4º SBRC
RECIFE - 24 A 26 DE MARÇO 86

DISTRIBUTED ALLOCATION OF IDENTICAL RESOURCES

Osvaldo S. F. Carvalho
Universidade Federal de Minas Gerais
Depto. de Ciência da Computação (*)

1. INTRODUCTION

In this paper we present a distributed algorithm for the allocation of a pool of R identical resources shared by $n > R$ nodes of a computer network. We assume that a node uses at most one resource at a time, during a finite but impredictable interval of time. The algorithm guarantees that the same resource is never used by two or more nodes simultaneously. It is also free of deadlock and starvation phenomena.

Each node following the algorithm exchanges messages only with its neighbours in an *acyclic graph* which covers the network. Therefore the algorithm requires only a communication schema supporting message-passing primitives between couples of neighbours in the acyclic graph. However we believe that this algorithm may be adopted even when a communication schema is available in which every node is connected with each other, because it prevents excessive resource migration which may result from broadcasting requests.

Up to a significant part this algorithm has been developed using predicate distribution techniques [C&R82,84]. In fact, the second main purposal of this paper is to illustrate the usefullness of this technique as a tool for the design of a class of distributed algorithms.

This paper is organized as follows. In section 2 we make a brief exposition of the technique of predicate distribution Then we give a specific distribution for the R-resources problem which is results from the application of the general

(*) Este trabalho foi desenvolvido na Universidade de Paris-Sul, Orsay, tendo sido parcialmente financiado pela CAPES.

schema given in [C&R84, CAR85] for the distribution of a predicate following an acyclic graph. We take as the predicate to be distributed a formal expression of *a resource is never used by two or more nodes simultaneously*. From the distribution of this predicate results a new version of the initial problem, where the network integrity is implied by the conjunction of a set of *local* and *global* predicates. Each of these global predicates involves variables localized at two neighbour nodes.Furthermore they are of a standard form for which a programming technique is known (the "acceptance thresholds method") which guarantees their invariance by means of a small set of simple rules.

The algorithm is presented in section 3. It is a solution to the transformed problem obtained in section 2. In order to arrive to the final program we make use of standard techniques for deadlock prevention introduced in [LAM78] and [R&A81].

In section 4 we compare our solution with an analogous algorithm proposed in [A&&82].

2. DISTRIBUTION OF THE INTEGRITY PREDICATE

The R-resources problem is a member of a class of distributed programming problems which may be stated as follows:

Given a network with n nodes such that at any instant node i $(1 \leqslant i \leqslant n)$ is at one and only one state belonging to a set E_i, construct a system of n controllers (one for each node) such that:

1) a predicate $P: \times_{i=1}^{n} E_i \rightarrow \{false, true\}$ is kept invariant;

2) some "additional requirements" are satisfied.

The "additional requirements" may include freedom of deadlock and starvation, [illegible]nal performance levels, etc.. During the distribution phase we shall not deal directly with these requirements.

For the R-resources problem we may associate to each node i the set $E_i=\{IDLE,1,\ldots,R\}$ which is to be interpreted as follows:

$e_i=r\neq IDLE$ => *node i is currently using resource r;*

$e_i=IDLE$ => *no resource is currently being used by node i*

Here e_i denotes the current state of node i and thus $e_i\in E_i$. Predicate P for the R-resources problem may then be expressed by

$$P \equiv [\ \forall i,j\in[1,N],\ (i\neq j \wedge e_i=e_j) \Rightarrow (e_i=e_j=IDLE)\ ] \qquad (1)$$

Each controller in a solution to a problem of this class will require a certain amount of control data. With the distribution of P, as defined bellow, we make a partial determination of this control data.

In what follows we consider the current state of a node as being composed (at least) by its "fundamental" state e_i plus the current state of the control data used by its controller.

Definition. A *distribution* of a predicate $P:\overset{n}{\underset{i=1}{\times}} E_i\rightarrow\{false,true\}$ over an n-node network is a determination of:

(1) n sets $C_1,\ldots,C_n$; each C_i is to be interpreted as a set containing all possible states of a part of the control data of node i;

(2) n *local predicates* $LP_1,\ldots,LP_n$; each $LP_i:E_i\times C_i\rightarrow\{false,true\}$ relates the fundamental and control states of node i;

(3) a *global predicate* $GP:\overset{n}{\underset{i=1}{\times}} C_i\rightarrow\{false,true\}$, relating the control states of all nodes in the network between themselves,

such that

$I)\ [\ \overset{n}{\underset{i=1}{\wedge}} LP_i\ \wedge\ GP\ ] \Rightarrow P$ (consistency) and

