# A USER SUPERVISED LOAD-BALANCING SCHEDULER

FLORES S., PICCOLI F., PRINTISTA M.[1],
GALLARD R.[2],

Grupo de Interés en Sistemas de Computación[3]
Departamento de Informática
Universidad Nacional de San Luis
Ejército de los Andes 950 - Local 106
5700 - San Luis
Argentina
E-mail:rgallard@inter2.unsl.edu.ar
rgallard@unsl.edu.ar
Phone: 54++ 652 20823
Fax : 54++ 652 30224

## ABSTRACT

In distributed systems load distribution and balancing are primary functions directed to system performance improvement and user content. In order to decide where the incoming workload have to be allocated, maintaining a reasonable balance in the system, a number of non trivial tasks will be necessary.

As a basement for further automatic system decision, this paper propose a user supervised processor allocation scheduler and shows which information should be collected, when and how to collect and disseminate it to support the user decision.

**KEY WORDS :** Distributed Systems, job scheduling, load distribution, load balancing.

[1]Members of the Research Group.

[2]Full Professor at the Informatics Department. Head of the Research Group.

[3]The Research Group is supported by the Universidad Nacional de San Luis, the CONICET (Science and Technology Research Council) and the SSID (Subsecretary of Informatics and Development)

## 1. INTRODUCTION

Load balancing attempts to optimize overall system performance by judicious allocation on processes to one of a set of available processors. This can be achieved by allocating processes at the time they are created or by re-allocating them while they are running (process migration).
In a previous work [1], some design and implementation considerations to build a system to support non migrating load balancing in a network of workstations were outlined.
On designing this support system the following assumptions, limitations and expectations were presupposed:

- Homogeneity of computer architectures and operating systems. Nodes differ only in processing speed and storage capacity.
- Only those tasks created by the user-system interaction will be involved with the load balancing system.
- Tasks are independent and can be created in any system site.
- It is expected a significant reduction in execution time for SPMD (Single Program Multiple Data) CPU intensive tasks.
- Global system state information will be collected periodically and maintained by each node in the network (decentralised policy for information gathering). Based on this information the node where the task was created is in charge of deciding the final allocation for that task. This corresponds to an scheme of global transfer policies with non migrating Load Balancing Scheduling, sender initiated.

Incremental system development was divided into three stages:

At the first stage, load balancing is attempted through user supervised load distribution. The user receives assistance from the system, which supplies global information about load conditions in the network and a facility for remote execution, and then decides the execution site for his job.
During this stage the system collects statistical data about the values of outstanding variables under diverse user-selected distribution policies based on diverse metrics approaches. These data will be used as a knowledge base to study policies to be designed, and then used by the system through intelligent tools (such as neural nets or genetic algorithms), for automatic decision of load distribution.

The second stage adds to the system a new subsystem; the *Decision Subsystem* which replaces the user on deciding the most appropriate node for the user requested program execution. (See figure 1).

The final stage in system development propose to provide an specification language to declare those tasks subject to parallel execution.

For this purpose some interfaces will provide specification files which will be the input of an interpreter. This interpreter will convert the sentences of the specification language into independent requests for the decision subsystem.

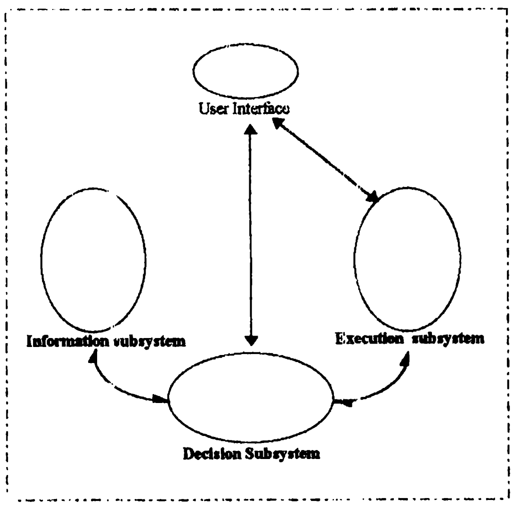

**Figure 1: Architecture of the Load Balancing System.**

As a part of the first system development stage above mentioned, this paper show design aspects on the subsystems devoted to allocate a site for incoming job execution under user supervision. The main objective of this stage is to build a basement for appropriate decision criteria according some quantitative measure of workload.

