Assignment of Resources in Distributed Systems

David L. LA RED MARTÍNEZ, Julio C. ACOSTA, Federico AGOSTINI
Faculty of Exact and Natural Sciences and Surveying, Northeastern National University
Corrientes, (3400), Argentine

ABSTRACT

In distributed processing systems it is often necessary to coordinate the allocation of shared resources that should be assigned in the processes in the modality of mutual exclusion; in such cases, the order in which the shared resources will be assigned in the processes that require them must be decided; in this paper we propose an aggregation operator (which could be used by a shared resources manager module) that will decide the order of allocation of the resources to the processes considering the requirements of the processes (shared resources) and the state of the distributed nodes where the processes operate (their computational load).

Keywords: Aggregation Operators, Concurrency Control, Communication Between Groups of Processes, Mutual Exclusion, Operating Systems, Processor Scheduling.

1. INTRODUCTION

The proliferation of computer systems, many of them distributed in different nodes with multiple processes that cooperate for the achievement of a particular function, require decision models that allow groups of processes to use shared resources that can only be accessed to in the modality of mutual exclusion.

The traditional solutions for this problem are found in [1] and in [2], which describes the main synchronization algorithms in distributed systems; In [3], it presents an efficient and fault tolerant solution for the problem of distributed mutual exclusion; In [4], [5] and in [6], which present algorithms to manage the mutual exclusion in computer networks; In [7], which details the main algorithms for distributed process management, distributed global states and distributed mutual exclusion.

The allocation of resources in processes should be performed taking into account the priorities of the processes and also the state in terms of workload of the computational nodes in which the processes are executed.

Also, solutions (which may be considered classic or traditional) have been proposed for very different types of systems distributed in [8][9][10][11] and in [12]. Other works focused on ensuring mutual exclusion have been presented in [13] and in [14]. An interesting distributed solution based on permissions is presented in [15] and a solution based on process priorities in [16].

The new decision models for allocating shared resources could be executed in the context of a shared resource manager for the distributed system, which would receive the shared resource requirements of the processes running on the different distributed nodes, as well as the computational load state of the nodes and, considering that information, the order (priority) of allocation of the requested resources for the requesting processes should be decided on. Consequently, it is necessary to count on aggregation operators specifically designed.

In this paper, a new aggregation operator will be presented specifically for the aforementioned problem. This falls under the category of OWA operators, more specifically Neat OWA. This will present an innovative method for shared resource management in distributed systems, based on [17].

2. DATA STRUCTURES TO BE USED

The premises, data structures and the operator mentioned in [17], are used as a starting point to create a new operator in the scenario described next.

Fig. 1 shows the resources requests by the processes, the resources already assigned and the nodes in which they are located.

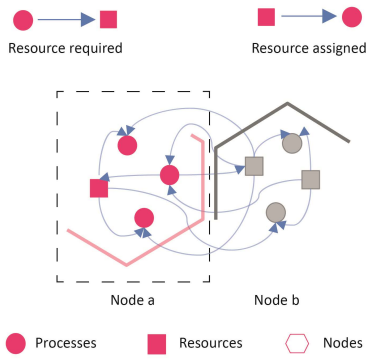

Fig. 1. Resources and processes at nodes in distributed systems.

These are groups of processes distributed in process nodes that access critical resources. These resources are shared in the form of distributed mutual exclusion and it must be decided, according to the demand for resources by the processes, what the priorities to allocate the resources to the processes that require them will be (to be assigned in the processes only those resources available will be taken into account, that is, those not yet allocated in certain processes):

- The access permission to the shared resources of a node will not only depend on whether the nodes are using them or not, but on the aggregation value of the preferences (priorities) of the different nodes regarding granting access to shared resources (alternatives) as well.
- The opinions (priorities) of the different nodes regarding granting access to shared resources (alternatives) will depend on the consideration of the value of variables that represent the state of each of the different nodes. Each node must express its priorities for assigning the different shared resources according to

the resource requirements of each process (which may be part of a group of processes).

Nodes hosting processes: 1, …. , *n*. The set of nodes is represented as follows:
nodes = {n_1, …. , n_n}

Processes housed in each of the n nodes: 1, …. , *p*. The set of processes is represented as follows:
processes = {p_{ij}} with i = 1, …, n (number of nodes in the distributed system) and j = 1, …, p (maximum number of processes in each node).

