receive all GRANT messages to enter its C.S., violating (1). i.e.,

$$\sum_{i=1}^{N} k_i > M \qquad \square$$

6. Fairness

We have to prove that each request will be satisfied in a finite time. Let us introduce the following lemma:

Lemma. *The algorithm of Section 4 resolves conflicts fairly.*

Proof. (Sketch) Conflicts among processes are resolved using the timestamp of their requests and their name. Each time that an unfair conflict resolution arises, a *recovery action* starts. Timestamp method [2] has been shown to be a fair policy to resolve conflicts in distributed systems in [1]. □

Theorem 2. *The proposed k-out of-M resources guarantees fairness.*

Proof. (Sketch) the claim follows by the *lemma* and by the following observations

- there is a finite number of processes;
- all conflicts are arbitrated;
- no process remains indefinitely in its critical section;
- each process releases all resources at the same time;

This guarantees that the waiting time for each request is finite, so fairness is preserved. □

7. The Arbiter and Request set size

First of all, we show that the *finite projective plane theory* does not apply if M is greater than one. At this purpose we give a case in which an axiom of Section 3 is violated.

Let Condition (iii) be verified, M be greater than one, w be the arbiter of the (M+1)-tuple $<i, j, h_3,..., h_{M+1}>$ and q be the arbiter of the (M+1)-tuple $<i, j, v_3,..., v_{M+1}>$. Then, we have

$$R_i \cap R_j \cap R_{h_3} \cap .. \cap R_{h_{M+1}} = \{w\}$$

$$R_i \cap R_j \cap R_{v_3} \cap .. \cap R_{v_{M+1}} = \{q\}$$

where, $<h_3,..., h_{M+1}> \neq <v_3,..., v_{M+1}>$. Therefore,

$$R_i \supseteq \{w, q\}$$

$$R_j \supseteq \{w, q\}$$

This implies that

$$R_i \cap R_j \supseteq \{w, q\}$$

This contradicts axiom 1 of Section 2. (i.e., Two lines contain at least two distinct points).

Hence, to compute the *minimal* cardinality of arbiter sets, we introduce a combinatorial approach. Of course, if M=1 we get the results of sections 2 and 3. Condition *(iii)* implies a relationship among the number of resources available (M), the number of processes (N) and the cardinality of the arbiter set (X), that in turn influences the cardinality of requests sets.

Let P be the set of all distinct (M+1)-tuple in a system of N processes and let Q_i be the set of all distinct (M+1)-tuples of processes arbitrated by i. The condition (iii') holds true iff

$$P = \bigcup_{i=1}^{N} Q_i \tag{8}$$

If (8) is verified, in numerical terms we have

$$\binom{N}{M+1} \leq \sum_{i=1}^{N} \binom{X_i}{M+1} \tag{9}$$

For a *minimal symmetric algorithm* each process should arbitrate the same number of (M+1)-tuples and each (M)-tuple must be arbitrated by one process:

$$|Q_1| = ... = |Q_N| \quad \textbf{and}$$

$$Q_i \cap Q_j = \varnothing \quad \forall i, j \in U \textbf{ and } i \neq j$$

In this case inequality (9) becomes

$$\binom{N}{M+1} = N\binom{X}{M+1} \tag{10}$$

that is

$$\prod_{i=0}^{M}(X-i) = \prod_{i=1}^{M}(N-i) \tag{11}$$

In general, for a *minimal symmetric algorithm* with M resources, the arbiter set cardinality is given by (11), hence we obtain

$$X = O(N^{\frac{M}{M+1}})$$

Let us compute the *minimal* request set cardinality (R). A request set is deduced from arbiter sets inverting the definition given in Section 3. Then for the process i we get

$$R_i = \{ h \mid i \in A_h \}$$

The number of distinct (M+1)-tuples that contain process i is given by

$$\binom{N-1}{M} \tag{12}$$

and, the number of (M+1)-tuples in which i is arbitrated by the same process is given by

$$\binom{X-1}{M} \tag{13}$$

Then, the following equality holds

$$R\binom{X-1}{M} = \binom{N-1}{M} \tag{14}$$

Given (11) we have

$$R = \frac{\prod_{i=0}^{M}(X-i)}{\prod_{i=1}^{M}(X-i)} \qquad (15)$$

Therefore,

$$R = X = O(N^{\frac{M}{M+1}})$$

In the next Section we will show a method to obtain a near-optimal set of request sets.

