ENHANCED GENETIC ALGORITHMS FOR THE CLUSTER ALLOCATION PROBLEM IN DISTRIBUTED SYSTEMS

APOLLONI R, MOLINA S., GALLARD R.H.

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: {rubenga, smolina,rgallard}@unsl.edu.ar
PHONE: + 54 2652 420823
Fax : +54 4652 430224

Abstract

In a distributed system, consisting of a set of interconnected local area networks, users migrate to different machines, users invoke different programs and users and programs need distinct data files to satisfy their expectations. Consequently, optimal allocation of program tasks can increase system performance as results of traffic cost reduction between clusters.

The problem of allocating a parallel program in a particular system can be seen as the allocation of the program components in a set of available clusters such that traffic costs are minimized. A total intercluster traffic cost function is the objective function to be optimized and used to evaluate the individuals in a population through the evolutionary process.

This paper discusses possible improvements of the a previous evolutionary approach to the cluster allocation problem, at the light of enhancements found in the evolutionary field itself.
A multiple crossovers per couple (MCPC) approach is applied to the evolutionary algorithm and contrasted against a modified simple genetic algorithm. Details on experiments and results are shown.

Keywords: distributed systems, genetic algorithms, multiple crossovers per couple.

[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

The problem of allocating a program in a particular system node can be divided into two subproblems: (i) allocate the program in a cluster[2] such that traffic costs are minimized and (ii) within a particular cluster choose the node following some load balancing criteria. To solve subproblem (i), in 1992 Borghoff [2] proposed the Individual Program Execution Location Algorithm IPELA, where essentially giving a distribution of data files the best allocation for program execution, minimizing the expected intercluster traffic, is searched. The algorithm uses diverse input data such us the cost for starting a program at some node [12], the dependencies between program and data files [1], separated read and write access costs [11], the impact of I/O activities on the communication costs [8] and the allocation of program and data files [3]. As the number of possible allocations induce high complexity and the model could not be solved to optimality Borghoff reduced the number of combinations by limiting the number of data file replicas and looking for those combinations where the relevant file sets's allocation is varied. This approach reduced complexity. Nevertheless running IPELA implied evaluation of each solution in a large problem space. Extending the Borghoff's individual program framework, we focussed on its application in a parallel program environment confronting the problem by means of an evolutionary approach [6].

2. THE MODEL

Now we briefly explain the nature of the problem. Given a user initiated parallel program, which during execution accesses to a set of files in a distributed system, allocate the program parallel tasks (modules) in order to minimize intercluster traffic (tasks migration, intermodule and module-file communication).

As an example Fig. 1 shows the final allocation of parallel tasks after both substrategies were applied. The parallel program, comprising ten tasks, was residing in workstation W_p of cluster 5, initiation took place from workstation W_i of cluster 1, ten files were distributed throughout the system and the parallel tasks, were first allocated to nine available clusters following the minimization criteria and then to a selected node, within their respective clusters.

In other words, our main objective is to minimize communication time between the clusters where the parallel program components are allocated under the following characteristics:

- The program, which is residing in some cluster, is decomposed in M parallel modules where each of them can be executed in an arbitrary available[3] cluster.
- During execution, parallel modules can communicate each other and access data blocks in files which are resident anywhere in the system.

In our model some data structures referring to programs profile and file distribution are supposed to be available in a Cluster Supervisor Node (CSN) within each cluster.
Assuming that,

[2] Nodes belonging to the same local area network are said to build a cluster.

[3] A cluster is available if it is connected to the internet.

