EVOLUTIONARY ALGORITHMS TO FACE COMPUTER SYSTEMS MANAGEMENT PROBLEMS

GALLARD R.H., ESQUIVEL S. C.

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: { rgallard esquivel}@unsl.edu.ar
Phone: + 54 652 20823
Fax : +54 652 30224

Abstract

Evolutionary Algorithms (EAs) became a powerful tool for environments where an optimization is needed or where an adaptive behaviour of a system is necessary.

Computer Systems have been evolving rapidly during the last five decades. Since the 70's, due to improvements in communication and computer hardware, an evolution of computer systems based on networked workstations has been taking place. This advance triggered present distributed systems. At each stage of this evolution, resource management was designer's main concern. Decisions on how to allocate a limited set of expensive resources and how to schedule arriving tasks with diverse requirements were based on various heuristics, attempting optimization of, frequently conflicting objectives, from the user and system perspectives.

This paper discusses feasible areas where EAs can be efficiently applied to solve resource management problems in Computer systems, show two feasible implementations and introduces last improvements to enhance EAs performance.

Keywords: Evolutionary algorithms, resource management, computer systems, task allocation.

[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

Computer Systems have been evolving rapidly during the last five decades. Between the 50's and 60's the emphasis was put on centralized computer systems with increasing power. Number crunching was the main goal for early scientific computing systems. Also, massive data processing was perceived as a major task. Consequently, the technological improvements aimed at the increase of secondary storage facilities. A development on operating systems resulted in a change: from serial processing to batch systems and time sharing. Also a shift from monoprogramming to multiprogramming and later from monoprocessing to multiprocessing were the main changes observed at that time.

Since the 70's, due to improvements (cost-wise and technological) in communication and computer hardware, an evolution of computer systems based on networked workstations has been taking place. This advance triggered present distributed systems. All these changes were possible due to (1) improved hardware technology, from vacuum tubes and relays to VLSI, and (2) software development, from absolute coding to programming environments and methodologies. At each stage of this evolution, resource management was designer's main concern. Decisions on how to allocate a limited set of expensive resources and how to schedule arriving tasks with diverse requirements were based on various heuristics, attempting optimization of, frequently conflicting objectives, from user and system perspectives.

The inclusion of intelligence into computer systems was a long overdue project. When the results of AI research became widely applicable, Blair et al. [1] and Nicol et al. [10] proposed a holistic approach for operating systems. Their approach was based on insertion of an expert system into the operating systems kernel to provide some degree of automatic decision making. These papers encouraged us to experiment with a new approach to intelligent resource management in computer systems: evolutionary algorithms, used as intelligent tools, instead of an expert or knowledge based system.

Under this proposal the following questions arise:

- Are EAs applicable, as intelligent tools, to problems in Computer System Resource Management?
- Can they compete with conventional approaches?
- Which are the most suitable problem sets?
- Dynamic or static problems?

As a consequence of these open questions the work deepened on the application field by analysing the ability of new proposed enhanced variants of evolutionary algorithms to solve a selected set of computer systems problems. The following sections discuss these aspects.

2. FEASIBLE APPLICATION OF EVOLUTIONARY ALGORITHMS TO RESOURCE MANAGEMENT IN COMPUTER SYSTEMS

As current trends in computer systems are oriented to multicomputer systems, our focus aims at these environments. Since resource management in computer systems is a wide area for research, it is first necessary to detect those areas where the application of evolutionary algorithms could be feasible by providing a timely solution to a problem. We caracterized the suitable problem sets as belonging to two main groups as follows:

a) Those that being static have a priori information on main system characteristics. Here EAs provide a set of solutions off line. The following are some examples:
   - Finding a shortest (or minimum cost) path between two nodes in a computer network. Existing connectivity and link costs are known in advance.

- Finding a cluster allocation distribution with minimum communication cost to allocate parallel program components in a distributed environment. The number of components, how they interact, which files they access, which is the input/output data transfer volume and other features are statistically known in advance.
- Scheduling of parallel tasks in parallel supporting platforms. The associated task graph and the number of available processors are known in advance.

Even in the case of static problems when EAs are used, the dynamical behaviour of the system can be faced through the set of alternative solutions provided.

b) Those that being dynamic or not having a priori information on main system features, are related with computer system environments where the response is not subject to critical time constraints. Here EAs provide a set of solutions on the fly. As examples of this kind of problems we have:

- Dynamic scheduling of independent tasks arriving in unknown order to a parallel computer system.
- Dynamic load balancing in distributed environments.

In these cases the EA is self-adapted to the system response and a sort of predictive ability is inserted in the EA.

