TOWARDS A PREDICTIVE LOAD BALANCING METHOD BASED ON MULTIPLES RESOURCES

GARCIA, J.L., PICCOLI, M.F., GALLARD R.
Proyecto UNSL-338403[1]
Departamento de Informática
Universidad Nacional de San Luis (UNSL)
Ejército de los Andes 950 - Local 106
5700 - San Luis, Argentina.
E-mail:{jlg,mpiccoli rgallard}@.unsl.edu.ar
Phone: + 54 652 20823
Fax : +54 652 30224

Abstract

Processors load unbalance in distributed systems is one of the main problems, because it involves system performance degradation. Load balance algorithms try to improve the system global performance through migration of processes, but they present also an additional problem, known as *instability*: It happens when processes spend an excessive amount of time migrating among different system nodes. In order to diminish this cost without affecting the mean system response time, load balancing algorithms based on different strategies have been proposed. Multiple Resources Predictive Load Balance Strategy (MRPLBS), is a new predictive, dynamic and nonpreemptive strategy for balancing multiple resources. The predictive approach is based on estimations computed as *weighed exponential averages* of the load of each node in the system.

This paper presents MRPLBS' system architecture and its performance and system a comparison on different scenarios against Random Load Balancing. The number of requirements, the mean response time, the number of failed migrations and the percentage of acceptance are shown.

Keywords: Distributed systems, Load Balancing Strategies, Multiple Resources Metric, Mean Response Time, Migrations.

[1] The Research Group is supported by the Universidad Nacional de San Luis and the ANPCYT (National Agency to Promote Science and Technology).

1. INTRODUCTION

A distributed system offers the potential necessary to work with shared resources [13]. In these systems it is possible, and generally it occurs, that some nodes are overloaded while others are underloaded. This unbalanced condition in the system load produces a poor system global performance[18].

Load balancing algorithms aim to improve system global performance by evenly distributing the load. This can be done at processes arrival time. Process allocation depends on the system load. Load balancing strategies can be classified considering *when* and *where* the system load is determined[7][16]. MRPLBS is characterised by the following attributes:

- *Dynamic*: The information about the system load is periodically collected and updated.
- *Deterministic*: The choice of a target machine is a unique function of system load, with no randomness in the decision.
- *Non preemptive*: jobs that start running cannot be interrupted and moved to other machines until completion. In other words, the job execution will be considered atomic from the load balancing viewpoint.
- *decentralised* because decision making is distributed between system nodes.

Many researchers have been working in load balance and load sharing [1], [2], [3], [5], [6], [8], [9], [10]. Some of them include evolutionary techniques [4], [11],[12],[14] and most of them consider only one resource to determine the load.

A fundamental problem with conventional load functions is that they completely ignores resources others than CPU[7],[16]. Therefore, while it may be reasonable predict the performance of purely CPU-bound tasks, its utility is questionable for tasks that also intensively use other resources: memory, disk, etc. Consequently, determination of system load by measuring several resources load state will improve the load balancing system performance.

MRPLBS, is a strategy that, based on a multiple resources metric, tries to predict The system nodes more inclined to accept migration requests. The prediction is based on an estimation of the present conditions of each node and their past behaviour. In the following sections the MRPLBS strategy and its architecture are shown, and results are analysed. .

2. MRPLBS DESIGN

Figure 1 shows the internal structure of MRPLBS. A short description of the main modules, their components and functions is presented below:

Initialisation Module , executes only once at node bootstrapping, it is in charge of activating the three central system modules which in their turn manage local or external requests and load balancing.

Local_Process_Adm, it is responsible, at local process creation time, to verify the local node balancing state by comparing the current load with a prefixed threshold to determine overloading and the memory requirements of process with the memory size of node. Depending on comparison results some of the following actions will be undertaken:

- If L ≤ *Threshold* and there are enough memory then the task will be locally executed and a child process, ***Local_Execution_Server*** will be activated.
- Otherwise, invokes ***Balance_Module***, who indicates if the new process can be migrated and to which node. If a receiving node can be found then the process is migrated. On the contrary, local execution will be accepted and behaves as above explained.

