A SUPPORT FOR REMOTE PROCESS EXECUTION IN A LOAD-BALANCED DISTRIBUTED SYSTEM

ARREDONDO D.,ERRECALDE M. [*]

GALLARD R. [**]

Grupo de Interés en Sistemas de Computación[***]

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

merreca@unsl.edu.ar

darred@unsl.edu.ar

Phone: 54++ 652 20823

Fax : 54++ 652 30224

ABSTRACT

Load distribution and balancing in a workstation-based network includes a number of intricate tasks. Among them, transparent remote process execution is an essential one. This work describes the main problems to be considered when implementing remote process execution and propose a design for an alternative system attempting to solve these problems.

KEYWORDS: Remote execution, load balancing, load distribution, distributed systems.

Members of the UNSL 338403 project.

Full Professor at the Informatics Department. Head of the Research Group.

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

Nowadays most organisations use workstations as computing platforms, typically interconnected to form a local area network. Usually some of this workstations have a heavy load, with degraded performance, while others remain under-utilised or even idle.
The rational and equitable use of this available computational power have been subject of study for many research groups interested on load distribution and balancing [2], [8], [12], [16]. Even though, intuitively, these topics do not seem to present a meaningful complexity their implementation is not a trivial task. Subjects such as management of information related to the load in each node of the network, load metrics to be adopted, decision making mechanisms for choosing a site where to execute a process, support for transparent remote process execution and process migration are some of the relevant aspects that, even by themselves, constitute important research matters.
Within this framework, this paper describes the design of a remote execution subsystem as part of a balanced distributed system [1], attempting to provide reasonable solution to some of these problems.

We begin with a brief description of the components of the load balancing system, then we analyse some of the important problems related to remote execution of processes and finally we show the design of the remote execution subsystem proposed.

2. LOAD BALANCING SYSTEM

For each workstation the load balancing system provides an *Information Subsystem*, a *Decision Subsystem* and an *Execution Subsystem* (See Fig. 1).
The *Information Subsystem* is in charge of collecting and maintaining global information about each available node in the network and its load condition.
The *Decision Subsystem*, by using the output of the information subsystem and data from other sources[1], is responsible to evaluate which is the most convenient node for the execution requested by the user.

[1] Related to job requeriments and attributes, and emanating from user specification, inserted in the object code, or automatically determined.

The *Execution Subsystem* is in charge of executing the user requested program on the workstation designated by the decision subsystem. Also, it must provide a facility to initiate a process in a remote machine. This remote process could interact with the user and access to data files as if it would be executing on its local machine.

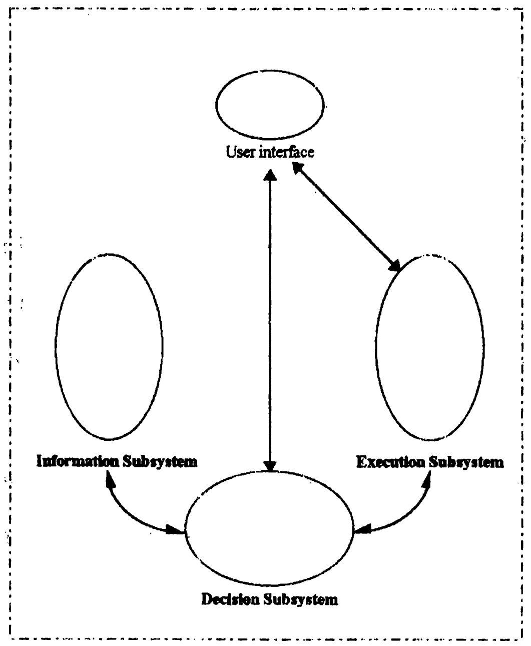

Fig. 1: Architecture of the Load Balancing Subsystem

The user interface is restricted to receive execution requests and to return results. Details such as system load state and selected site for execution are entirely transparent to the user.
In addition, in its first stage, this design is directed to collect statistical data associated to the different alternatives chosen for distribution, in order to determine if the strategies used by the decision subsystem were either adequate or not.

3. <u>ISSUES TO BE CONSIDERED IN REMOTE PROCESS EXECUTION</u>

These topics are, directly or indirectly, related to the fact of configuring the remote process in such a way that its vision of the environment be the same as that of the workstation where initially resided (*the source machine*). In this case we are referring not only to the problem of moving the code but also to the fact that the process have to preserve the same vision of the

file system, working directory and environment variables. Furthermore, during its execution the system calls must behave as in the original site.
This later observation deserves special attention because it happens that some system calls could be executed in remote machines while others could not, and should be redirected to the source machine. For example, system calls requiring information about the state of the site where the process is executing (CPU and memory usage, IP address, etc.) could be satisfied by that remote node but those closely related to the original site, such as those involved with user interaction or access to own devices, ought to be accomplish by the source machine.
In the case of accessing user files (READ, WRITE, OPEN, etc., system calls), some systems solve the problem by working with shared file systems (NFS, AFS, etc.) while others do it by redirecting the system call to the machine where the files are resident [2], [15].
Eventually, migration of active processes is an additional problem to be considered. There exists some distributed operating systems, such as Sprite and Vkernel, which have finely solved this problem, but implementing migration on top of a conventional operating system is cumbersome and benefits obtained do not, sometimes, justify their implementation cost.
Some trade-off solutions allowed active processes migration subjected to forbid the use of interprocess communication mechanisms (sockets, pipes, etc.) or process creation system calls (fork, exec, etc.) [2].