Distributed Process Groups: 1, …, *g*. The set of distributed process groups is represented as follows:
groups = {p_{ij}} with i indicating the node and j the process in this node.

Size of each of the *g* process groups. The number of processes in each group indicates the group's cardinality and is represented as follows:
card = {*card*(g_i)} with i = 1, …, g indicating the group.

Group priority of each of the *g* processes groups. These priorities can be set according to different criteria; in this proposal it will be considered to be a function of the cardinality of each group and is represented as follows:
prg = {*prgi* = *card*(g_i)} with i = 1, …, g indicating the group.

Shared *resources* in distributed mutual exclusion mode available on *n* nodes: 1, …., *r*, The set of resources is represented as follows:
resources = {r_{ij}} with i = 1, …, n (number of nodes in the distributed system) and j = 1, …, r (maximum number of resources at each node).

These available shared resources hosted on different nodes of the distributed system may be required by the processes (clustered or independent) running on the nodes:
Possible states of each process:

- Independent process.
- Process belonging to a group of processes.

Possible state of each of the nodes:

- Number of processes.
- Priorities of the processes.
- CPU usage.
- Main memory usage.
- Use of virtual memory.
- Additional memory required for each resource requested by each process (depending on the availability of the data).
- Additional estimated processor load required for each resource requested by each process (depending on data availability).
- Additional estimated input / output load required for each resource requested by each process (depending on data availability).
- Status of each of the shares in the distributed mutual exclusion mode in the node:
  - Assigned to a local or remote process.
  - Available.
- Predisposition (nodal priority) to grant access to each of the *r* shared resources in the modal of distributed mutual exclusion (will result from the consideration of the variables representative of the node status, the priority of the processes and the additional computational load which would mean allocating the resource to the requesting process).
- Current load of the node, which can be calculated as the average CPU, memory and input / output usage percentages at any given time (these load indicators may vary depending on the case, some may be added or changed); the current load categories, for example, High, Medium and Low, should also be defined, with value ranges for each category being indicated.

3. DESCRIPTION OF THE AGGREGATION OPERATOR

The proposed operator consists of the following steps:

1. Calculation of the current computational load of the nodes.
2. Establishment of the categories of computational load and the vectors of weights associated with them.
3. Calculation of the priorities or preferences of the processes considering the state of the node (they are calculated in each node for each process).
4. Calculation of the priorities or preferences of the processes to access the shared resources available (calculated in the centralized manager of shared resources) and determination of the order and to which process the resources will be allocated.

Each of the steps above is described below.

In Fig. 2, there is a list of the necessary steps to obtain the final global priorities to assign the resources (***DSAF***, Distributed Systems Assignment Function)

Fig. 2. Steps to obtain the ***DSAF*** and ***ODSAF*** functions.

Calculation of the current computational load of the nodes

To obtain an indicator of the current computational load of each node, different criteria can be adopted; in this proposal the criteria will be the percentage of CPU usage, the percentage of memory usage and the percentage of use of input / output operations, as will be seen in the example.

The computational load of each node will be calculated as follows:
Establishment of the number of criteria to determine the load of the nodes:
$card(\{criteria\}) = c$

Establishment of the criteria that apply (may differ from one node to another):
criteria = $\{c_{ij}\}$ with $i = 1, \ldots, n$ (number of nodes in the distributed system) y $j = 1, \ldots, c$ (maximum number of criteria for each node).

Eventually, all nodes could use the same set of criteria.

Calculation of the computational load of each node:
$load_i = (value(c_{i1}) + \ldots + value(c_{ic})) / c$ with $i = 1, \ldots, n$

Establishment of the categories of computational load and of the vectors of weights associated thereto

Different criteria can be adopted to establish the current computational load categories of each node; in this proposal the categories will be: High (if the load is more than 70%), Medium (if the load is between 40% and 70% inclusive) and Low (if the load is less than 40%), as you will see in the example.

Establishment of the number of categories to determine the load of the nodes:
$card(\{categories\}) = a$

Establishment of the categories that apply (they may differ from one node to another):
categories = $\{cat_{ij}\}$ with $i = 1, \ldots, n$ (number of nodes in the distributed system) and $j = 1, \ldots, a$ (maximum number of categories for each node).

Eventually all nodes could use the same set of categories.