II) $\forall (e_1, \ldots, e_n) \in P,\ \exists (c_1, \ldots, c_n) \in GP$
such that (completeness)[1]
$\quad \forall i,\ (e_i, c_i) \in LP_i$

Whenever we have a distribution (C, LP, GP) of P we may transform the initial problem statement into:

Given a network with n nodes such that at any instant node i is at one and only one state belonging to the set $E_i \times C_i$, construct a system of n controllers such that:

(1) the predicate GP is kept invariant;

(2) each controller i will have LP_i as a local invariant[2]*;*

(3) some "additional requirements" are satisfied.

Consider a solution to the transformed problem. The consistency property of the distribution insures that it will also keep P invariant; the completeness property says that it will not forcibly implement an invariant which is stronger than P.

The work spent in doing predicate distribution will be worthwhile if the new statement is more easily solvable. In [C&R84] we propose general schemata for the distribution of any predicate P following an acyclic graph which covers the network, in such a way that the resulting global predicate - which we may assume to concentrate the "difficulty" in this class of programming problems - is a conjunction of "bi-local" predicates, each one refering to data in neighbour nodes in the graph. In this way the initial n-node invariance problem is decomposed into a collection of 2-node invariance problems. Note however that we

[1] In the expression of the completeness property we identify a predicate with the set that contains all and only those elements which satisfy that predicate.

[2] We say that a predicate Q is a local invariant of controller i iff (1) Q refers only to the state of node i and (2) the invariance of Q does not depend on node interaction.

are only concerned here with invariance aspects; deadlock and starvation, e.g., will remain global problems even with such a distribution.

Using those schemata one founds directly the distribution for the R-resources problem that follows.

- Let V_i denote the set of neighbours of node i in the acyclic graph of the network. For each $j \in V_i$ we shall have at node i an array $m_i^j[1..R]$ of **boolean** ; *C_i will be given† by the set of possible values for the arrays m_i^j.*

  The *local predicate* of node i will be given by

$$LP_i \equiv \left[\begin{array}{c} \forall j \in V_i,\ \forall r \in [1,R], \\ m_i^j[r] = \textbf{true} \Rightarrow \left[\begin{array}{c} e_i \neq r \\ \wedge \\ \forall k \in V_i,\ k \neq j,\ m_i^k[r] = \textbf{false} \end{array} \right] \end{array} \right] \qquad (2)$$

- The *global predicate* will be given by:

$$GP = \bigwedge_{i=1}^{N} [\bigwedge_{j \in V_i} GP_{ij}]$$

where

$$GP_{ij} \equiv \forall r \in [1,R],\ \left[m_i^j[r] = \textbf{false} \Rightarrow m_j^i[r] = \textbf{true} \right] \qquad (3)$$

It may be verified that this distribution is consistent and complete.

The local and global predicates given above are better understood when the variables m_i^j are interpreted in the following way. Let j be a neighbour of node i. We denote by $t(j/i)$ the set of nodes belonging to the maximal connected component of the network containing j that would result from the rupture of the $i-j$ connection. Since the network is acyclic, i does not belong to $t(j/i)$. In the network shown in figure 1 we have, e.g., $t(4/2) = \{4,5,6\}$, $t(1/2) = \{1\}$, etc. Using this notation, $m_i^j[r] = \textbf{false}$ *means that no node belonging to $t(j/i)$ may* *be using resource r; otherwise $m_i^j[r]$ =* **true**.

To close this section we remark that effectively the distribution of (1) makes the solution of the R-resources problem easier. This is so because we have decomposed our initial n-node distributed problem into a collection of subproblems. Each strictly distributed subproblem is the maintenance of a predicate involving only two neighbour nodes; this invariance may be assured by the use of the acceptance thresholds method, as we shall see in next section.

3. ALGORITHM

The algorithm is presented in the form of an array

controller *R−RESOURCES* (I: 1..n)

of **controllers** which are permanent data structures accessed only through their **public procedures**. Except for the **waitfor** statement, the public procedures of a controller are executed in a locally indivisible way.

There are two processes at each node. One of them actually uses the resources, calling cyclically the public procedures *REQUEST−RESOURCE* and *RELEASE−RESOURCE*. The other process receives the arriving messages and calls the *TREAT−MESSAGE* procedure for their treatment.