7.1 The Creation of R_i's

Consider a grid of $L_1xL_2x..L_{M+1}$ points (i.e., processes) and number the points from 1 to L^{M+1}. This grid defines a discrete space of dimension M+1. In the grid there exist L^{M+1} orthogonal hyperplanes of dimension M.

A request set R_i is the set of the grid points on the M+1 orthogonal hyperplanes of dimension M passing through i.

For simplicity's sake we assume M equal to 2 and L equal to 3. Then, we have a grid of 27 points on 3 parallel planes of 9 points each, as shown in figure 3.

In this case R_i is the set of grid points on the three orthogonal planes passing through i.

It is clear that

$$R_i \cap R_j \cap R_k \neq \varnothing \qquad \forall \text{ i, j, k} \in U \text{ and } i \neq j \neq k$$

In figure 3 the processes that belong to the request set of process 1 are shown. In this construction the cardinality of the request set is given by

$$|R_i| = L^2 + (L^2 - L) + (L^2 - 2L + 1)$$

given that

$$L = N^{1/3}$$

we have

$$|R_i| = 3N^{2/3} - 3N^{1/3} + 1$$

Therefore, the cardinality of the request sets is greater than that minimal of previous section but it is still $O(N^{2/3})$. If N is not a cube of an integer we create R_i's considering a grid of D^3 points where D is the smallest integer such that $N<D^3$ (i.e., we insert D^3-N dummy points). Afterwards we create the D^3 request sets and remove all dummy points from R_i's.

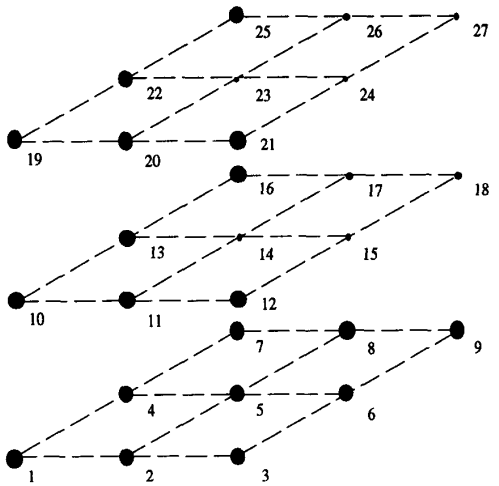

Figure 3. The grid in case of N=27 and M=2

8. Message Traffic

The number of messages for acquiring a set of resources in the *minimal symmetric* algorithm lies between $3\lceil N^{(M/(M+1))} - 1\rceil$ and $5\lceil N^{(M/(M+1))} - 1\rceil$.

Raynal's algorithm [6] needs at most *3(N-1)* messages per critical section entry. Then, for low values of M our proposal is better. However, the greater M is, the better is Raynal's algorithm due to the extra messages of our recovery actions. To improve our upper bound we need local recovery actions that do not exchange messages.

This is possible, by means of a clever choice of the arbiters. In fact, Singhal shows [12] a distributed mutual exclusion algorithm derived from that of Maekawa [3] that does not need extra messages to avoid deadlocks. Among other things, Singhal's algorithm works correctly iff the following conditions are true:

(iv) $i \in R_i \qquad \forall i \in U$

(v) $R_i \cap R_j = \{i\}$ *or* $\{j\} \qquad \forall i, j \in U$ **and** $i \neq j$

Condition (v) says that for each distinct pair of processes the arbiter must be one of them. Let us suppose that process i is the arbiter of i and j and i is in its critical section. If j's request has a lower timestamp than that of i, i must exit from its critical section and send the GRANT message to j. Given Condition (v) this recovery

action is local and does not require transmission of extra messages. Condition (v) is stronger than Condition (ii) and implies the cardinality of the request set to be *O(N/2)* [12].

Let us now derive the condition that permits local recovery in the *k*-out of-*M* problem. In the worst case M processes may be in their critical sections simultaneously. Let us suppose $<h_1,...,h_M>$ to be those processes and that all of them have a greater timestamp than that of a request for using M resources of process *i*. Local recovery may be carried out only if each process of $<h_1,..., h_M>$ exits from its critical section and sends the GRANT message to *i* without other messages. i.e., the arbiter of the (M+1)-tuple $<i, h_1,..., h_M>$ must be the (M)-tuple $<h_1,..., h_M>$.