In order to establish the vectors of weights associated with the current computational load categories of each node, different criteria can be adopted; in this proposal, the criteria will be: number of processes in the node, percentage of CPU usage, percentage of memory usage, percentage of virtual memory usage, process priority (process priority in the node where it is executed), memory overhead (additional memory that will require the requested resource to be available, if the data is available), processor overhead (additional processor use that will require the requested resource if the data is available), and input / output overhead (input / additional output that will require to arrange the requested resource, if the data is available), as will be seen in the example.

Establishment of the number of criteria to determine the priority or preference that will be granted in each node according to its load to each order of a shared resource made by each process:
$card(\{critpref\}) = e$

Establishment of the criteria that apply (same for all nodes):
criteria for preferences = $\{cp_{ij}\}$ with $i = 1, \ldots, a$ (number of categories of computational load) and $j = 1, \ldots, e$ (maximum number of criteria).

Eventually, all nodes could use different sets of criteria applicable to the different categories of computational load; in this proposal and as will be seen in the example, the same criteria are used for all nodes.

First, the categories to indicate the load of the nodes and the criteria that will be applied to evaluate the priority to be given to each request of resources of each process are determined. Then the values corresponding to the criteria that constitute the vectors of weights for the different categories of load are established.

Establishment of vectors of weights (same for all nodes):
weights = $\{w_{ij}\}$ con $i = 1, \ldots, a$ (categories number of computational load) y $j = 1, \ldots,$ e (maximum number of criteria).

The assignment of weights to the different criteria will be a function of previously performed statistical studies about the distributed system; there will then be a weight assignment function to the criteria for constituting the weight vectors of each load category:
$w_{ij} = norm(function(cp_{ij}))$ con $i = 1, \ldots, a$ (numbers of category) y $j = 1, \ldots, e$ (numbers of criteria); *norm* indicates that the values must be normalized (in the range of 0 to 1 inclusive) and with the constraints that the sum of the elements of a vector of weights must give 1:
$\Sigma \{w_{ij}\} = 1$ with $j = 1, \ldots, e$ for each constant i.

This means that the sum of the weights assigned to the different criteria will be 1 for each of the categories, or equally, that the sum of elements of the vector of weights of each category is 1.

Calculation of the priorities or preferences of the processes taking into account the status of the node (they are calculated in each node for each process and could be called nodal priorities)

These priorities are calculated at each node for each resource request originated in each process; the calculation considers the corresponding weight vector according to the current load of the node and the vector of the values granted by the node according to the evaluation criteria of the request. The range of values is between 0 and 1, where a value close to 0 means that the related criterion will contribute little to the calculation of the priority of the request, while a value close to 1 means otherwise. Thus a node can influence a request for a resource by a process according to its state and the additional impact or burden that would mean assigning the requested resource to the requesting process, e.g., if accessing the request means increasing the memory usage and the node has little memory available, then it could assign to that criterion a value close to 0, in turn, if the additional processor consumption is considered low and the CPU usage of node is little, then a value close to 1 would be assigned to that criterion.

The valuation vectors that will be applied for each request of a resource by a process, according to the criteria established for

the determination of the priority that in each case and moment will fix the node in which the request occurs, are the following:
valuations (r_{ij} p_{kl}) = {cp_m} con i = 1, …, n (node where the resource resides), j = 1, …, r (resource on node i), k = 1, …, n (node where the process resides), l = 1, …, p (process at node k) and m = 1, …, e (valuation criteria of the requirement priority).

To sum up, the nodal priority (to be calculated at the node where the request occurs) of a process to access a given resource (which can be at any node) is calculated by the scalar product of the mentioned vectors:
nodal priority (r_{ij} p_{kl}) = Σ w_{om} * cp_m indicating the weights vector according to the load of the node, keeping the other subscripts the meanings explained above.

Calculation of process priorities or preferences to access available shares (it is calculated in the centralized manager of the shared resources). In addition, determining the order in which the resources will be allocated and to which process each resource will be allocated

At this stage, the nodal priorities calculated in the previous stage are considered for each requirement of access to resources by the processes. The global or final priorities must be calculated from these nodal priorities, that is, with what priority, or in what order, the requested resources will be provided and to which processes the allocation will be made. The requirements that cannot be attended because they result in low priorities, will be considered again in the next iteration of the method.