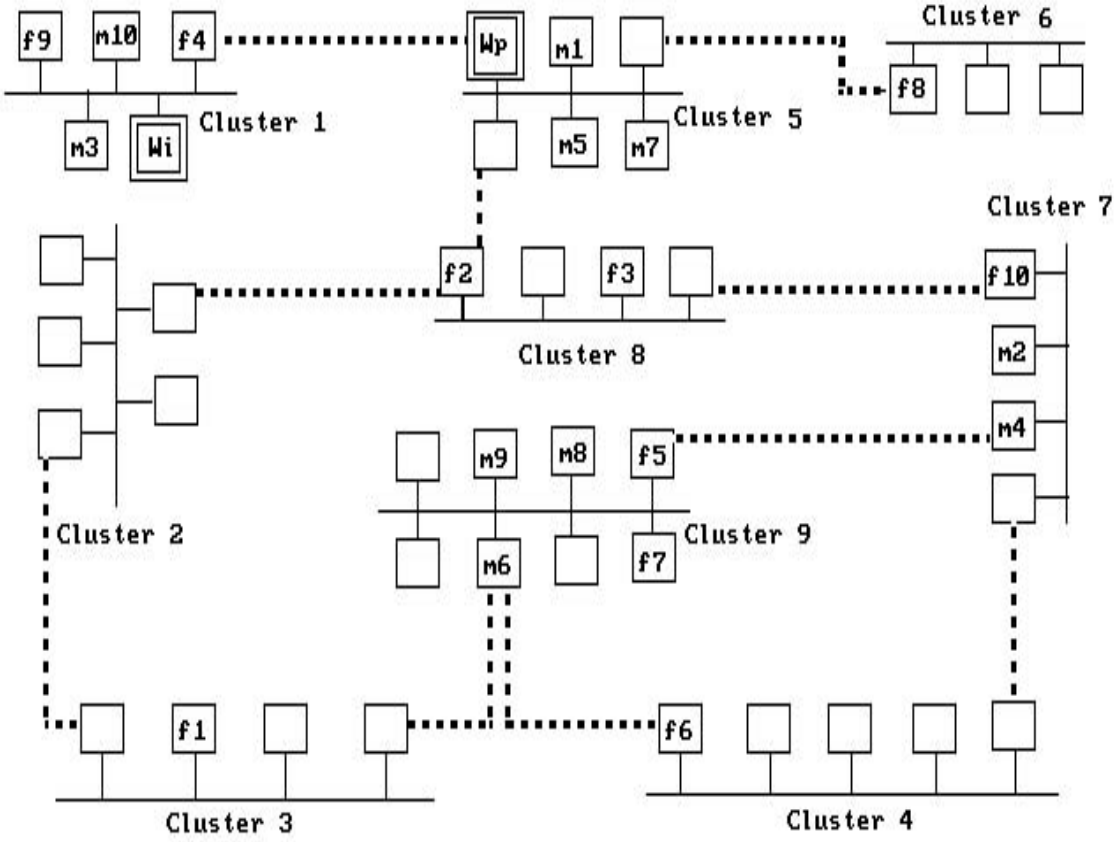


Figure 1. A possible layout of parallel tasks within clusters.

- A parallel program can be divided into *M* migrating modules.
- Each migrated module generates a number of output data blocks (transient or partial results) to the node where it is residing. These blocks are to be transferred to the program residing cluster when computation completes.
- *K* files, distributed throughout the system and involved in the computation, are accessed by the modules.
- The current system state provides *N* available clusters.

then we need the following data structures to hold information associated to each parallel program:

- *M:* Modules. An scalar describing the number of modules comprising the program.
- *CB*: Code Blocks (including executing environment). An *M*-vector indicating the length of code in blocks for each parallel module to be transferred from the program file residing cluster to each executing cluster.
- *MODB:* Module Output Data Blocks. An *M*-vector specifying the number of output data blocks produced by each parallel module to be transferred to the program file residing cluster. $MODB_i$ indicates the number of output data blocks (partial results) produced by module *i* during computation.
- *PODB:* Program Output Data Blocks. A scalar specifying the number of output data blocks, corresponding to global results, to be transferred from the program residing cluster to the initiating cluster.