## 2. LOAD-BALANCING SCHEDULING

Performance degradation due to *workload unbalancing*, ordinarily conveys to lower *global throughput* and increased *response time* for arriving jobs [4], [5], [8], [9], [10], [11], [13], [15]. This three different aspects are intimately correlated and consequently system performance can be improved by enhancing only one of them [16]. We can devise a mechanism to determine a convenient, not necessarily optimum, execution site for an arriving task. Under this vision the problem becomes a scheduling problem. For solving it, proper and updated information about system state and arriving tasks attributes should be necessary.
The scheduler, which is installed in each node of the distributed system, assists the user by providing him with information related to current global load conditions. Basically, this dynamically updated information consists of a set of load levels corresponding to an index (measure of) load in each network node. A node load condition is described by a number of parameters related to the state of some resources. Based on this information, knowing task attributes and by choosing some criteria the user finally decides a suitable execution site for the incoming task.

The *Load-Balancing Scheduler* is part of the *Load Balancing System* shown in figure 1. The scheduler, as shown in figure 2, consists of two subsystems; *Information Subsystem* and *User Interface* and their main tasks are to provide updated system condition information to the user, interaction with the user and the conversion of user directives in a request to the *Execution Subsystem*. The later is in charge of process re-allocation (migration) to another site, if required. In particular the *User Interface* is divided into two layers: an external one, the ***user interaction*** and an internal one, the ***information requester***.
Now, we will concentrate on some relevant design and implementation aspects of the scheduler.

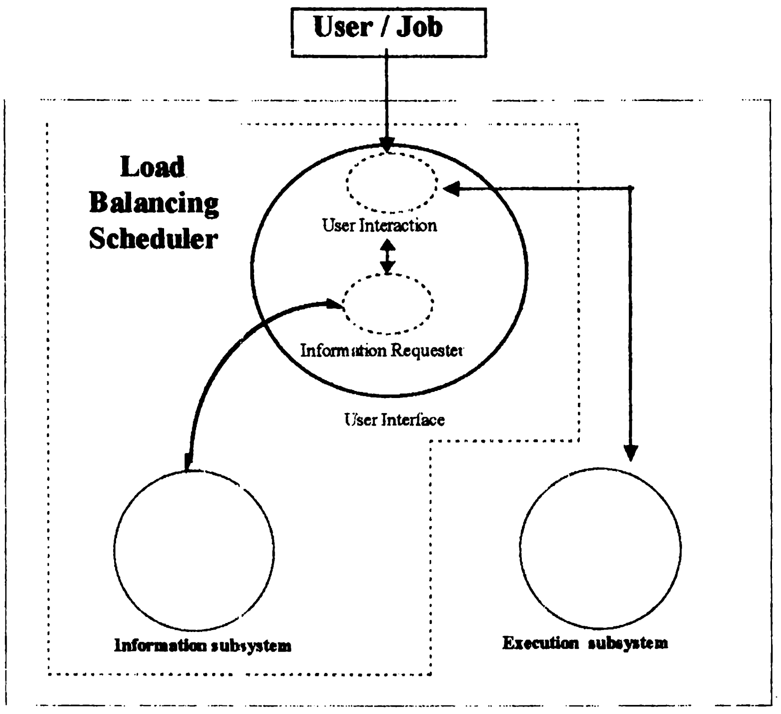

**Figure 2: Architecture of the Load Balancing System for the Initial Development Stage.**

The scheduler, by means of a decentralised policy for information gathering, collects and maintains information about load conditions corresponding to each node in the network. The information is available at user will (typically at new task arrival time) who finally, using an scheme of global transfer policies, decides the execution node. When adopting this design for the scheduler, three main questions arouse:

- *What* to collect ? (Section 2.1)
- *When* a request (or an exchange) of information should be initiated ? (Section 2.2)
- *How* to collect information from each network node ? (Section 2.3)

In the following sections we explain our approach to answer these questions.