Each controller keeps some permanent constant data related to the position of the associated node in the acyclic graph. The integer I identifies each

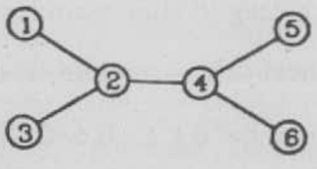


Figure 1: An acyclic network

controller. The identifications of the neighbours of node I are stored in an array $V_I[1],\dots,V_I[v_I]$; v_I is the number of neighbours of node I (Remark that the subscript I is implicit in the text of the algorithm). These constants will naturally differ from node to node; we assume the existence of a protocol that assigns appropriate values to them.

Permanent data structure of controller I includes the following variables:

- an array $m_I[1..v_I,1..R]$ of **boolean** ; $m_I[j,r]$ corresponds to $m_I^{V_I[j]}[r]$ in the notation of the last section.

- an array $TH_I[1..v_I]$ of **integer** ; $TH_I[j]$ is the acceptance threshold (see below) of node I relative to neighbour $V_I[j]$.

- an integer HSN_I such that we have always $HSN_I \geq \mathbf{max}(TH_I[1],\dots,TH_I[v_I])$; HSN_I is also used as a logical clock [LAM78].

- a variable $STATE_I$, which can assume the values "IDLE", "WAITING" or an integer between 1 and R, with obvious meanings.

- a list $RESOURCES_I$ which contains all resources which are available at node I.

Since local invariant (2) is preserved, a resource r belongs to $RESOURCES_I$ iff $STATE_I \neq r$ and $m_I[j,r] =$ **false** *for all j in $[1,v_I]$.*

All messages used in the algorithm have the form:

TYPE	*SENDER*	*TH*	m

The meaning of the *SENDER* field is quite obvious; the *TYPE* field may assume values "REQUEST" or "RELEASE". A message of type "REQUEST" is sent by node I to neighbour $V_I[j]$ when there is at node I a situation of lack of resources. When there are enough resources to satisfy node I itself and also the REQUEST's of the other neighbours which have already arrived to node I, a type "RELEASE" message may be sent to node $V_I[j]$. Fields *TH* and m are used as specified by rule 2 below.

Consider now the following set of programming rules, which have been used by the author during the design of the solution proposed here. They constitute a "translation" of the acceptance thresholds method [C&R82,83,84]; their observance leads to algorithms which are guaranteed to keep predicates (3) invariant.

Rule 1.

all the *TH* variables are monotonically increasing integers;

Rule 2

when node I sends a message σ of either type to neighbour $V_I[j]$, the field $\sigma.TH$ will contain the value of $TH_I[j]$ at the sending instant, and $\sigma.m$ will contain the value of the row $m_I[j,1],...,m_I[j,R]$ at the sending instant;

Rule 3.

at any instant the value of $TH_I[j]$ must be greater than or equal to the maximal *TH* field present at any message sent to or received from $V_I[j]$ up to that instant;

Rule ·

for any resource r, any event at node I where the value of $m_I[j,r]$ passes from *true* to *false* must be deflagrated by the arrival of a message ρ, coming from $V_I[j]$, such that at the arrival instant (i.e., before that any modification could be produced by this event):

$\rho.TH > TH_I[j]$ or $(\rho.TH = TH_I[j] \wedge V_I[j] > I)$, and

$\rho.m[r] = \textbf{true}$ and $m_I[j,r] = \textbf{true}$

These are indeed practical rules because they are simple enough both to permit an easy verification of their observance by means of an inspection of the text of the algorithm, and also to be kept consciously in mind during the design of the algorithm.

For a complete verificati >n of the invariance of (1), it is also necessary to show that the execution of any procedure of controller I keeps invariant the local predicates (2). But this can also be done through an inspection of the text of the algorithm, which has been constructed in such a way that this invariance does not depend on any interaction with the other nodes.

More permanent variables are included in the data structure of controller I:

- an ordered list $REQUESTING_I$, which contains all requesting neighbours of node I. This list is ordered by the TH field of the $REQUEST$ message used to communicate to node I this request situation.
- an array $REP-EXP_I[1..v_I]$ of *boolean* ; $REP-EXP_I[j]$ = **true** if node I has sent a $REQUEST$ message to $V_I[j]$ which has not been replyed.