- *RWDB:* Read/Write Data Blocks. A *KxM* matrix specifying the number of data blocks accessed in a given file by each parallel module during computation. $RWDB_{i,j}$ indicates the number of read/write accesses of module *j* on file *i*.
- *IMC:* InterModule Communication. An *MxM* matrix indicating the intermodule communication measured in data blocks transferred. $IMC_{i,j}$ specifies the number of data blocks transferred from module *i* to module *j* during execution.
- *FD:* File Distribution. A *KxN* matrix indicating the file distribution throughout the clusters. $FD_{i,j}$ = 1 indicates that file *i* is stored in some node of cluster *j*. $FD_{i,j}$ = 0 indicates that file *i* is not stored in any node of cluster *j*.
- *MSP:* Maximum SPeed. An *NxN* matrix specifying the maximum transfer speed between clusters. $MSP_{i,j}$ indicates the maximum transfer speed between cluster *i* and cluster *j*, following some of the interconnections allowed by network topology.

2.1. THE OBJECTIVE FUNCTION : TOTAL INTERCLUSTER TRAFFIC COST (TITC)

If $C_A = \{c_{A1},..., c_{AN}\}$ is the set of available clusters in the network, our objective is to find an execution cluster distribution[4]

$$C_D = <exec_1, ..., exec_M> \text{ where } exec_i \in C_A, 1 \le i \le M$$

to allocate each parallel module in such a way that execution leads to a minimization of intercluster traffic according to the parallel module individual profiles, the current allocation of the program file and the current allocation of involved data files. Our objective function deals with the following partial costs:

- *Initiating Cost (IC):* Includes the cost to handle the user request to run the program, plus migration of parallel tasks to available clusters.
- *Intermodule Communication Cost (IMCC):* Includes the cost to transfer messages and/or data between modules.
- *File Access Cost (FAC):* Which includes read/write accesses from modules to data files.
- *Output Cost (OC):* Includes the cost to transfer results from execution clusters to the initiating cluster.

$$TITC = IC + IMCC + FAC + OC$$

Initiating Cost (IC)

A user can initiate the execution of a program *prog* from any cluster. Let be c_{init} the cluster where the user node resides, and c_{prog} the cluster where the invoked program code resides. The initiation process can be divided into two stages, the first, related to the transfer of a remote command execution request from c_{init} to c_{prog} and the second involved in parallel tasks migration from c_{prog} to each selected

[4] The execution cluster distribution C_D, is a m-vector specifying that module i must be allocated to cluster $exec_i$.

execution cluster according to an allocation distribution vector $C_D=<exec_1, ..., exec_M>$[5], specifying that module *i* must be allocated to cluster $exec_i$.
So, if *RCEDB* is the number of data blocks involved in the remote command execution request, then the first component of IC is given by:

$$IC_1 = \begin{cases} \dfrac{RCEDB}{MSP_{init, c_{prog}}} \; if \; init \neq c_{prog} \\ \\ 0 \;\; otherwise \end{cases}$$

The Cluster Supervisor Node in cluster c_{prog} is responsible, by applying the optimization strategy, of distributing the M modules composing the parallel program, among available clusters. Then the second constituent of *IC* is given by:

$$IC_2 = \sum_{i=1}^{M} \frac{CB_i}{MSP_{c_{prog}, exec_i}}$$

and consequently

$$IC = IC_1 + IC_2$$

Intermodule Communication Cost (IMCC)

Assuming that communication links of similar technology between two clusters exists in either direction, then communication between module *i* and module *j* is measured as the number of blocks transferred from one module to another disregarding direction. This assumption is introduced to simplify the model by using a triangular matrix, without loss of generality. The *Intermodule Communication Cost* is given by the following expression:

$$IMCC = \sum_{i=1}^{M-1} \sum_{j=i+1}^{M} \frac{IMC_{i,j}}{MSP_{exec_i, exec_j}}$$

File Access Cost (FAC)

In our model we assume that a single copy of the files accessed during parallel computation exists in the system. A generalization considering file replication implies slight modification of the objective function and data structures. Access to files typically encompass input data requests and data file modifications. This cost includes transfer of data blocks to be read (write) from (to) the cluster where a file exists to (from) the cluster where the executing module is allocated. *FAC* is given by:

$$FAC = \sum_{i=1}^{K} \sum_{j=1}^{M} \frac{RWDB_{i,j}}{MSP_{exec_j, p}(p: p = 1,.., N \; \wedge \; FD_{i,p} = 1)}$$

[5] $C_D^* = < exec_1^*, ..., exec_M^* >$, the near optimal allocation distribution vector comes as a result of a minimization strategy

Output Cost (OC)

In computing intensive task processing, minimum interaction is advisable for achieving high throughput. In our model we assume that the final output (partial results) from each parallel module is first transferred to c_{prog} (the cluster where the parallel program file is residing). Then after some local computations in c_{prog} global results are transferred from c_{prog} to c_{init} (the cluster where the program was initiated).