Consequently, various problems such as routing, cluster allocation, parallel task scheduling and load balancing, were considered as suitable areas. All the applications showed clearly the benefits of the evolutionary approach, which (when possible) was contrasted with conventional approaches. Two of these problems, load balancing and cluster allocation, are discussed in some details in the following sections.

3. ENHANCED EVOLUTIONARY ALGORITHMS FOR LOAD BALANCING

Load balancing algorithms attempt to improve systems performance through process migration. Here we present a hybrid strategy for load balancing in distributed systems, which exploits the benefits of evolutionary and predictive approaches [3]. In order to decrease the communication traffic in a local area network typically generated by load balancing schemes, we sought for a reduction in the number of requests done by an overloaded node. The predictive strategy applied to achieve this goal uses the knowledge attained by each node, through its previous experience.

A set of processors interconnected by a Local Area Network (LAN) is a classical example of a Distributed System. The model considered here consists of a set of homogeneous and independent nodes: hardware and software are similar for each node and every node has its own processing resources and information storage. In such a system, users from different sites create autonomous processes which occasionally need synchronisation to share critical resources. The workload in a node embodies a set of local and possibly external demands on all or some of its resources. This global demand varies during the execution of processes and could lead the system to an imbalanced state where some of the nodes are overloaded while others are underloaded or even idle.

Previous works [8], [3] showed how dynamic load balancing strategies based on evolutionary algorithms achieved improved results when confronted with traditional load balancing algorithms. In those cases the enhancements came as a result of decreasing the number of migration requests from an overloaded node. Towards an improvement in performance related to a more balanced use of resources, the proposed system uses, in a dynamic and automated way, an evolutionary load balancing strategy. This section shows improved results achieved by applying a predictive approach based on the knowledge gained by each node through the evolutionary process. In this strategy the number of migration requests is notably reduced and, consequently, better mean response times are obtained. The proposed strategy is:

- *Dynamic*. Decisions on processor allocation are made during execution
- *Non-preemptive*. Process migration is not allowed once the process has begun execution
- *Decentralized*. Each node has decision making capability.

As the model assumes independence of processes the goal of the strategy is to minimize the mean time of processes in the system. This performance variable will be called *mean response time.*

Dynamic load balancing strategies are prone to instability, a problem that arises when processes waste considerable extra time due to continuos process migration. In order to minimize this overhead evolutionary algorithms are used expecting an adaptive behaviour of the strategy on response to changes in the system. This way the requests are sent to those nodes more inclined to accept them. Besides the classification given above load balancing strategies can be categorized as *sender-initiated*, this means that the migration request is done by the overloaded node, or *receiver-initiated,* where idle or underloaded nodes request for processes to be processed. We have implemented a sender initiated strategy.

3.1. SYSTEM'S DESIGN

The various components implementing the load balancing strategy in each node are sketched in Fig. 1.

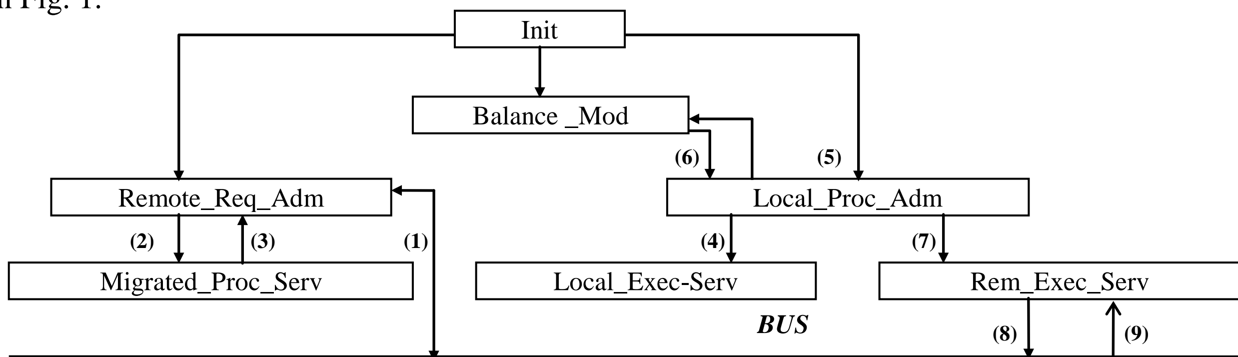

Fig. 1. Load Balancing Strategy Modules

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

BALANCE_ MOD implements the load balancing strategy. In the simulated system, besides an evolutionary algorithm, it contains diverse load balancing algorithms for performance studies.