- If L ≤ *Threshold* and there are enough memory then the task will be locally executed and a child process, ***Local_Execution_Server*** will be activated.
- Otherwise, invokes ***Balance_Module***, who indicates if the new process can be migrated and to which node. If a receiving node can be found then the process is migrated. On the contrary, local execution will be accepted and behaves as above explained.

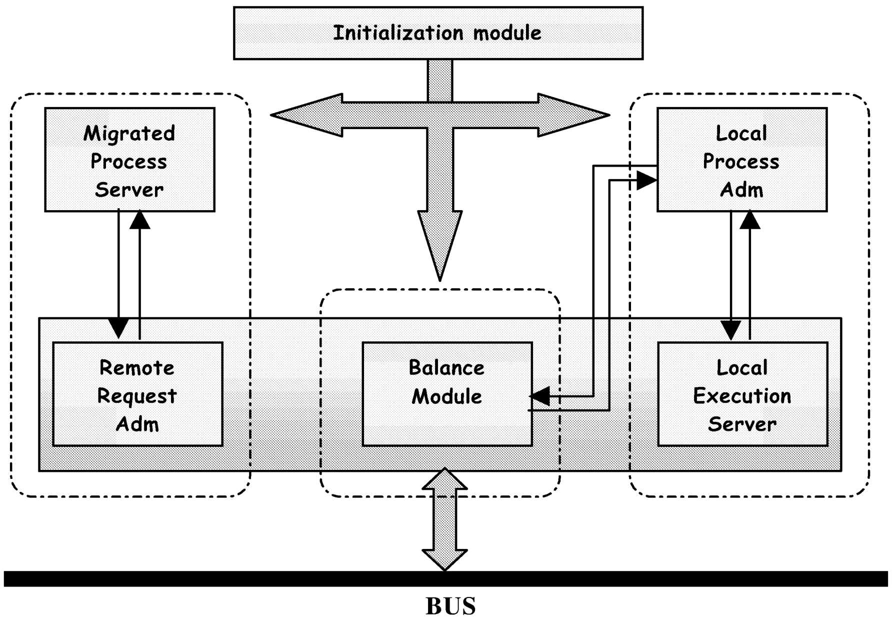

Figure 1: Internal structure of MRPLBS

Local_Execution_Server, if the process is executed locally, then it is in charge of execution, otherwise it ask to migrate the process to a receiving node, and blocks itself waiting for reply related to the remote execution completion.
Remote_Request_Adm, has two main tasks:

- Replies migration requests from other nodes, giving information about local loading state (number of waiting processes, or ready queue length). Also, when an immigrated process finishes execution in the local node, informs about this event to the (original) sending node.
- Activates a child server process when a remote process from an overloaded node arrives and the local node is idle or in a low loading condition.

Migrated_Process_Serv, executes locally an immigrated process and, on completion, signals the event to ***Remote_Request_Adm***.
Balance_Module, this module implements the load balancing strategy. In Section 2.1, there is a detailed description of this module.

2.1. Load Balancing Module: Internal Structure

In order to carry out the predictive strategies, the load balancing module is composed by three demons processes: *Decision module*, *Spread* and *Refresh* and a data structure to load information system: *Global Load State Table*. Figure 2 shows the structure and the relation between modules.