Then the *Output Cost* is given by:

$$OC = \sum_{i=1}^{M} \frac{MODB_i}{MSP_{exec_i, c_{prog}}} + \frac{MODB_i}{MSP_{c_{prog}, init}}$$

3. THE EVOLUTIONARY APPROACH

Given m modules and n available clusters all possible allocations must be searched and corresponding costs calculated. The size of the problem space is n^m. A difficult tractable problem depending on n and m. In our model we chose to obtain timely near-optimal allocations instead of out-of-time optimal results. Genetic Algorithms are a valuable tool to face this kind of problems.

3.1. EXPERIMENTS AND RESULTS

The algorithms were applied on many scenarios with diverse number of modules, clusters and files each, and ten series of twenty runs, with diverse values for the corresponding input parameters were accomplished. Here we discuss the case of 10 modules, 9 clusters and 10 files.

We decided to work with an integer representation as a more natural description of the allocation problem. So, a particular position (gene) in the chromosome represented a module identifier which is allocated in the cluster identified by its value (allele). As an example when 10 modules are to be allocated in 9 available clusters, the chromosome of Fig. 2 indicates the following cluster distribution: modules 1 and 7 will be allocated in cluster 3, modules 2 and 5 in cluster 1, modules 8 and 9 in cluster 9, module 3 in cluster 4, module 4 in cluster 5, module 6 in cluster 6 and module 10 in cluster 7. Clusters 2 and 8 will not allocate any module.

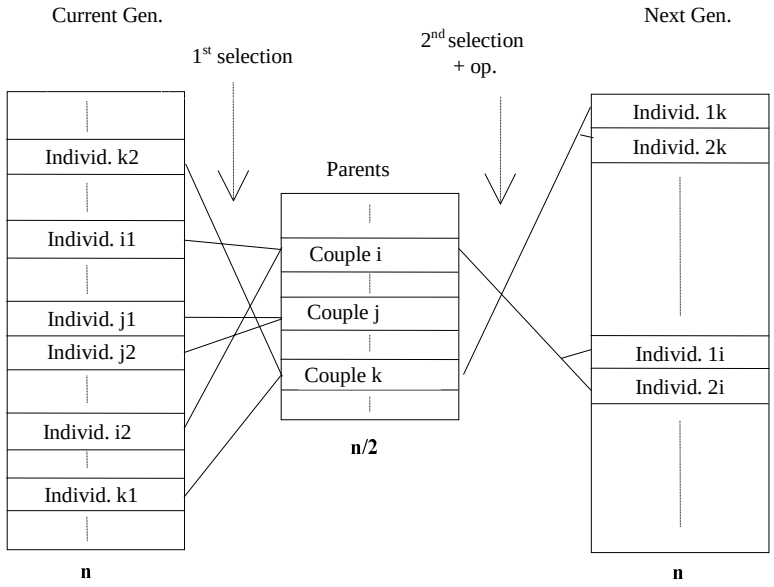
cluster →	3	1	4	5	1	6	3	9	9	7
module →	1	2	3	4	5	6	7	8	9	10

Fig. 2. Chromosome structure for the cluster allocation problem

Two genetic algorithms were contrasted, a modified simple genetic algorithm (MSGA) and a combination of multiple crossovers per couple with fitness proportional couple selection (MCPC-FPCS). The MSGA used the above indicated integer representation, uniform crossover and little-creep operators [4] for mutation.