Next, it is necessary to calculate the vector of final weights that will be used in the process of aggregation to determine the order or priority of access to the resources.
final weights = {wf_{kl}} con k = 1, …, n (number of nodes) y l = 1, …, p (Maximum number of processes per node), where np is the number of processes in the system and $prgi$ is the priority of the process group to which the process belongs (explained in the previous section).

The next step is to normalize the newly obtained weights by dividing each by the sum of all of them.

Thus a normalized weight vector (in the range of 0 to 1 inclusive) is obtained and with the restriction that the sum of the elements of the vector must give 1:
Σ {nwf_{kl}} = 1 with k = 1, …, n (number of nodes) and l = 1, …, p (maximum number of processes per node).

The nodal priorities taken row by row for each resource will be scalar multiplied by the normalized final weight vector. In this way it is possible to obtain the final global access priorities of each process to each resource. It is indicated below how the order or priority with which the resources will be allocated is obtained and to which process each one will be assigned.

overall final priority (r_{ij} p_{kl}) = nwf_{kl} * p_{kl} with r_{ij} indicating the resource j of node i, p_{kl} the process l of node k and the product of the overall final priority of the process to access such resource. The greater of these products made for the different processes in relation to the same resource will indicate which of the processes will have access to the resource.

The addition of all these products in relation to the same resource will indicate the priority that will have that resource to be assigned, in relation to the other resources that will also have to be assigned. This is what will be called Distributed Systems Assignment Function (***DSAF***), as shown in Fig. 3:
DSAF(r_{ij}) = Σ nwf_{kl} * p_{kl} = resource allocation priority r_{ij}.

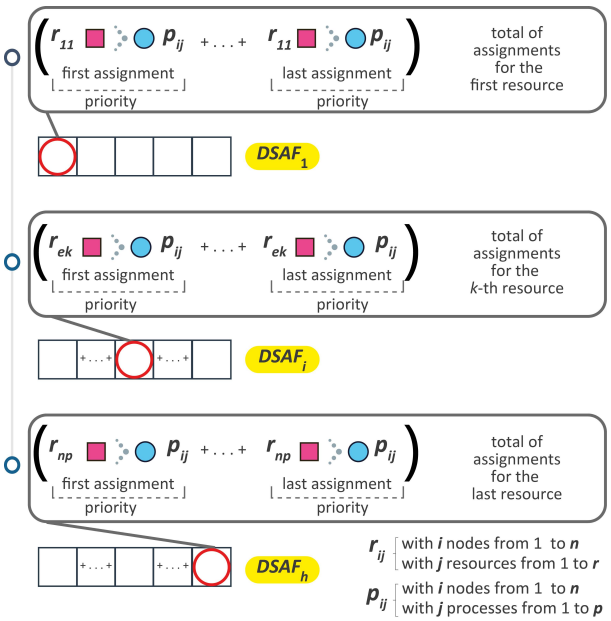

Fig. 3. Calculation of the ***DSAF*** of each process.

The representation of the p_{ij} indicate the processes (whose first subindex represents the node where it is and the second subindex represents the process number itself) that are assigned to the r_{11}, r_{ek}, r_{np} resources (whose first subindex represents the node where it is and the second subindex represents the resource number itself), in each round.

By calculating the ***DSAF*** for all resources a vector will be obtained, and by ordering its elements from highest to lowest, will be obtained the vector Ordered Distributed Systems Assignment Function (***ODSAF***), which will included the order of priority of resource allocation, as shown in Fig 4.

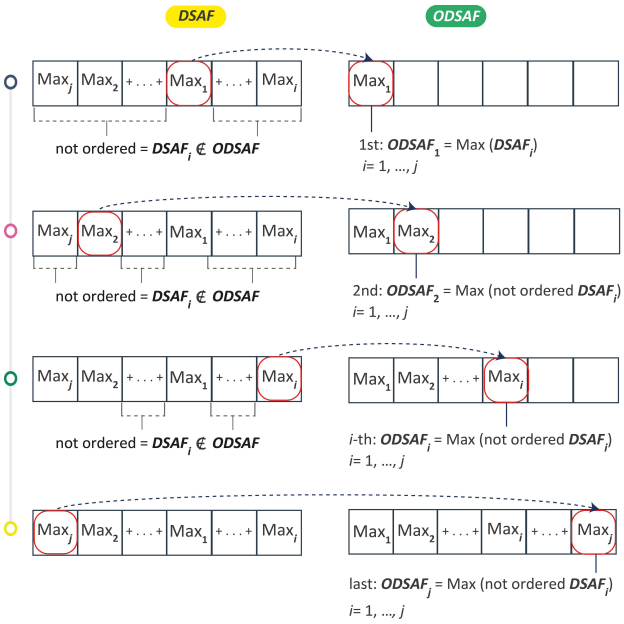

Fig. 4. Resource allocation priority ordered from highest to lowest (***ODSAF***).