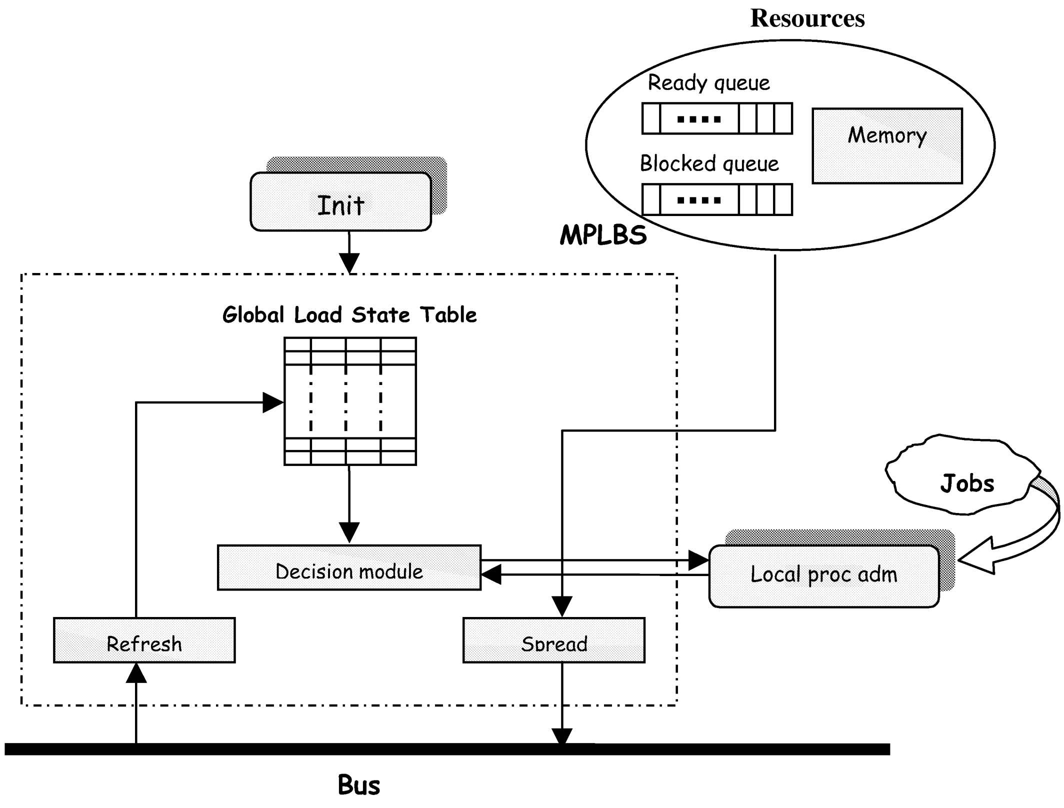

Figure 2: Structure of the Balance_Module

The *spreader* process disseminates local metric information (CPU queue and Blocked queue lengths) to a subgroup of nodes in the system. Periodically, it has to check the load node state and spread it when either metric values changed or the elapsed time without changes is considerable. Its objective is to maintain updated information on the system state. The memory size is communicated once, when the node bootstrapping. The *refresh* process maintains, in a global information table at each node location, the metric values of all nodes in the system. Values in that table are weighted exponential average values of collected metric (load) values. The *decision module* is in charge of deciding where to send the incoming process. It is activated by request from Local_Process_Adm when a job incomes. *Global Load State Table* is a data structure, which contains one entry per node or processor in the system. Each entry is a 4-tuple

<NId, WEA-CPU, MS, BQL>

where NId is the Node Identifier: WEA-CPU is the weighed exponential average of the ready queue length. In others words it maintains a "*history*" of how the ready queue in each node evolves. MS is the Memory Size of the node and BQL is the Blocked Queue Length. Initially it contains the current blocked queue length but finally due to incremental work, it will have a "*history*" of the blocked queue.

3. DESCRIPTION OF THE MRPLBS STRATEGY

A load balancing policy consists of three components:

1. *The information policy,* which specifies the amount of load the job information available to the job placement decision maker(s), and the way by which the information is distributed.
2. *The transfer policy,* which determines the eligibility of a job for load balancing based on the job and the loading state of hosts.
3. *The placement policy,* which decides for eligible jobs, the hosts to which the jobs should be transferred.

The above three component policies of a load balancing algorithm are not isolated from each other, but interact in various ways. In MRPLBS, they interact in this manner: the placement policy utilises the load index information supplied by the information policy, and acts only on the jobs selected by the transfer policy.

As an improvement on PLBS proposed in [5] and APLBS proposed in [6], MRPLBS attempts to reduce the number of requests from an overloaded node. The applied policy by MRPLBS is explained by in the next subsection.

3.1 Information Policy

This policy involves three important tasks: to determine the system load, to distribute this information and to establish the job requirements. For establishing a reliable current metric value, Weighted Exponential Average [17] permits to predict a value on the basis of values appeared during certain elapsed time. In our model, due to the dynamic behaviour of the system, it is appropriate to give a larger weight to the recent history. For an arbitrary processor, the predicted working load is given by:

$$L_{n+1} = \alpha S_n + (1 - \alpha) L_n , \quad 0 \leq \alpha \leq 1 \qquad (1)$$

where:
L_{n+1}: predicted processor load for the next migration request.
S_n : effective processor load at the n^{th} sample interval.