The basic idea of FPCS [7] is to create, from the current population of individuals, an intermediate population of couples which subsequently undergoes selection for mating under MCPC those pairs showing higher fitness (see Fig. 3). To assign the fitness to the couple a criterion based on the parents fitness average ($FPCS_{AVG}$) was chosen. The method can be sketched as follows:

- A number of individuals are initially selected by proportional selection to build the intermediate population of parents.
- A couple fitness value, computed in accordance to the couple fitness criterion, is assigned to each mating pair.
- Couples are selected for repro-duction by proportional selection (according to couple fitness). The process of producing multiple offspring (MCPC) is controlled, for each mating pair, in order not to exceed the population size.



1st Selection: proportional selection to the individual fitness.
2nd selection + op: proportional selection to the couple fitness plus classic genetic operators.
n: population size.

Fig. 3**.** Couple Selection

The following relevant performance variables were chosen:

Best: It is the objective value of the best found individual.

Ebest = (Abs(*opt_val* - best value)/*opt_val*)100
It is the percentile error of the best found individual when compared with the known, or estimated, optimum value *opt_val*. It gives us a measure of how far are we from that *opt_val*.

Epop = (Abs(*opt_val*- pop mean fitness)/*opt_val*)100
It is the percentile error of the population mean fitness when compared with *opt_val*. It tell us how far the mean fitness is from that *opt_val*.

Gbest: Identifies the generation where the best value (retained by elitism) was found.

Time: It is the running time in seconds of the algorithm implemented on a Pentium PC (MMX of 233 MHz).

A particular benchmark case with known optimum was chosen for experimentation. Different setting of parameters, such as crossover and mutation probabilities, population size and maximum number of generations, were used.

Tables 1 and 2 show the 10 best results under each method. Those in boldface correspond to the optimum solution.

P.Cross	P.Mut.	#Pob	#Gen	Ebest	Epop	Gbest	Best
0.650000	0.001000	200	50	0.000006	0.032412	21	**4.037463**
0.650000	0.001000	200	100	0.000006	0.000006	21	**4.037463**
0.650000	0.050000	200	100	0.000006	26.899259	65	**4.037463**
0.500000	0.050000	200	100	0.053974	16.329121	68	4.039642
0.500000	0.050000	200	50	0.237420	19.403321	49	4.047049
0.500000	0.050000	100	100	0.263320	17.944509	64	4.048094
0.650000	0.050000	50	50	0.422890	7.381497	47	4.054537
0.650000	0.050000	50	100	0.422890	7.851154	47	4.054537
0.500000	0.001000	200	50	0.473060	0.545403	28	4.056563
0.500000	0.001000	200	100	0.473060	0.473060	28	4.056563

Table 1. Results of the best ten runs for MSGA

P.Cross	P.Mut.	#Pob	#Gen	Ebest	Epop	Gbest	Best
0.500000	0.050000	200	50	0.000006	19.659855	39	**4.037463**
0.500000	0.050000	200	100	0.000006	16.261753	39	**4.037463**
0.650000	0.050000	200	100	0.476376	27.080477	90	4.056696
0.650000	0.001000	200	50	0.487356	0.627650	29	4.057140
0.650000	0.001000	200	100	0.487356	0.832470	29	4.057140
0.650000	0.050000	200	50	0.915364	26.391131	24	4.074420
0.500000	0.001000	200	50	1.032137	1.032137	37	4.079135
0.500000	0.001000	200	100	1.032137	1.102010	37	4.079135
0.650000	0.050000	50	100	3.160282	11.271904	87	4.165058
0.500000	0.050000	100	100	3.466643	15.800654	97	4.177427

Table 2. Results of the best ten runs for MCPC-FPCS

Time, was recorded in 2 seconds and 4 seconds for 50 and 100 generations respectively.

The following observations can be done:

- Both approaches find the optimum value for some setting of parameters: different crossover and mutation probabilities but equal population size of 200. This fact indicates that a greater population size improves the search process.

- MSGA finds the optimum in earlier generations.

- MCPC-FPCS finds near optimal solutions in earlier generations.