## 2.1. PERFORMANCE METRICS SELECTION

Many performance parameters could be considered to decide what to collect when seeking for knowledge of system load. It is important to define criteria for selecting these parameters (user or system sensitive?). Conventional approaches use as a referential metric the number of processes waiting, locally, for their CPUs (ready queue length, *rql*). Such approaches are attracting in view of their simplicity but show to be inadequate in some not unusual circumstances, for example:

- When there exists a diversity of CPUs processing power in the network; slow CPUs with few processes in their ready queue will be overqualified when contrasted against a faster CPU with a longer queue .
- When some attributes about process behaviour (CPU bound, I/O bound, etc.) are known; it have been shown that no correlation exists between the mean number of processes in the ready queue and the response time of an arriving I/O bound job. This parameter could be a useful indication only for CPU bound jobs.

Conclusions from previous research ( [6], [12], [16], [17], [18]) observe:

- The mean number of processes waiting for their CPUs cannot be generally applied to any job type and any load condition in a node. The problem with this metric arise from ignoring conditions on other system resources.
- An extremely sensitive correlation exists between I/O traffic in the node and the response time for I/O bound jobs.
- A correlation exists between free memory size and the number of processes in ready queue. Therefore, a correlation exists between free memory size and response time for a job.
- A correlation exists between the number of I/O packets being transferred by the network interface and the response time for a job.
- As expected, it was shown that a metric based on a single resource results not suitable for the general problem.

Consequently, combining *rql* and job attributes seems to be insufficient for helping the user to decide in the general case and hence a ***multiple resource metric*** (***mrm***) was devised.
This metric supplies an index load for each considered resource, and a copy of the set of all *mrm* nodes is available, for each node, in a MRM data base .

On deciding which resources and metrics should be included, issues such as *reliable representation* of load condition, *computing overhead* and *independence of control* were extensively considered.
For our computing environments in the network of workstations, we defined *mrm* consisting of a representative group of metrics related to basic resources, as follows:

*mrm* = <*CAPP, CMC, DT, NT*> where,

- *CAPP*, current available processing power in the node; is a relation between the declared processing speed (MIPS) and the number of processes currently serviced by the local CPU.
- *CMC*, current memory capacity, is the free memory size in the node.
- *DT*, disk transfers, is the amount of data transferred on each disk (internal I/O traffic).
- *NT*, network transfers, is the number of I/O packets on each network interface (external I/O traffic).

Load condition of each node in the network is provided to the system by a metric such as the one above described. Each parameter value in the 4-tuple is dynamically computed. A dedicated process *Local_Load_Evaluator* is in charge of recording changes in the loading state of resources.
Depending on system objectives and user skill, decision making could be based on diverse and complex criteria such as user process requirements *upr* (turnaround time, response time, reliability, etc), user processes attributes *upa* (CPU or I/O bound, memory required, execution time, etc.), system performance *sp* (throughput, CPU usage, load balancing, etc.) and nodes available capability *mrm* (CPU availability, communication links, memory and other devices capabilities, etc.).

As a simple example, let us assume that we are in a system condition where N available nodes differ essentially in Current Memory Capacity (*CMC*) and MIPS provided. Due to system dynamics they also differ in Current Available Processing Power (*CAPP*)[1]. User processes are CPU intensive tasks and their main requirements are Memory Required (*MR*) and Desired Response Time (*DRT*).
*MC*, *CAPP*, *MR* and *DRT*, are each classified or divided into a number of levels (high, medium, low, or more levels). Other processes requirements on system resources, such as access to secondary storage, can be equally fulfilled by any of the available nodes and there will not be network transfers (except for initial process migration, which is equally costly for every node).
The user can adopt a simple allocation criteria such as the one sketched below;

- Having *MR* best fit satisfied, satisfy *DRT* by allocating the process to the best fitted node (the one with minimum *CAPP* fulfilling process requirement). In case of equal *CAPP* values for more than one node then node selection is random.

[1] Where CAPP is defined as the ratio of the specific value for its CPU speed (MIPS) to the number of current allocated processes.