4. REMOTE EXECUTION SUBSYSTEM

In this section we delineate an implementation to provide simple and clear solutions to the problems above mentioned. Also a user interface for specification of tasks to be executed by the system, is proposed.

4.1 PROPOSED IMPLEMENTATION

The following restrictions are imposed to the system:

- Processes to be executed are non migrating processes. Once a process is initiated in a machine it remains there until completion (it is not allowed to move the process to another site).

- Processes created by remote tasks execute in the same machine as their parents.

- The network nodes are homogeneous in hardware and operating system.

- Nodes in the network are arranged as a computing pool. Many users, with equal privileges, can be logged to a single node and nobody is the workstation owner.

Previously we referred to the need of a process common vision for file system, working directory and other resources. In our case we decided to face the problem by using NFS, provided by SUN Microsystems, because it allows system file sharing among many workstations [4],[15].
In this way a user accessing to the system facilities will always see the same files and directories independently of the machine where he is working. The effect is similar as being logged into a diskless workstation connected to a unique file server [15].
By maintaining a single copy of user executable files and data files, costs related to transfer and store data and those problems associated to data consistency are considerably reduced. Furthermore in our design, to minimise the network load, disks that are local to workstations are devoted to paging and allocation of temporary and binary files.
In this stage, two are the main components of the remote execution subsystem;

- demon *cli_exec* residing in the requesting machine
- demon *serv_exec* residing in the target (request executing) machine.

The next section describes how these components interact to facilitate the remote execution of a process.

4.2 INTERACTION BETWEEN SUBSYSTEM COMPONENTS

The *Decision Subsystem* retrieves request specifications from a queue where user requests were stored. (See figure 2). Once decided the fittest node for execution, ask to *cli_exec* remote execution for this job. Then *cli_exec* creates a child process known as the ***shadow process*** (so called because it acts, in the source machine, as a shadow of the remote executed process).

The *shadow process* supplies the information needed for execution (program name, arguments, etc.) to *serv_exec.*