- Conventional probabilities of crossover and mutation (0.65 and 0.001) favours the MSGA approach, while lesser values for Pc and higher values for Pm (0.5 and 0.05) favours MCPC-FPCS.

- The error of the best individual (Ebest) ranges from 0.0% to 0.5% for MSGA and from 0.0% to 3.4% for MCPC-FPCS.

- The error of an average individual in the population (Epop) ranges from 0.0% to 26.6% for MSGA and from 0.6% to 27.1% for MCPC-FPCS.

Summarizing, MSGA seems to behave better than MCPC-FPCS for the selected suite of parameters. But under any approach, it is worth remarking that since GAs work on a population of feasible solutions, instead of providing a single optimal or near optimal solution a subset of optimal or near optimal solutions is always available after reaching the final generation. This peculiar characteristic yields to fault tolerance in the model by providing a significant subset of near-optimal solutions as alternates to be used in case the real world constraints prevent using the first selected alternate (eg. lack of availability of some cluster to be allocated). Alternate strategies can be ordered according their goodness [6].

4. A PROPOSED IMPLEMENTATION

The implementation would require a monitoring process to determine a Mean Parallel Program Profile (MPPP) and a GA process to determine a set of near-optimal cluster allocation strategies. The output obtained from these processes, includes part of a data base accessible to the system and residing in the Cluster Supervisor Node of each cluster. As we said above, some information about program profile is assumed to be available in data structures within the CSN of each cluster. To obtain a MPPP, the program and modules behaviours are monitored, relevant information gathered and associated data structures (e.g., M, CB, MODB, RWDB, IMC) initialized or updated.

As any cluster in the network could become an initiation cluster (c_{init}) or an execution cluster ($exec_i$) each CSN maintains a data base to provide information about locations of data files and program files. In addition, CSNs belonging to those clusters holding at least a program file in any of its nodes, maintains the corresponding MPPPs and the output from the initial GA run. Besides, each CSN updates a description of the availability state of remaining clusters.

After written and compiled, parallel programs will be submitted to a set of initial runs to create the corresponding MPPP. Depending on user requirements, programs could be occasionally or frequently modified. In the former case only occasional updates will be necessary to maintain a reliable MPPP, while in the latter case updates would be as frequent as new program versions are created. In either case after obtaining a new MPPP a genetic algorithm is run to acquire a new subset of near-optimal cluster allocation strategies. But for how long do initial strategies remain valid?

The GA run provides its output based on the current set of available clusters. The current state of each cluster (available or not available) is ready for checking within each CSN. Concerning to cluster

[6] In our model we say that solution *i* shows higher goodness than solution *j* iff $TITC_i < TITC_j$.

availability, internets show variable behaviour. Consequently, after each request of a parallel program execution, CSN in c_{prog} follows a simple procedure to provide a feasible near-optimal allocation strategy:

I. It checks, in an orderly manner, each previously determined alternate strategy with the current available clusters set. (e.g., it verifies if for any of the near optimal C_D^* vectors all its components belong to the current set C_A of available clusters).

II. If any of these allocation alternates could be satisfied within the current system state, then process halts as soon as the first match is found and that strategy is selected. Otherwise, if no alternate satisfies (e.g., because of clusters unavailability) a new GA run is necessary to establish the current valid subset of near-optimal cluster allocation strategies.

In those internets where clusters availability remains almost unchanged it will rarely be necessary to run a GA process while satisfying a request of a parallel program execution and the system performance will be near optimal. If due to changes in cluster availability, a GA process would be necessary then some extra seconds are required to establish a subset of near-optimal strategies but the system keeps trying to make efficient use of communication resources.

Once parallel tasks are allocated to near-optimal clusters a selected load balancing strategy can be used to settle the corresponding executing node within each cluster (Eager et al. [5], Jacqmont et al [9], Wang et al. [13]). This also depends on how CPU availability is defined in the system. If an idle workstation approach is used then selection of executing node will not require load balancing [10]. In case of searching for an underloaded workstation one of the simplest approaches is deciding to choose that workstation with maximum ratio of attested processing power to the number of running processes.