The parameter α allows controlling the relative weight to be given to immediate or old history. If α is equal to zero the recent history is considered irrelevant (present conditions are transient), otherwise if α is equal to one then recent history is important and past history obsolete. In this way the past and recent history is maintained and weighted for each system component and when a migration is needed from an overloaded node, requests are addressed to those candidates more inclined to accept the request. The corresponding side effect is lower number of request, high acceptance hit ratio and therefore an enhanced performance of the distributed system is achieved. In order to decrease the communication traffic, typically generated by load balancing schemes, exchange of information relative to load level in a node is controlled by the *Spreader* process, local to each node, and then broadcasted to a random select subset of nodes each time. In order to determine the load state, MRPLBS uses two thresholds O and U to define the load state L of a node

$L>O$	$\Rightarrow$ overloaded,
$U = L = O$	$\Rightarrow$ medium load
$L < U$	$\Rightarrow$ underloaded

Each node maintains its loading state metric, which is determined at fixed time intervals. In our case this metric is related to the number of processes in the ready queue and number of processes in the blocked queue. Each job, when it arrives to the system, specifies its memory requirement and its type, CPU or I/O bound. The success of good decisions depends on accuracy of the job information and the information in the Global Load State Table.

3.2. Transfer Policy

When a new process incomes, it might be a candidate to migrate if at least one of these two conditions is true:

- The local node is overloaded, L > O,
- Its memory requirement is greater than the available memory in the node, and in consequence, if locally executed, its response time will increase because memory swapping is introduced.

3.3. Placement Policy

To selected the node for job execution, MRPLBS applies the next philosophy: *"Predict the current working load of a given processor when a migration request is necessary"*. When a migration is necessary, migration requests are sent to a subset of nodes. A node belongs to this subset if its probable low loading state suggests that it might accept the request. This information is retrieved from the global load state table. The administrator fixes the subset size. After requesting migration of a process to candidate nodes, the requester has the answer from each of the targets. Each node answer to the requester with its actual information load and MRPLBS selects one of then. The favourite is that node whose conditions are the best to execute the job. If no nodes are founded, therefore the process will be executed locally.

4. PERFORMANCE ANALYSIS OF MRPLBS

PARASOL [15] is a computer systems modelling tool, which is oriented to modern distributed or parallel computer systems. It was used to simulated the distributed system and the implemented MRPLBS.
The system parameters were defined as follow: Systems with 30, 40 and 50 nodes were simulated, but we show only 30 nodes results. Each node executed concurrent processes under a *round-robin* policy maintaining a *ready* queue and a *blocked* queue. The network topology was *Ethernet.* The network transfer rate was of 10 Mbits. Diverse process types were considered according to their needs of CPU or I/O. Their service time were variable for all processes, considering fix service time like special case of it. Each process when arriving at the system specifies its memory requirements and the run time and type are determined at random. Process arrivals follow a Poisson distribution of mean λ. Experiments were carried out on five different scenarios, using 1/λ as mean interarrival time with λ = 0.1, 0.2, ..., 0.9, 1. A simulation was completed when 50,000 processes where executed in the network nodes. The next indicated scenarios were used in the simulation.

- **Scenario 1:** 60% of the nodes are receiving processes with equal arrival rate λ while in the remaining nodes does not occur any arrival. This schema allows simulation of a clearly unbalanced situation.
- **Scenario 2:** 40% of the nodes are receiving processes with low arrival rate (λ) and the other 60% with more high arrival rate (2 λ).
- **Scenario 3:** Each node has its own arrival rate λ.
- **Scenario 4 :** Every node has its own arrival rate λ, which varies randomly each 200 tics.
- **Scenario 5:** Each node has its own arrival rate λ , which varies randomly each 2000 tics.

Scenario 1 attempted to reflect a real situation, which frequently occurs, where the workload is not evenly distributed. Scenarios 2, 3, 4 and 5, are similar in the sense that arrivals occur in every node, but scenario 4 and 5 differs reflecting time depending arrival rates as often occurs in a computer network.