Full details and additional data structures may be seen directly in the algorithm.

```
controller R-RESOURCES[1..N];

({1..R},"IDLE","WAITING"): STATE initial ("IDLE");
constant integer R, v, V[1..v];
boolean m[1..v,1..R];          /* see note on initialization */
integer TH[1..v] initial (0),
        HSN initial (0),
        SNDR initial (0),          /* SNDR is used to prevent backward propaga-
        tion of requests */
        HRSN[1..v] initial (0);  /* Highest Received Sequence Number */
  list RESOURCES of ( integer );
  ordered list REQUESTING of ( integer, key integer);
  /* see note on initialization */
  boolean REP-EXP[1..v] initial (false);
  type Σ = record ( ("REQUEST","RELEASE"): TYPE;
                      integer SENDER, TH;
                      boolean m[1..R]  );
```

Communication medium. We assume an underlying communication medium which delivers correct messages in a finite but arbitrarily long delay, at an order which may differ from the order in which the messages are sent.

```
public procedure
REQUEST-RESOURCE( integer: RES):
begin
    STATE:= "WAITING"; NORMALIZE;
    waitfor ($1 \leq$ STATE $\leq$ R);   RES:= STATE
end REQUEST-RESOURCE.
```

```
public procedure
RELEASE-RESOURCE( integer : RES):
begin
    put(STATE,RESOURCES); STATE:= "IDLE";  RES:= 0;
    NORMALIZE
end RELEASE-RESOURCE.
```

```
public procedure TREAT-MESSAGE($\Sigma$: $\rho$):
begin integer i,j,r;
    j:= "i such that V[i] = $\rho$.SENDER";
    if $\rho$.TH > TH[j] or ($\rho$.TH = TH[j] and $\rho$.SENDER > I)
    then /* consider the m field of the message */
        TH[j]:= $\rho$.TH;  HSN:= max(HSN,$\rho$.TH);
        HRSN[j]:= $\rho$.TH;
        for r:=1 until R do
            if $\rho$.m[r] = m[j,r] = true
            then m[j,r]:= false; put(r,RESOURCES);
            REP-EXP[j]:= false
            fi
        od;
        if $\rho$:TYPE = "RELEASE"
        then delete (j,REQUESTING) fi
    fi;
    if $\rho$. TYPE = "REQUEST" and $\rho$.TH > HRSN[j]
    then insert (j,$\rho$.TH,REQUESTING); SNDR:= j;  HRSN[j]:= $\rho$.TH
    fi;
    NORMALIZE
    /* the insert operation deletes older (with less valued TH) request's of node
    j,if any */
end TREAT-MESSAGE.
```

```
procedure NORMALIZE:
begin integer r,k;
    HSN:= HSN + 1;
    if STATE = "WAITING" and #(RESOURCES) > 0
    then get(STATE,RESOURCES)
    fi;
    if #(REQUESTING) > #(RESOURCES) or STATE = "WAITING"
    then while #(RESOURCES) > 0 do
            get(r,RESOURCES); get(k,REQUESTING);
            m[k,r] := true; REP-EXP[k]:= true; TH[k]:= HSN;
            send Σ("REQUEST",I,HSN,m[k,.]) to V[k]
        od;
        for k:=1 step 1 until v do
            if REP-EXP[k] = false and k ≠ SNDR and $\bigvee_{r=1}^{R}$ (m[k,r]) = true
            then REP-EXP[k]:= true;
                TH[j]:= HSN;
                send Σ("REQUEST",I,HSN,m[k,.]) to V[k]
            fi
        od
    else while #(REQUESTING) > 0 do
            get(r,RESOURCES); get(k,REQUESTING);
            REP-EXP[k]:= false; m[k,r]:= true; TH[j]:= HSN;
            send Σ("RELEASE",I,HSN,m[k,.]) to V[k]
        od
    fi;
    SNDR:= 0
    /* #(list) gives the number of elements in list */
    /* m[k,.] denotes the row m[k,1],...,m[k,r]     */
end NORMALIZE.

end R-RESOURCES.
```

Initialization. There are many correct starting points for this algorithm; we suggest the following one. For initialization purposes only, each node has designated a special one, among its neighbours $V_I[FATHER_I]$, which is his father with respect to the tree rooted in node 1. Naturally any other node could serve as well.

Every node $I \neq 1$ should then be initialized with $m_I[j,r] = \text{false} \;\; \forall j \neq FATHER_I, \; \forall r \in [1,R]$, and with $m_I[FATHER_I,r] = \text{true} \;\; \forall r$ Node 1 must be initialized with $m_1[j,r] = \text{false} \;\; \forall j, \; \forall r$ Consistently we must have the list $RESOURCES_I$

initially empty for every node, except for node 1 which has initially all R resources in its list $RESOURCES_1$. Without exceptions the list *REQUESTING* should be initially empty for all nodes.