- If the strategy also considers the situation where idle nodes exists, then; if for two or more nodes *CAPP* is equal, and some of these nodes is in idle[1] state then the process is allocated to that idle node.

Using this simple criteria, load balancing is dynamically attempted, by allocating to the process that processor with best fitted remaining processor power. Under equality, when idle processors exists these are preferred to avoid context switch overhead.

This ought to be done following our primary objective in this first stage; gathering data to help building a mapping function δ, for automatic decision of an execution site, such that

$$\delta\ (upr, upa, sp, MRM) = node_i \quad \text{where, } MRM = \{mrm_i\}\ (1 \leq i \leq N)$$

## 2.2. DATA GATHERING STRATEGIES

In order to avoid falling into an unwanted overhead and its involved side effects, it was necessary to decide when and how frequently to collect data.
Aiming to provide recently updated information on system load condition we proposed a *periodic-and-event-based* strategy. The basic actions within this strategy are:

- Every certain prefixed interval, a timer interrupt occurs and, each node broadcasts[2] its own state (*mrm*) to the rest of the nodes. Individual intervals are shifted along the global time to reduce collision rate.
- Each time a job arrives to the system , the entry node checks (ageing control procedure) if every item in MRM was updated within that prefixed interval. If some node update is out of this threshold then a request, for updated *mrm* values, will be sent to that node.

Simple data structures (to describe the multiple resource metric for each network node) and procedures (for periodic and event based updates) will be necessary.

## 2.3. THE SCHEDULER

This section describes how information is collected and how the scheduler components interact. See figure 3.

1 A node is defined as being in an idle state when no user process is running.
2 Message length is small enough to be transmitted in only one packet.