4.1. Results

Results were obtained from the application of MRPLBS in the five scenarios with different processes types. These results are compared with the Random load balancing strategy for the 30 nodes case. We choose some representative instances of the diverse scenarios. But in general the same trend is observed in any scenario with any number of nodes in the network. In every case the relevant performance variables were the issued number of requests (NR), number of failed migration (FM), mean response time (MRT) and acceptance hit ratio (HR). Table 1 show general results of the simulation. Figure 3 show average values of the performance variables throughout all type of processes and scenarios.

Scenario	*Type Proc.*		**Random**				**MRPBLS**			
Number	**CPU**	**I/O**	**NR**	**FM**	**MRT**	**HR**	**NR**	**FM**	**MRT**	**HR**
1	100%		1370	159	55.61	88.39	1297	122	55.00	90.59
		100%	2182	452	103.18	79.28	2064	382	100.26	81.49
	50%	50%	1331	178	77.54	86.63	1243	177	75.38	85.76
Average			1627.66	263	78.77	84.76	1534.66	227	76.88	85.94
2	100%		2406	254	64.24	89.44	2279	140	63.42	93.86
		100%	2882	459	107.32	84.07	2721	346	104.94	87.28
	50%	50%	2053	248	85.16	87.92	2045	143	83.75	93.00
Average			2447	320.33	85.57	87.14	2348.33	209.66	78.15	91.38
3	100%		2212	307	58.22	86.12	2789	112	61.04	95.98
		100%	3403	803	106.79	76.40	3393	347	103.32	89.77
	50%	50%	2108	378	82.35	82.07	2534	148	82.26	94.16
Average			2574.33	494.66	82.45	81.53	2905.33	202.33	82.21	93.30
4	100%		2105	285	63.81	86.46	2249	217	64.31	90.35
		100%	2980	745	108.94	75.00	2664	408	105.72	84.68
	50%	50%	2038	351	85.80	82.78	1850	155	83.89	95.40
Average			2374.33	460.33	86.18	81.41	2254.33	260	84.64	90.14
5	100%		3339	581	66.64	82.60	2652	172	62.84	93.51
		100%	4113	1085	111.17	73.62	3212	413	105.49	87.14
	50%	50%	2995	583	89.32	80.53	2396	124	83.75	94.82
Average			3482.33	749.66	89.04	78.92	2753.33	236.33	84.02	91.82

Table 1. Number of Requests, Accepted Number of Requests, Failed Migrations and Acceptance Hit Ratio for each process type, under each scenario, for the 30 nodes case.

A rapid look on table 1 clearly shows that in general MPBLS outperforms the Random load Balancing strategy on each of the considered performance variables. There is only one exception in scenario 1 for the mixed process type (50% CPU, 50% I/O) and HR variable.

The advantages in performance of MPBLS over Random is more evident when we look at its average behaviour (see figure 3). Specially in FM where this number is reduced in a range from 13.7% (in scenario 1) to 68.5% (in scenario 5). This is the consequence of the ability of the predictive approach to make requests to the nodes that are more inclined to accept a migration.

Similar behaviour was found for networks of 40 and 50 nodes. In this last case due to the existence of much more nodes FM decreases under both strategies but for Random they are about 12.23% while in MRPLBS they are about 5.86%.