Max from 1 to *j*, indicates the priority values of the resources found in the ***DSAF*** vector. *Not ordered* indicates the resources that are not yet loaded in the ***ODSAF*** vector. In addition, as already indicated, the largest of the products $nwf_{kl} * p_{kl}$ for each resource will indicate the process to which the resource will be assigned, as shown in.

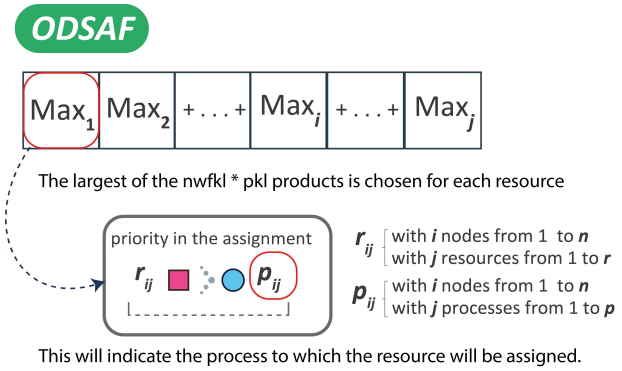

Fig. 5. Selection of the process to which the resource is to be assigned.

Considerations for Aggregation Operations

The characteristics of the aggregation operations described allow to consider that the proposed method belongs to the family of aggregation operators Neat-OWA, which are characterized by the following:

The definition of OWA operators indicates that

$$f(a_1,a_2,\ldots,a_n)=\sum_{j=1}^{n} w_j \cdot b_j \tag{1}$$

where b_j is the *jth* highest value of the a_n, with the restriction for weights to satisfy

$$w_i \in [0,1] \tag{2}$$

$$\sum_{i=1}^{n} w_i = 1 \tag{3}$$

For the Neat OWA operator family the weights will be calculated according to the elements that are added, or more exactly of the values to be added orderly, the b_j, maintaining conditions (2) and (3). In this case the weights are $w_i=f_i(b_1,\ldots,b_n)$, defining the operator

$$F(a_1,\ldots a_n)=\sum_{i} f_i(b_1,\ldots,b_n)\cdot b_i \tag{4}$$

For this family, where the weights depend on the aggregation, the satisfaction of all properties of OWA operators is not required.

In addition, in order to be able to assert that an aggregation operator is *neat*, the final aggregation value needs to be independent of the order of the values. $A=(a_1,\ldots,a_n)$ being the entries to add, $B=(b_1,\ldots,b_n)$ being the ordered entries and $C=(c_1,\ldots,c_n)= Perm(a_1,\ldots,a_n)$ a permutation of the entries. An OWA operator is defined as *neat* if

$$F(a_1,a_2,\ldots,a_n)=\sum_{i=1}^{n} w_i \cdot b_i \tag{5}$$

It produces the same result for any assignment $C = B$.
One of the characteristics to be pointed out by Neat OWA operators is that the values to be added need not be sorted out for their process. This implies that the formulation of a neat operator can be defined by directly using the arguments instead of the orderly elements.

In the proposed aggregation operator, the weights are calculated according to context values. From this context arise the values to be aggregated.

4. EXAMPLE AND DISCUSSION OF RESULTS

This section will explain in detail an example of application of the proposed aggregation operator, based on [17]. The distributed processing system has three nodes:
nodes = {1, 2, 3}

The processes running on the nodes are as follows: three processes on node 1, five processes on node 2 and seven processes on node 3.

processes = {p_{ij}} with *i* indicating the node y *j* indicating the process.

Several processes are independent and others constitute groups of cooperative processes. In this example four groups will be considered.

The number of processes in each group indicates the cardinality of the group and is represented as follows:
card = {*card* (g_i)} = {3, 2, 2, 3} with *i* indicating the group.