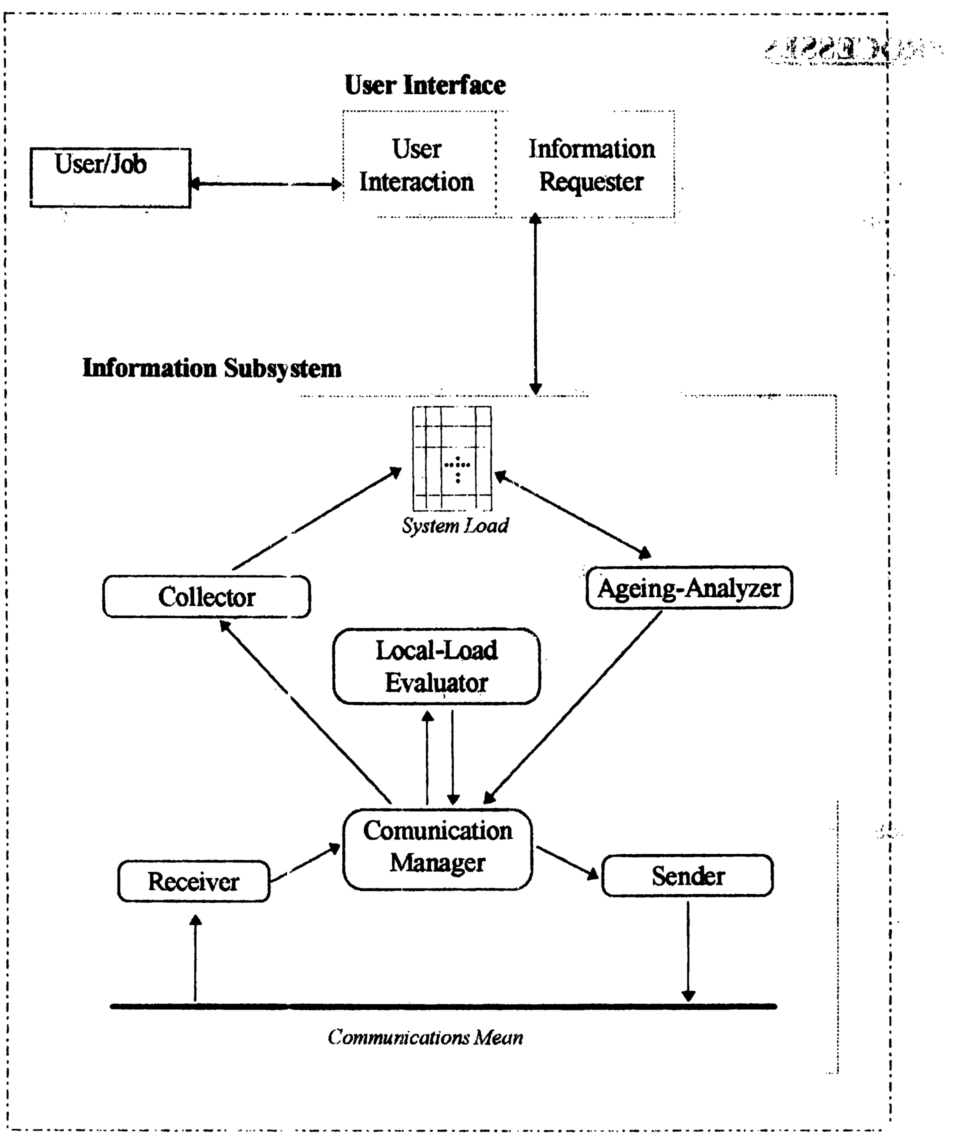

**Figure 3: Load Balancing Scheduler Architecture.**

As we previously said, the *User Interface* is divided into an external layer responsible of interacting with the user and another internal layer in charge of requesting information. On this respect we briefly say that, as in our previous works, this interface is designed under an object oriented approach.

Because we want to explain here *how* information is collected, now we go into some detail about information subsystem design.

Two main groups of processes can be identified here, one is dedicated to communications and the other to maintain loading information.

The first group is composed of *Sender*, *Receiver* and *Communication_Manager* processes and the later consists of *Collector*, *Local_Load_Evaluator* and *Ageing_Analyzer* processes.

Next section describes how these processes intercommunicate.

## 2.3.1. PROCESSES DESCRIPTION

***Sender*** and ***Receiver*** processes interact with the network interface. *Sender* sends a message to a given destination while *Receiver* remains listening for a ***mrm*** information message arrival. Both interact with *Communication_Manager*.

***Communication_Manager*** performs the following functions:

- Packing and unpacking of messages to be delivered and received.

And depending on the type of message received,

- Redirect the ***mrm*** information coming from a remote node to the *Collector* process or
- Request to the *Local_Load_Evaluator,* local ***mrm*** updated information to be sent to a remote node requiring it.

This process becomes active when:

- A message arrives to the node.
- Local ***mrm*** information must be transmitted to one, some or all nodes in the system.
- External ***mrm*** information is old (out of the interval threshold) and must be updated.

Communications are of two types:

- Point-to-point communication, when (local or external) ***mrm*** information update is required.
- Broadcast, at the moment a node is initiated (connected to the network) an initiating message is sent. The content of that message mainly refers to the new node arrival, its local ***mrm*** and an external ***mrm*** request.

Three types of messages are recognized:

- Request of local ***mrm*** information.
- Arrival of external ***mrm*** information.
- Initiating message.

***Local_Load_Evaluator***, evaluates periodically or under request, the loading conditions of node resources. The information produced (***mrm***) is sent to the *Communication_Manager* which pack it in a message and depending on the type of service send the message to the *Collector* process or broadcasts it to the network nodes or send it to an specified destination.

*Collector*, handles data structures supporting information (*mrm*) relative to all system nodes. Messages containing this information and received by the *Collector*, were originated asynchronously in external nodes or periodically in the local node by the *Local Load Evaluator*, and afterwards delivered by the *Communication Manager*.

***Ageing_Analyzer***, when a job arrives to the system this process is activated by *Information_Requester* (which resides in the User Interface area). It is responsible of controlling, for every node in the system, if the time elapsed from the last update of ***mrm*** information is below the prefixed threshold. If an out-of-date condition is detected for the ***mrm*** information corresponding to a system node, then *Ageing Analyzer* request to *Communication Manager* to send a message to that node requiring the up-to-date version.