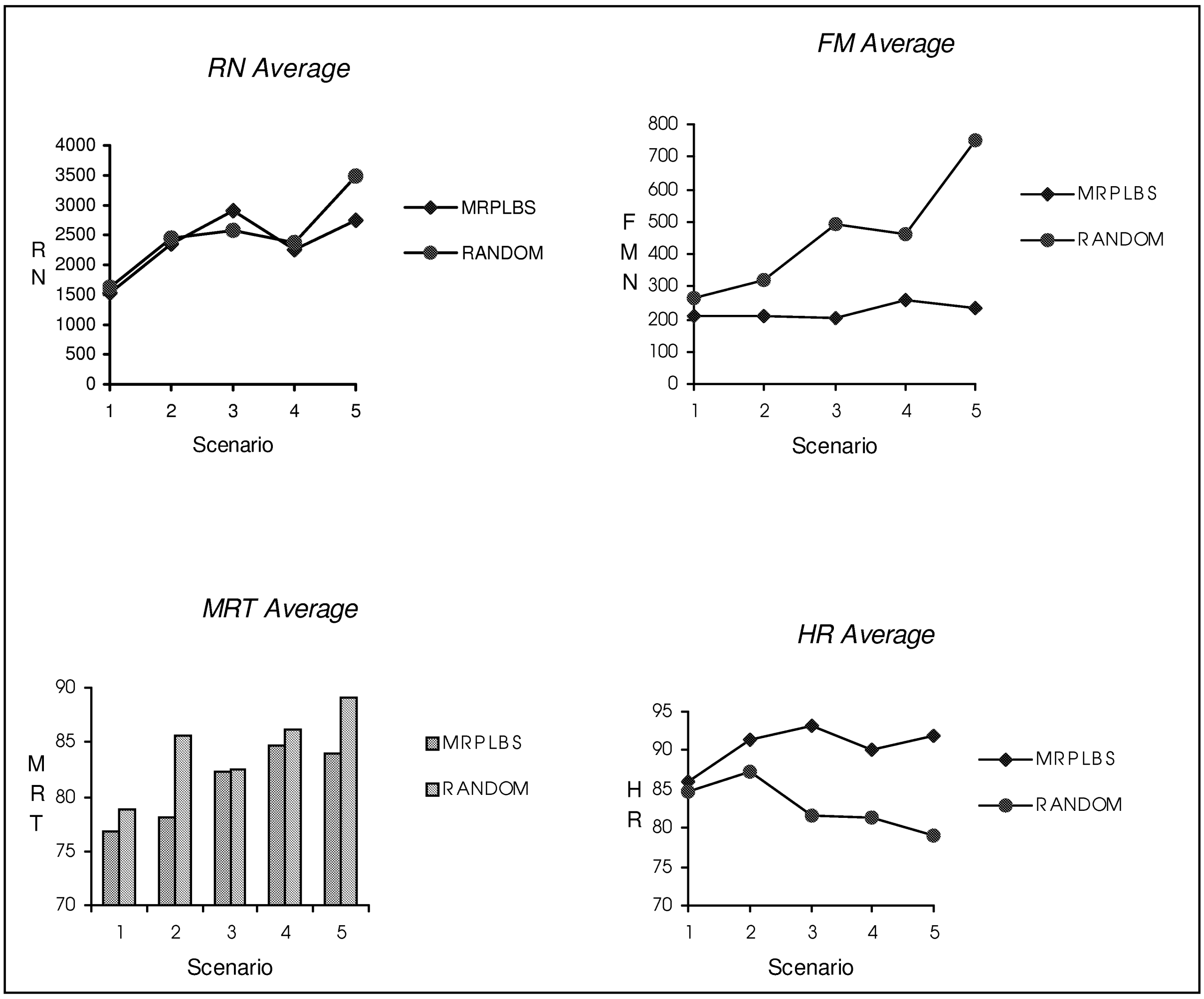

Fig. 3. Mean values of Number of Requests, Accepted Number of Requests, Failed Migrations and Acceptance Hit Ratio, under each scenario, for the 30 nodes case.

5. CONCLUSIONS

In this report we have discussed the results of comparing two load balancing strategies: Random and Multiple Resources Predictive Load Balance Strategy. Both approaches considered not only processor but also memory requirements. It was again shown that, when an predictive approach is used, better results than those obtained with traditional algorithms can be expected. The present work was faced bearing in mind not only the migration of processes but also a substantial decrease in the number of requests a node has to do before migration takes place by choosing a subset of nodes more inclined to accept these requests. In other words, we searched for a reduction in traffic in the system and a high hit ratio when a load balancing strategy is to be applied. To achieve this, the Load balancing strategy was enriched by incorporating a prediction function to the load balancing module. In this way, by using the knowledge gathered by each node, the number of nodes to be consulted when overload occurs was drastically reduced. The ability of MRPLBS to predict was clearly shown in the final number of migrations failed which is significantly lesser than under the Random strategy. The experimental results obtained through simulation give a clear indication of the efficiency of the proposed hybrid strategy.