5. CONCLUSIONS

Evolutionary computation has recently been recognized as a research field. Applications are possible in diverse areas. A broad but not exclusive categories involve scheduling, packing, routing, on line and off line control and design. Computer systems, including networks, are promising areas of applications for evolutionary computation.

The present paper introduced a model to near-optimally allocate available clusters of an internet, for execution of program parallel components. Two different evolutionary approaches, MSGA and MCPC-FPCS, were contrasted and despite their differences in implementation and results both, definitely provide not a single optimal solution but a set of timely optimal or near optimal solutions to the first subproblem for parallel task allocation.

The architecture of the proposed implementation contemplates the dynamics of the system behaviour by incorporating procedures within the Cluster Supervisor Node to check the feasibility of the current set of near-optimal strategies and by contributing to fault tolerance through multiple alternates.

Also, the flexibility of the genetic algorithm approach allows by changing parameters, such as population size and number of generations achieved, and genetic operators to tune the goodness of alternate strategies according to the accuracy and speed on response demanded by the problem. Future work is related to the incorporation of different multiple recombination approaches and incest prevention to escape from premature convergence.

6. ACKNOWLEDGEMENTS

We acknowledge the cooperation 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. REFERENCES

[1] Akoka, J. – *Designs of Optimal Distributed Database Systems*- Proceedings of the 1st Int. Symposium on Distributed Data Bases, Paris, France, pp 229-245, 1980.

[2] Borghoff U. M. - *Design of Optimal Distributed File Systems: A Framework for Research* - ACM Press, Operating Systems review, Vol 26, No 4, October 1992.

[3] Chen P. P. S., Akoka J., - *Optimal Design of Distributed Information Systems - IEEE Transactions on Computers c- 18:10, pp 885-889. 1969.*

[4] Davis L. - *Adapting Operators Probabilities in Genetic Algorithms* - Proceeding of the Third International Conference on Genetic Algorithms- pag. 61-69, 1989.

[5] Eager D., Lazowska E., Zahorjan J. - *Adaptive Load Sharing in Homogeneous Distributed Systems* - IEEE, Transactions on Software Engineering SE-12, 662, 675, 1986.

[6] Esquivel S., Leguizamón G., Gallard R., - *A Quasi-Optimal Cluster Allocation Strategy for Parallel Execution in Distributed Systems Using Genetic Algorithms*, ACM Press, Operating Systems Review, USA, Vol. 29, Nr. 2, pp 82-96, April 1995.

[7] Esquivel S., Leiva A., Gallard R.: *Couple Fitness Based Selection with Multiple Crossover per Couple in Genetic Algorithms*. Proceedings of the International Symposium on Engineering of Intelligent Systems (EIS´98), Vol. 1, pp 235-241, La Laguna, Tenerife, Spain, February 1998.

[8] Hač A., Johnson T. J. - *Dynamic Load Balancing through Process and Read-Site Placement in a Distributed System* - AT&T Bell Tech. Journal, pp 72-85, 1988

[9] Jacqmont C., Milgrom E., Joosen W., Berbers Y. - *Unix and Load Balancing: a Survey* -Proc. Europ. UNIX System User Group Conf. Spring '89, Brussels, Belgium, April 1989, Butingford Herts, UK:EUUG, pp 1-15.

[10] Kleinrock L., Korfhage W. - *Collecting Unusing Processing Capacity: An Analysis of Transit Distributed Systems*.- Proc. of XIth ACM Simposium on Operating Systems principles, 1989, pp 482-489.

[11] Morgan H. L., Levin K. D. – *Optimal Program and Data Location in Computer Networks* – Communications of the ACM, 20:5, pp 315-321, 1977.

[12] Wah B. W., - *File Placement in Distributed Computer Systems*- IEEE Computer 17:1, pp 23-33, Jan. 1984.

[13] Wang Y., Morris R., - *Load Sharing in Distributed Systems* - IEEE Transactions on Computers, c-34:3, pp 204-217,