## 3. SYSTEM TUNING

Some considerations on system performance are illustrated here.
In order the system, for the expected improvements, could be adopted it is necessary to ponder its cost which must not exceed benefits. As an example, costs and benefits related to performance, can be measured in time units (even though, depending on goals, other type of units can be used).
That is to say, that at least the following relation must hold:

$$(T_1 - T_2) / T_3 \leq 1 \quad \text{, where}$$

$T_1$ is the *time* without using the system,
$T_2$ is the *time* when using the system,
$T_3$ is the time needed to execute the procedures for improvements
and *time* is any time variable in the *upr* or *upa* sets.

Components of $T_3$ should be carefully studied and observed while executing the system. Typically, communication overhead is an extremely important factor and thus the following issues are being carefully considered:

- Elapsed time between *mrm* local evaluations (***inter-mrm-evaluation*** time).
- Elapsed time between *mrm* external updates for ageing decision (threshold or *inter-mrm-update* time). This parameter should be very much similar to the expected job interarrival time.
- Alternative interprocess communication primitives and communication protocols.
- Alternative approaches with multicast communication and dedicated metric-storing servers.

## 4. CONCLUSIONS

The User Supervised Load-Balancing Scheduler presented here is part of an incremental development for an automatic Load-Balancing System. This incremental approach aims to learn from system behaviour those relevant factors affecting its performance. Diverse metrics and strategies are being devised and will be evaluated.
Statistical information built during this first stage will be used as input for training embedded intelligent tools in further stages. Such intelligent tools, as neural networks and genetic algorithms, are being implemented to help automatic system decision. On the other side, it is expected that the proposed system architecture will contribute to learn how to better exploit the parallel nature of these tools by a suitable distribution of their software components all over the system.

## 5. ACKNOWLEDGEMENTS

We acknowledge the cooperation of the project group for providing new ideas and constructive criticisms. Also to the UNSL, the CONICET and the Subsecretary of Informatics and Development from which we receive continuous support.

## 6. BIBLIOGRAPHY

[1] Aguirre G., Arredondo D., Errecalde M., Piccoli F., Printista M., Gallard R. - "Load Distribution and Balancing Support in a Local Area Network Based Distributed System" UNSL, Computer Systems Interest Group, Internal Report may 1996.
[2] Berman F., Snyder L. : "On mapping parallel algorithms into parallel architectures" - Parallel and Distributed Computing, pp 439-458, 1987.
[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] Crichlow Joel M. : "An introduction to Distributed and Parallel Computing" - Prentice Hall, 1988.
[5] Colouris G., Dollimore J., Kindberg T.: "Distributed Systems: Concept and Design" - Addison-Wesley, 1994.
[6] Ferrari, D. : "Study of Load Indices for Load Balancing Schemes" - Universidad de California, Berkeley, 1985.
[7] Foster Ian T. : "Designing and Building Parallel Programs" - Addison Wesley, 1995.

[8] Johnson I. D., Harget Alan : "On The Performance Of Load Balancing Algorithms" - PhD Thesis, Aston University, Birmingham, U.K., 1991.

[9] Lewis T.G. "Foundations of Parallel Programming. A Machine-Independent Approach" - IEEE Computer Society Press , 1993.

[10] Lewis T.G. , El-Rewini H. : "Introduction to Parallel Computing" - Prentice Hall, 1992.

[11] McEntire P. L. - O'Reilly J. G. - Larson R. E. - (Editors) : "Distributed Computing: Concepts and Implementations" - Addison Wesley, 1984 .

[12] Mehra P. : "Automated Learning of Load Balancing Strategies for a Distributed Computer System", PhD. Thesis, University of Illinois at Urbana, Chapaign, 1993.

[13] Mullender S. : " Distributed Systems" - Addison Wesley, Segunda Edición , 1995.

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

[15] Tanenbaum A. S. : "Modern Operating Systems". Prentice Hall , 1992.

[16] Watts J. and Taylor S.:"A practical Approach to Dynammic Load Balancing" - California Institute of Technology, Pasadena, 1995.

[17] Zatti S. : "A Multivariable Information Scheme to Balance the Load in a Distributed System". - University of California, Berkeley , 1985.

[18] Zhou S. : " Performance Studies of Dynamic Load Balancing in Distributed Systems" - University of California , Berkeley, 1987.