6. ACKNOWLEDGEMENTS

We acknowledge the co-operation of the project group for providing new ideas and constructive criticisms. Also to the Universidad Nacional de San Luis and the ANPCYT from which we receive continuous support.

7. BIBLIOGRAPHY

[1] Arredondo, D., Errecalde, M., Flores, S., Piccoli, F., Printista, M., Gallard, R. “Embedded Intelligent Assistance for Load Distribution and Balancing ”. Ninth IASTED International Conference on Parallel and Distributed Computing and Systems. PP 188-195, ISBN 0-88986-240-0, ISSN 1027-2658. October 1997. Washington D.C., USA

[2] Arredondo, D., Errecalde, M., Flores, S., Piccoli, F., Printista, M., Gallard, R. “Load Distribution and Balancing Support in a Workstation-based distributed Systems” . Operating Systems Review, Vol 31, Nº 2, pp 46-59, ISNN 0-163-5980. April 1997.

[3] Chu W., Holloway L., Lan M., Efe K. - “Task Allocation in Distributed Data Processing” - Distributed Computing: Concepts and Implementations, pp 109-119, Addison Wesley - 1984.

[4] Esquivel S., Leguizamón G., Gallard R. - “A Hybrid Strategy for Load Balancing in Distributed Systems Environments”, Proceedings of the Fourth IEEE International Conference on Evolutionary Computation (ICEC’97), pp. 127 -132, Indianapolis, USA, April 1997.

[5] Esquivel S., Pereira C. and Gallard R. - “Predictive Load Balancing for a Workstation Distributed System, Proceedings de la International Conference on Applied Informatic, Garmish, Germany, February 1998.

[6] Esquivel S., Pereira C. and Gallard R. – “An Efficient Adaptive Predictive Load Balancing Method For Distributed Systems”. IV Congreso Argentino de Ciencias de la Computación en la Universidad del Comahue. Vol. 1, pp. 345-365. Octubre 1998. Neuquen, Argentina.

[7] Ferrari D., - “Study of Load Indices for Load Balancing Schemes”, University of California, Berkeley, 1985.

[8] Flores S., Piccoli F., Printista M., Gallard R. “A User Supervised Load-Balancing Scheduler”. Segundo Congreso Argentino de Ciencias de la Computación. PP 109-120. Noviembre 1996. San Luis, Argentina

[9] Garcia, J.L., Piccoli, F., Gallard, R., "Un Método de Balance de Carga Predictivo-Múltiple Recursos para Sistemas Distribuidos". VI Congreso Internacional de Ingeniería Informática, ICIE Y2K. PP 123-131. Abril 26-28, 2000.

[10] Jun C., Li X., Zhong-xiu S. - "A Model for Intelligent Task Scheduling in Large Distributed System", ACM Press, Operating Systems Review, Vol.24, N° 4, October 1990.

[11] Kremien O. and Kramer J. - "Methodical Analisys of Adaptative Load Sharing Algorithms", IEEE Transaction on Paralell and Distributed Systems, V. 3, N° 6, págs. 747-760, November 1992.

[12] Kremin O., Kramer J. and Magee J. - "Scalable, Adaptive Load Sharing for Distributed Systems - IEEE Parallel and Distributed Technology, pp. 62-70. August 1993.

[13] Mullender S. - "Distributed Systems" - Addison Wesley, 2da. edition, 1995.

[14] Munetomo M., Takai Y., y Sato Y. - " A Genetic Approach to Dynamic Load Balancing in a Distributed Computing System", Proceeding of the First IEEE. Conference on Evolutionary Computation, June 1994, Vol. 1, pp.419-421.

[15] Neilson J., - " Parasol User's Manual ", School Of Computer Science, Carleton University, Canada.

[16] Panjak, M. - " Automated Learning of Load Balancing Strategies for a Distributed Computer Systems" PhD. Thesis, University of Illinois at Urbana, Chapaign, 1993.

[17] Stallings W. - "Operating Systems" - MacMillan publishing Company, New York, 1992.

[18] Stone, H., Bokhari, S.- "Control of Distributed Processes" - Distributed Computing: Concepts and Implementations, pp. 109-119, Addison Wesley - 1984.