REMOTE_REQ_ADM has two main tasks:

- (1) To reply to migration requests from other nodes giving information about the local loading state (number of waiting processes, or ready queue length). Also, when an immigrated process finishes execution in the local node, it informs about this event to the (original) sending node.
- (2) To activate 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 (*queue_length* ≤ 2).

MIGRATED_PROC_SERV executes locally an immigrated process and, on completion, signals (3) the event to ***REMOTE_REQ_ADM*** .

LOCAL_PROC_ADM is responsible, at local process creation time, for verifying the local node load balancing state by comparing the current queue length to a prefixed threshold to determine overloading. Depending on comparison results one of the following actions is undertaken:

- If *queue_length* $\leq$ *threshold* then the task is locally executed and a child process, ***LOCAL_EXEC_SERV*** is activated (4).
- Otherwise, invokes (5) ***BALANCE_MOD*,** which indicates (6) if the new process can be migrated and to which node. If a receiving node can be found then a ***REM_EXEC_SERV*** process is created (7). On the contrary, local execution is accepted and it behaves as above

LOCAL_EXEC_SERV is in charge of the local execution of a process.
REM_EXEC_SERV is responsible for migrating (8) the process (indicated by ***LOCAL_PROC_ADM***) to the receiving node (pointed by ***BALANCE_MOD***). Finally, it blocks itself waiting for a reply (9) to confirm the remote execution completion.

3.2. LOAD BALANCING STRATEGIES DESCRIPTION

Two strategies are describe now.

Evol_St_1

When a local created task needs to be migrated, the strategy proceeds following three steps:

1) The local ***TASK_DISPATCHER*** (a system node component) sends a migration request to a subset of network nodes asking about their willingness to accept a migrated task.
2) Target nodes, depending on their own load status, reply with an accepting (**x**) or rejecting (**0**) message. Where $0 < \mathbf{x} \leq 2$ indicates the *queue_length* of the replying node.
3) If more than one node sends an accepting message then the ***TASK_DISPATCHER*** chooses, for migration, the node with fewer processes in its ready queue.

The strategy is implemented following an evolutionary approach. For this purpose, each node has its own population on which the genetic operators of selection, crossover, mutation and elitism are applied [6], [7]. Each chromosome in the population is coded as a binary vector $[\mathbf{p_1,....,p_m}]$, representing the set of processors composing the network. The semantics for each possible value of $\mathbf{p_i}$, is the following:

- If $\mathbf{p_i = 1}$ then p_i ***is a possible candidate*** to send the next migration request.
- If $\mathbf{p_i = 0}$ then p_i ***is not a candidate***.

When ***LOCAL_PROC_ADM*** determines that the local node is overloaded, it issues a migration request to ***BALANCE_MOD*** and then remains ready for new requests. The module ***BALANCE_MOD*** selects an individual from its population with a probability proportional to its fitness and sends migration requests to the nodes indicated by the vector (chromosome), then it blocks itself waiting for a response. This request is sent to each candidate node. When replies arrive then: If more than one accepting reply appears, the module decides to migrate the task to the less loaded node and in case of equal *queue_length* values a random selection is done. If none of them accepts, then it asks to ***LOCAL_PROC_ADM*** for local execution. After asking for migration or local execution, the ***BALANCE_MOD*** computes the fitness for each chromosome and creates a new population from the old population. The genetic operators applied during this stage are elitism, two point crossover and uniform mutation. The learning process is performed through the fitness function which looks at the rewards given to each individual for determining their corresponding fitness values. Rewards are proportional to the number of hits when requesting migration:

Reward = # of accepts / # of requests

Evol_St_2

The predictive evolutionary strategy:

- profits from the knowledge the node has (subset of nodes prone to receive requests) but
- for the sake of efficiency, selects each time no more than 50% of the candidates.

For deciding to which nodes to send a request to, predictions were done using the technique known as Weighted Exponential Average [11]. This technique permits to predict a value on the basis of values appeared during the whole elapsed time. In our case, values represent the acceptance hit ratio

for the processor's previous requests. For an arbitrary processor, the simplest average to be used for prediction of its response is given by:

$$V_{n+1} = \frac{1}{n}\sum_{i=1}^{n} P_i \ , \qquad (1)$$

where:

V_{n+1} : predicted processor's response (accept or reject) for the next migration request.
P_i : is the effective processor response to the ith request