The priority of the groups of processes will be considered the cardinality of each group and is represented as follows:
prg = {*prgi* = *card*(g_i)} = {3, 2, 2, 3} with *i* indicating the group.

The shared resources available in the nodes are as follows: three resources in node 1, four resources in node 2 and three resources in node 3.

resources = {p_{ij}} with *i* indicating the node y *j* indicating the process.

Each of the calculation steps will now be described. Calculation of the current computational load of the nodes to obtain an indicator of the current computational load of each node, the same three criteria will be adopted in the three nodes:
card ({*criteria*}) = 3
criteria = {% CPU usage, % of memory usage, % use of input / output operations}.

Establishment of the categories of computational load and of the vectors of weights associated thereto.

In this proposal, the categories will be the same for all nodes: High (if the load is greater than 70%), Medium (if the load is between 40% and 70% inclusive) and Low (if the load is less than 40%).

card ({*categories*}) = 3

categories = {High, Medium, Low}

To establish the weight vectors associated with the current computational load categories of each node, the following criteria will be used for all nodes and for all load categories: Number of processes in the node,% CPU usage,% memory usage,% virtual memory usage, process priority (process priority in the node where it is executed), memory overhead (additional memory that will require (additional processor use that will require the requested resource to be available, if the data is available) and input / output overhead (additional input / output that will require the requested resource to be available) , if the data is available).

card ({*critpref*}) = 8

criteria for preferences = {Node of processes in the node, % of CPU usage, % of memory usage, % of virtual memory usage, process priority, memory overhead, processor overload, input / output overhead}. Next, the values corresponding to the criteria must be established, constituting the vectors of weights for the different categories of load, which will be the same for all nodes. The sum of the weights assigned to the different criteria is 1 for each of the categories, i.e. the sum of elements of the vector of weights of each category is 1.

Calculation of the priorities or preferences of the processes taking the status of the node into account (they are calculated in each node for each process and could be called nodal priorities)

The valuation vectors are applied for each requirement of a resource made by a process, according to the criteria established for the determination of the priority that in each case and moment fixes the node in which the request occurs; each vector of evaluations of each requirement is scalar multiplied by the vector of weights corresponding to the current load category of the node to obtain the priority according to each criterion and the nodal priority granted to each requirement.

Calculation of the priorities or preferences of the processes to access the shared resources available (calculated in the centralized resource manager) and determining the order in which the resources will be allocated and which process each resource will be assigned.

From the nodal priorities, the global or final priorities must be calculated, that is, with what priority, in what order, the requested resources will be awarded and to which processes such grant will be made. Next it is necessary to calculate the vector of final weights that will be used in the final process of aggregation to determine the order or priority of access to the resources. The nodal priorities taken row by row, that is, for each resource, will be scalar multiplied by the normalized final weight vector to obtain the final global access priorities of each process to each resource, and from there, the order or priority with which the resources will be allocated and to which process each one will be assigned. The greatest of these products made for the different processes in relation to the same resource will indicate which of the processes will have access to the resource (in the case of ties the process identified with the smallest number could be chosen). The addition of all these products in relation to the same resource will indicate the priority of such resource to be assigned. This is the Distributed Systems Assignment Function (***FASD***).

The next step is to reiterate the procedure, but removing from the requests for resources the assignments already made; it should also be taken into account that the allocated resources will be available when the processes have released them and can therefore be assigned to other processes. In this way, all the requests for resources of all the processes have been taken care of, respecting the mutual exclusion and the priorities of the processes, the nodal priorities and the final priorities, according to the scenario presented in [17].

5. CONCLUSIONS

The proposed model makes it possible for the distributed system to self-regulate repeatedly according to the local state of the *n* nodes, resulting in an update of their local states, as a consequence of the evolution of their respective processes and the decisions of access to resources: the distributed system in whose groups of processes access to critical resources is executed, produces access decisions to resources that modify the state of the system and readjusts it repetitively, also guaranteeing the mutual exclusion in access to the shared resources, indicating the priority of granting access to each resource and the process to which it is assigned. This process is repeated as long as there are processes that request access to shared resources.

For future work, it is planned to develop variants of the proposed method considering other aggregation operators (especially the OWA family) and the possibility of being used by a resource manager shared (instead of centralized as in the proposed method).