Hence, Condition (v) can be extended as follows:

(vi) $R_{h_1} \cap .. R_{h_M} \cap R_{h_{M+1}} = \{w_1,..,w_M\}$

$<w_1,...,w_M> \in <h_1,...,h_{M+1}>$ ***and***

$<h_1,...,h_{M+1}> \in U$ ***and*** $h_i \neq h_j$

The constraint imposed by Condition (vi) is stronger than Condition (iii) and is weaker than what is required in Raynal's algorithm [6]. In fact, the latter works correctly iff

(vii) $R_i \equiv U \; \forall i$

The comparison between Conditions (vi) and (vii) shows that the average cardinality of the request sets of Raynal's algorithm is greater than or equal to that of the proposed algorithm.

Hence, if Condition (vi) is verified, in our algorithm, the average number of messages exchanged per critical section entry (Nm) is given by

$$Nm_1 = 3\left\lceil \frac{N-1}{2} \right\rceil \qquad \text{if M=1 } \textbf{\textit{and}}$$

$$Nm_{M-1} < Nm_M \leq 3(N-1) \qquad 1<M<N-1$$

Conclusions

An $O(N^{M/(M+1)})$ distributed solution to the *k*-out of-*M* resources allocation problem has been shown. This solution reduces the complexity of the problem, in terms of messages exchanged per critical section entry, compared to that proposed by Raynal [6] which was *O(N)*. The result has been obtained investigating the concept of *arbiter* of conflicting processes and extending conditions that permit *conflict detection and resolution* in a system of N competing processes to M shared resources. Moreover, a method to create a near-optimal set of requests sets has been given and the condition that permits local recovery in our proposal has been discussed.

Acknoledgements

The author would like to thank my Ph.D. advisor Giacomo Cioffi for his generous support and encouragement and Bruno Ciciani for many helpful comments on an earlier version of the paper. The author is also grateful to Alberto Marchetti-Spaccamela for some helpful conversations. Suggestions of Arif Ishaq were useful in clarifying the presentation.

References

[1] CHANDY, K. M., AND MISRA, J. The Drinking Philosopher Problem. *ACM Trans. Prog. Lang. Syst.* **6**, 4 (Oct. 1984), 632-646

[2] LAMPORT, L. Time, Clocks and the Ordering of Events in a Distributed System. *Commun. ACM* **21**, 7 (July 1978), 558-565

[3] MAEKAWA, M. A $\sqrt{N}$ Algorithm for Mutual Exclusion in Decentralized Systems. *ACM Trans. Comput. Syst.* **3**, 2 (May 1985), 145-159

[4] NAKAJIMA, A. Using a Finite Projective Plane with a Duality for Decentralized Consensus Protocols. *Proc. 12th IEEE Int. Conf. on Dist. Comp. Systems*. Yohohama (1992) 665-672

[5] RAYMOND K. A Distributed Algorithm for Multiple Entries to a Critical Section, *Inform. Process. Lett.* 30 (1989) 189-193

[6] RAYNAL, M. A Distributed Solution to the *k*-out of-*M* Resources allocation Problem. *Lect. Notes. Comp. Sci.* 497, Springer-Verlag (1991) 599-609

[7] RAYNAL, M. A Simple Taxonomy for Distributed Mutual Exclusion Algorithms. *ACM Operat. Syst. Review* **25**, 1, (1991)

[8] RAYNAL, M. *Distributed Algorithms and Protocols*. John Wiley & Sons (1988)

[9] RICART, G., and AGRAWALA, A.K. An Optimal Algorithm for Mutual Exclusion in Computer Networks. *Commun. ACM* **24**, 1 (Jan.1981), 9-17

[10] SANDERS, B.A. The Information Structure of Distributed Mutual Exclusion Algorithms. *ACM Trans. Comput. Syst.* **5**, 3 (Aug.1987), 284-299

[11] SINGHAL, M. A Class of Deadlock-free Maekawa-type algorithms for Mutual Exclusion in Distributed Systems. *Distr. Comput.* **4**, (1991), 131-138

[12] SRIMANI,P.K. AND REDDY R.L.N. Another Distributed Algorithm for Multiple Entries to a Critical Section. *Inform. Process. Lett.* 41 (1992) 51-57