In equation (1) an equal weight is given to each previously observed value. In our model, due to the dynamic behaviour of the system, it is appropriate to give a larger weight to more recent history. Then using the weighted exponential average we have:

$$V_{n+1} = \alpha P_n + (1-\alpha)V_n, \ \ 0 \le \alpha \le 1 \qquad (2)$$

In equation (2), when using any constant value for α, all observed values contribute to the predicted value but those that are further in the past have lower weight. The parameter α permits to control the relative weight to be given to recent or past history. If α is equal to 0 then recent history is considered irrelevant (present conditions are transient), otherwise, if α is equal to 1 then recent history is important and past history is obsolete. In the first step of the previous section, under this new approach, ***BALANCE_MOD*** is not sending a request to each processor with its corresponding bit set in the chromosome. Predictions are done to determine the best subset of processors (50% of the candidates at most) for sending migration requests. To implement this policy for *m* processors, it was only needed to add a *m*-entry table. Each entry storing the corresponding V_n value while P_n , the sample value, was obtained from the node response.

3.3. EXPERIMENTAL TESTS, SIMULATION SOFTWARE AND RESULTS

The distributed system implementing the strategy was simulated using a computer systems modelling tool, PARASOL [9], which is oriented to modern distributed or parallel computer systems. PARASOL consists of a set of libraries written in C which, besides typical facilities found in other discrete simulation systems, offers a set of constructors to define system hardware and network topology. PARASOL runs under UNIX-like operating systems.

The distributed system was simulated with the following parameters.

- Number of nodes: 10, 16 , 20, and 25.
- Each node execute concurrent processes under a *round-robin* discipline maintaining a *ready* queue.
- Network topology: *Ethernet*.
- Process type: All of them are CPU intensive.
- Service time: fixed for all processes at 1 sec.
- Process size: fixed for all processes at 64 Kb.
- Transfer rate: 10 Mbits.
- Process arrivals follow a Poisson distribution of mean λ. In order to analyse diverse workload levels different values for λ were used.
- The simulation is completed when 10,000 processes were executed in the network nodes.

For the evolutionary algorithm the following parameter settings were used.

- Population size in each node: 10 individuals.
- Crossover probability: 0.5
- Mutation probability : 0.005

These parameter values were determined after many experimental runs for tuning the evolutionary algorithm. As mentioned earlier, ***LOCAL_PROC_ADM*** determines if an arriving task can be

locally processed based on its ready queue length. The loading state of a node was determined according to the following criteria:

- queue_length > 4 ⇒ overloaded
- 3 ≤ queue_length ≤ 4 ⇒ medium load
- queue_length ≤ 2 ⇒ underloaded.

Experiments were carried out on three different scenarios, using $1/\lambda$ as mean interarrival time with λ = 0.1, 0.2, ..., 0.9, 1. For Evol_St2 after various preliminary runs, α=0.5 and an initial predicted valueV_0=0 were used for all nodes. Both strategies are sender-initiated.

- *Scenario 1:* 40% of the nodes are receiving processes with equal arrival rate while in the remaining nodes arrivals do not occur. This scheme allows simulation of a clearly imbalanced situation.
- Scenario 2: 40% of the nodes receive processes at a low rate (λ) while the remaining 60% receives processes at a higher rate (2λ).
- *Scenario 3:* Each node *i* has its own arrival rate which is established as a function of time $\lambda_i(t)$.

Scenario 1 attempted to reflect a real situation, which frequently occurs, where the workload is not evenly distributed. Scenarios 2 and 3, are similar in the sense that arrivals occur in every node, but scenario 3 differs reflecting time depending arrival rates as often occurs in a computer network. Next we will describe simulation results under each considered scenario with different number of nodes in the system. Figures 2 and 3 and table 1, clearly show that both strategies improve the mean response time (MRT) but under *Evol_St_2* the system behaves better. A similar behaviour was observed for networks of 10, 16 and 25 nodes. For higher workload levels *Evol_St_2* reduces MRT values nearly to half of the corresponding values under *Evol_St_1*. For scenario 3, the same global results analysed in scenario 2 are obtained. In all cases, *Evol_St_2* obtains remarkable inferior values for MTR while the number of requests is reduced in an amount ranging from 20% and 30%. In the case of a network with 10 nodes it can be seen that, as expected, the number of failed migrations (FM) increases, although this does not impair performance significantly. On the contrary, corresponding MRT values resulted better than under *Evol_St_1*. It is worth noticing that, for such a small network the chromosome size is 10 and by reducing to 50% the number of nodes receiving requests a broadcast to at most 5 nodes will be done. Consequently, the number of failed migrations increases. When the number of nodes is augmented, under *Evol_St_2* the number of failed migrations decreases and behaves similar to *Evol_St1* on this respect. But concerning to the number of requests the latter showed an increment while the former exhibited a considerable reduction. A larger number of nodes helps the learning process in each node, because the chromosome size augments. This fact provides a greater genetic diversity in the population making the task easier for the evolutionary algorithm. Consequently, a chance for a better initial chromosome also improves prediction ability.