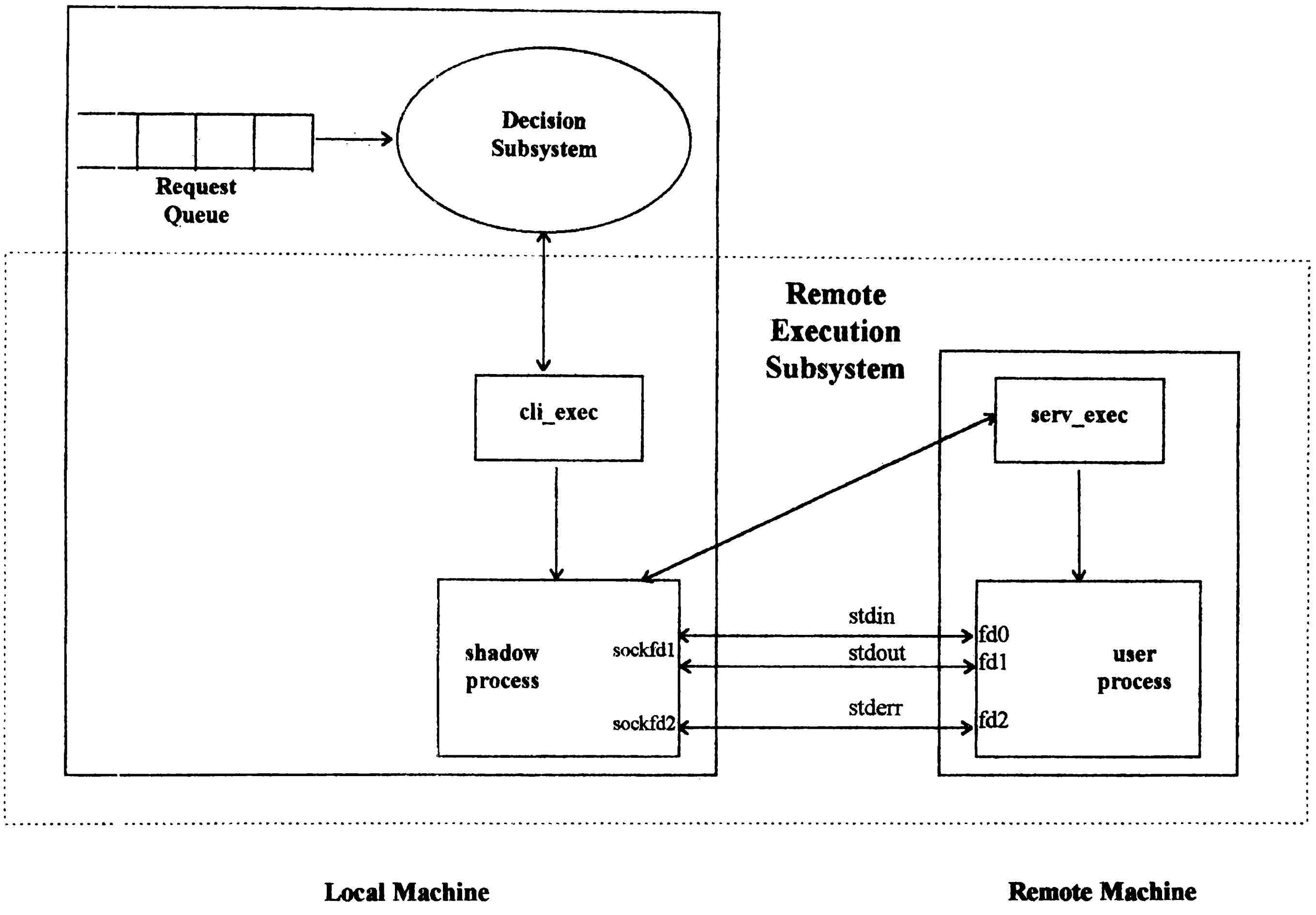

Fig. 2:Interaction between Components of the Remote Execution Subsystem for Interactive Processing

At the other end, *serv_exec,* creates a child for executing the requested process. Two are the possibilities contemplated by the system here: *interactive* and *non interactive processing.*

In the case of *interactive processing* the standard input/output and standard error is redirected to sockets for communication with the corresponding *shadow process* in the source (client) machine. The *shadow process* is attached to a window which is used by the user as a means to communicate with the remote process (via this socket) in the same way as when it is executed locally.

In the case of *non interactive processing* communication is not established via sockets, instead of that the standard input/output and standard error is performed via the user specified files. These files can be directly accessed by the process because it is running in a shared file system environment (NFS, See figure 3)

Process completion is informed to the user through a mail originated by *serv_exec* at the moment it receives the corresponding signal from the *exit()* system call of the terminating process.