It is considered to develop decision models from the cognitive point of view for decision making in groups of processes, contemplating the principles of cybernetics of second order, in the context of complex systems of self-regulation, which transcend the traditional approach of computer science considering the possibility of imputation of missing data, for example, as a consequence of problems in communications between processes, and fuzzyfication of variables to support situations where it is not possible or convenient to express exact values.

It is planned to develop a simulator in which the different possible scenarios are considered in order to allow the system to predict, compare and optimise the behaviour of its simulated processes in a very short time without the cost or risk of carrying them out, making it possible to represent the processes, resources and nodes in a dynamic model. With the help of the corresponding computer support, the simulation model will allow the ability to consider complex interrelated tasks and project them through the realization of many alternative combinations in a matter of seconds. In addition, the interaction of resources with processes will result in a large number of scenarios and possible results impossible to cover and assess without the help of a computer simulation model.

Acknowledgement. This work has been supported by the Project: “Decision models and aggregation operators for process management in distributed systems”, code 16F001 of Northeastern National University (Argentine).

6. REFERENCES

[1] A. S. Tanenbaum, **Sistemas Operativos Distribuidos**, Prentice - Hall Hispanoamericana S.A., México, 1996.

[2] A. S. Tanenbaum, **Sistemas Operativos Modernos**, 3ra. Edición, Pearson Educación S. A., México, 2009.

[3] D. Agrawal, A. El Abbadi, "An Efficient and Fault-Tolerant Solution of Distributed Mutual Exclusion", **ACM Trans. on Computer Systems**, Vol. 9, pp. 1-20, USA, 1991.

[4] G. Ricart, A. K. Agrawala, "An Optimal Algorithm for Mutual Exclusion in Computer Networks", **Commun. of the ACM**, Vol. 24, pp. 9-17, 1981.

[5] G. Cao, M. Singhal, "A Delay-Optimal Quorum-Based Mutual Exclusion Algorithm for Distributed Systems", **IEEE Transactions on Parallel and Distributed Systems**, Vol. 12, No. 12, pp. 1256-1268, USA, 2001.

[6] S. Lodha, A. Kshemkalyani, "A Fair Distributed Mutual Exclusion Algorithm", **IEEE Trans. Parallel and Distributed Systems,** Vol. 11, No. 6, pp. 537-549, USA, 2000.

[7] W. Stallings, **Sistemas Operativos**, 5ta. Edición. Pearson Educación S.A., España, 2005.

[8] G. Andrews, **Foundation of Multithreaded, Parallel, and Distributed Programming,** Reading, MA: Addison-Wesley, USA, 2000.

[9] R. Guerraoui, L. Rodrigues, **Introduction to Reliable Distributed Programming**, Berlin, Springer-Verlag, 2006.

[10] N. Lynch, **Distributed Algorithms**, Morgan Kauffman, San Mateo, CA, USA, 1996.

[11] G. Tel, **Introduction to Distributed Algorithms**, Cambridge University Press, 2nd ed. Cambridge, UK, 2000.

[12] H. Attiya, J. Welch, **Distributed Computing Fundamentals, Simulations, and Advanced Topics**, John Wiley, 2nd ed., New York, USA, 2004.

[13] P. Saxena, J. Rai, "A Survey of Permission-based Distributed Mutual Exclusion Algorithms", **Computer Standards and Interfaces**, Vol. 25, No. 2, pp. 159-181, 2003.

[14] M. Velazquez, "A Survey of Distributed Mutual Exclusion Algorithms", **Technical Report** CS-93-116, University of Colorado at Boulder, 1993.

[15] S.-D. Lin, Q. Lian, M. Chen, Z. Zhang, "A Practical Distributed Mutual Exclusion Protocol in Dynamic Peer-to-Peer Systems", **Proc. Third International Workshop on Peer-to-Peer Systems**, Vol. 3279 of Lect. Notes Compo Sc., (La Jolla, CA), Springer-Verlag, Berlin, 2004.

[16] L. Sha, R. Rajkumar, J. P. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization", **Computers, IEEE Transactions on**, Vol. 39, No. 9, pp. 1175–1185, 1990.

[17] D. L. La Red Martínez, "Aggregation Operator for Assignment of Resources in Distributed Systems", **International Journal of Advanced Computer Science and Applications**, Vol. 8, No. 10, (IJACSA), pp. 406-419, ISSN N° 2156-5570, The Science and Information (SAI) Organization, England, U.K, 2017.