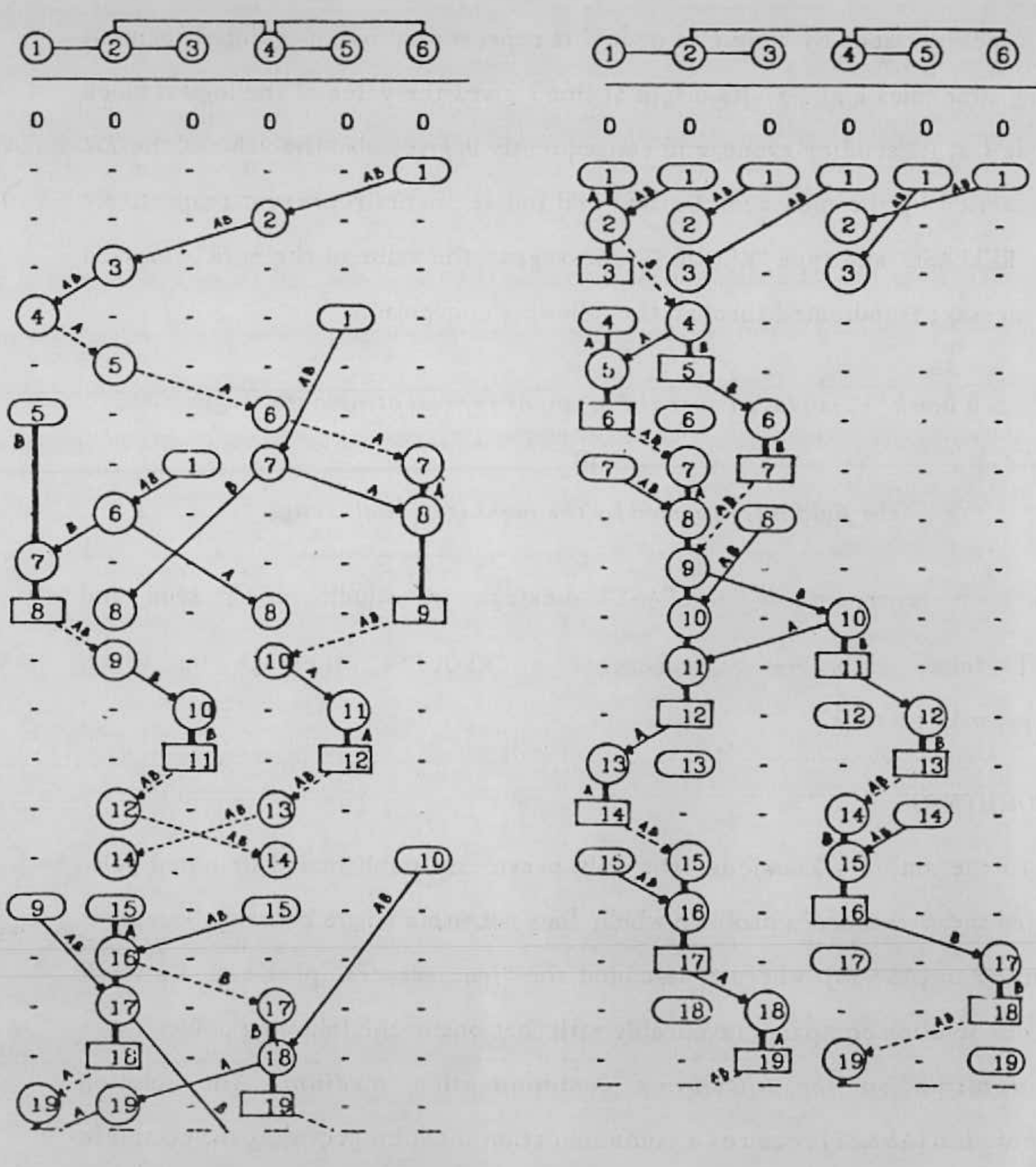
Absence of deadlock and starvation. It can be verified that this algorithm is free of deadlock and of starvation phenomena. By lack of space we shall omit the proof, which relies basically on the guaranties given by the communication medium for message delivering, on the monotonically increasing character of the acceptance thresholds and logical clocks, and also on the fair policy of the *REQUESTING* queue.