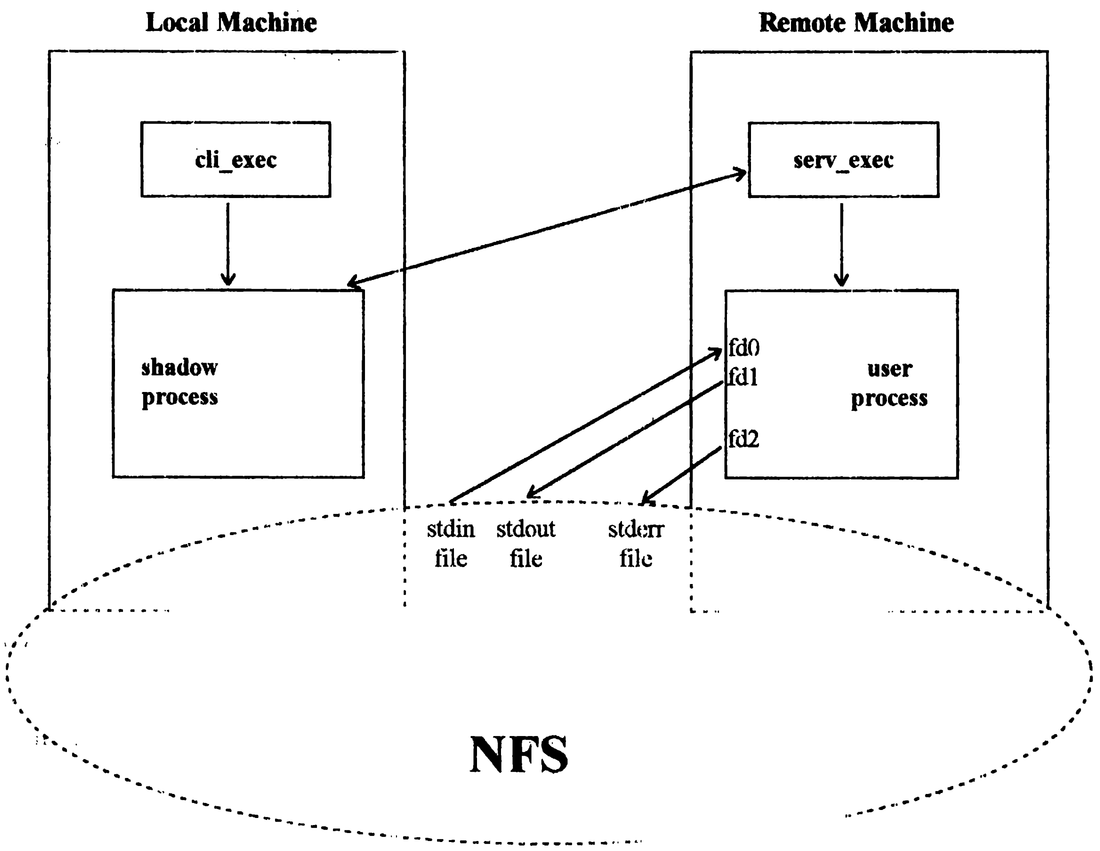

Fig. 3: Interaction between Remote Execution Subsystem Components and Network File System. (non interactive processes)

4.3 <u>REMOTE EXECUTION USER INTERFACE</u>

This section describes how a user can deliver to the system one or more requests for remote execution of his jobs. The simplest way to do it is by entering at command interpreter level:

```
    remote <program name> <arguments>
```

In this case a unique request of the interactive type for remote execution is created (standard input/ouput and error through terminal). When this request is processed the system creates a window in the local machine, on behalf of the remote process, intended for the management of its input/output and error.

This kind of interaction could be satisfactory for certain type of users. But in a network also conceived for intensive CPU tasks it is usual, for long duration tasks, to leave to the system the responsibility of inputting data and collecting results instead of entering data on line and waiting for completion.

In these cases it is appropriate that standard input/output and error be performed by means of user specified files that are defined and analysed separately.

Additionally, it is common to execute a program with different standard files and arguments. This type of more complex requests are performed in this system by providing to the ***remote*** command a ***request specification file***, where diverse execution modes for the jobs can be described:

```
    remote <request specification file>
```

A request specification file is an ordinary text file which contains the following directives:

```
program = <name>
[arguments = <argument list>]

                 { terminal
standard_files = {
                 { user_files        [input = <pathname>]
                                     [output = <pathname>]
                                     [error = <pathname>]
[priority = <priority>]
add_request
```

The ***program*** directive, which is mandatory and specified only once inside the request specification file, identifies the executable file.

The optional ***arguments*** directive, specifies the arguments used when invoking the program determined by the ***program*** directive.

The *standard_files* directive, establishes if the standard input/output and error are performed through the terminal (interactive processing) or by means of user specified files (if any is omitted, then */dev/null* is assumed).

The *priority* directive specifies which is the priority (from -100 to 100, and 0 by default) for the task to be executed. A higher priority value means higher priority.

The *add_request* directive, according to the previously specified directives, creates a request for remote execution and add it to the request queue. This request will be thereafter processed by the remote execution subsystem.

A single program can be repeatedly executed with diverse sets of arguments, standard files and priorities by alternating the *add_request* directive with those before mentioned (*arguments*, *standard_files* and *priority*).

5. CONCLUSIONS

Transparent remote process execution constitute by itself a research area in distributed systems which is intimately related to those systems supporting load balancing.

The design of the remote execution subsystem proposed here is an initial step that in a straightforward and simple manner permits to understand and face most of the relevant problems above mentioned.

Even though restricted, this subsystem will assist to evaluate the remaining load balancing system components on their capability for solving concrete problems.

Once the whole system be integrated and tested it will be contrasted against systems adopting other policies seeking for performance improvements.

6. 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.

7. 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] Bricker A., Litzkow M. J. and Livny M., Condor Technical Summary, Version 4.1b, Technical Report 1069, Computer Sciences Department, University of Wisconsin-Madison, Wisconsin, USA (1992).

[3] Colouris G., Dollimore J., Kindberg T. -Distributed Systems: Concept and Design - Addison-Wesley, 1994.

[4] Comer Douglas E., Stevens David L. - Internetworking with TCP/IP - Vol. III - Prentice Hall.

[5] Crichlow Joel M. - An introduction to Distributed and Parallel Computing - Prentice Hall - 1988.

[6] 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.

[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. , El-Rewini H. - Introduction to Parallel Computing - Prentice Hall - 1992.

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

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

[12] Pankaj M. - Automated Learning of Load Balancing Strategies for a Distributed Computer System -, PhD. Thesis, University of Illinois at Urbana, Chapaign, 1993.

[13] Stevens W. Richard - UNIX Network Programing - Prentice Hall.

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

[15] Tanenbaum Andrew S. - Sistemas Operativos Modernos - Prentice Hall.

[16] Theimer Marvin M., Lantz Keith A. and Cheriton David R. - Preemptable Remote Execution Facilities for the V-System - ACM - 1985.