3.1. Example.

We show in figure 2 two examples of the use of algorithm R-RESOURCES for the allocation of two identical resources A and B, shared by the nodes of the network in figure 1. We use a graphical schema where associated to each node there is a vertical "time" line, representing successive values (growing downwards) of a global discrete clock.

Simultaneous events (on different nodes) are allowed. This does not means that simultaneity may be enforced by suitable programming. On the contrary, whenever the schema shows two simultaneous events, it means that we cannot infer from an examination of the code of the algorithm which event will precede the other. In other words, whenever an algorithm admits a computation where two events arrive at the same discrete instant, it will also admit at least two other computations, both with the same previous history, but each one with a different order of arrival of those events.

An event at node i is represented by one of the following symbols, which are placed over the time line of node i in a position that corresponds to their instant of arrival. A *REQUEST–RESOURCE* operation is represented by "$\boxed{12}$", a



(a) A computation (b) Another computation

Figure 2: Allocation of resources *A* and *B*

RELEASE operation by "$\boxed{5}$" and a *TREAT−MESSAGE* operation by "⑫" The integer placed inside the operation symbols gives the value of the variable HSN_i

immediately after the execution of the corresponding procedure.

A message sent by node i to node j is represented by an oriented segment linking time lines i and j. Its origin at line i gives the value of the logical clock of node i at its sending event, and consequently it gives also the value of the *TH* field carried by the message. Dashed and full segments represent respectively type "RELEASE" and type "REQUEST" messages. The value of the m field carried by a message is indicated through the following convention:

a mark "r" is placed over the segment representing a message

iff

the field $m[r]$ carried by the message equals true.

Thus $\overset{A}{\dashrightarrow}$ represents a "RELEASE" message for which $m[A]$ = **true** and $m[B]$ = **false**, while $\xrightarrow{AB}$ represents a "REQUEST" message for which $m[A] = m[B] = true$.

4. COMMENTS.

To the author's knowledge the only previously published distributed solution for the R-resources problem which does not use a single central allocator is proposed in [A&&82], where it is called the "free servers" problem. We think that our solution compares favourably with that one in the following points.

Requirements on the underlying communication medium. The solution presented in [A&&82] requires a communication medium providing for complete network interconnection. In our solution only nodes that are neighbours in an acyclic graph must be connected. Furthermore algorithm R-RESOURCES is resistant to inversions in the order in which the messages are delivered

Symmetry. All controllers in the R-RESOURCES algorithm are textually identical. There is no need to distinguish between "client" and "allocator" nodes, as is the case in [A&&82]. However actual behaviour of each controller will depend on its position in the graph: the more "central" a node is, the more its behaviour will present an allocator character; the more external a node is, the more its behaviour will present a client character.

Performance (with high demand rates). Both message traffic and waiting time are not bounded in [A&&82] when the resources load factor approaches 1. Even with relatively low demand rates it is always possible to imagine a conspiration that leads a node to ***starvation*** when using [A&&82]. The solution we give here is free of these inconvenients.

It is out of the scope of this paper to make a more complete analysis of the performance of the R-RESOURCES algorithm. Also we have not examined which modifications are needed to make it resistant to failures.

To terminate we would like to remark that, from our point of view, the strongest quality of this algorithm is the sound basis over which relies (partially) its correctness. Over these same basis other algorithms can be constructed, implementing other allocation policies, perhaps with better performances. Most important, these basis have been obtained through a systematic development of the initial statement of the R-resources problem.

5.References.

[A&&82] ANDREWS,G., DOBKIN,D., DOWNEY,P. Distributed allocation with pools of servers. ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, Ottawa, 1982.

[C&R82] CARVALHO, O.S.F., ROUCAIROL, G. On the distribution of an assertion. ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, Ottawa, 1982.

[C&R83] CARVALHO, O.S.F., ROUCAIROL, G. On mutual exclusion in computer networks. Communications of the ACM 26,2 (Feb. 1983) 146-147 (Tech. Corr.)

[C&R84] CARVALHO, O.S.F., ROUCAIROL, G. Assertion decomposition for acyclic networks. E.R.A. CNRS no. 452 Rapport de Recherche no. 165, Université Paris-Sud, - LRI 91405 ORSAY, Fev. 1984.

[LAM78] LAMPORT, L. Time, clocks, and the ordering of events in a distributed system. Communications of the ACM 21,7 (July 1978).

[R&A81] RICART, G., AGRAWALA,A. An optimal algorithm for mutual exclusion in computer networks. Communications of the ACM 24,1 (January 1981)