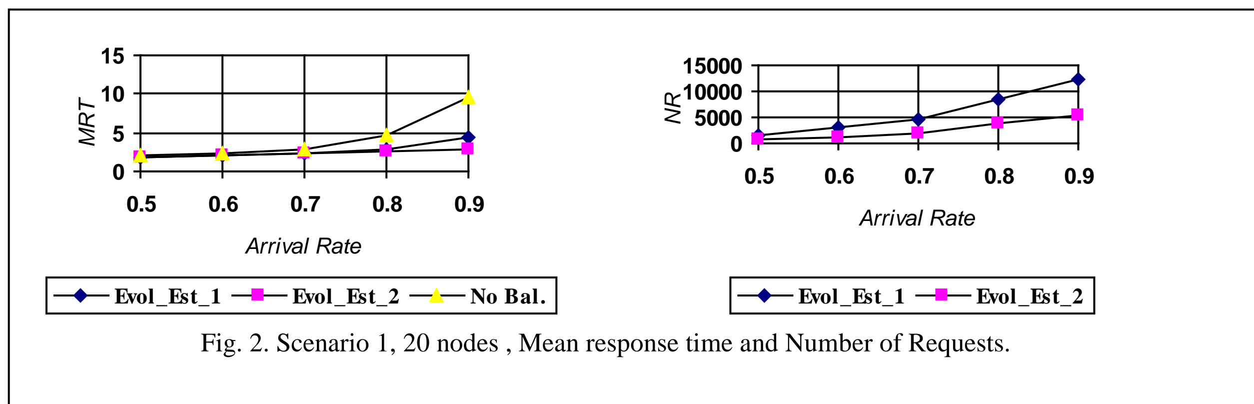

Fig. 2. Scenario 1, 20 nodes , Mean response time and Number of Requests.

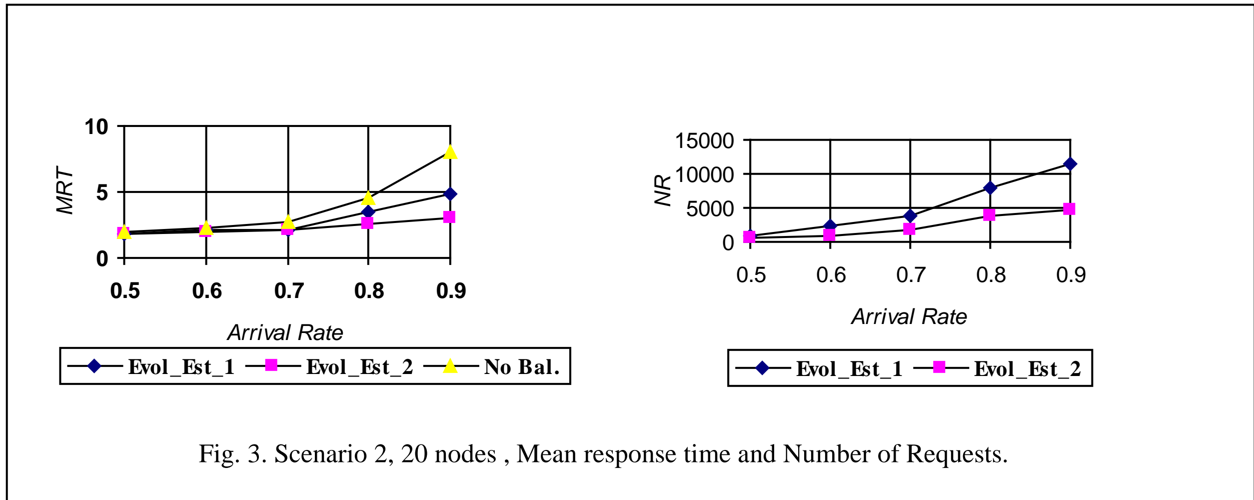

Fig. 3. Scenario 2, 20 nodes , Mean response time and Number of Requests.

Number of Nodes	*Evol_Est_1*			*Evol_Est_2*		
	MRT	NR	FM	MRT	NR	FM
10	11.49	21877	10	4.13	9117	745
16	18.95	33847	2	5.40	13621	3
20	29.05	47071	0	9.92	20332	0
25	37.44	56029	0	13.03	24965	0

Table 1. Scenario 3 Mean response time, Number of Requests and Migrations Failed.

4. ENHANCED EVOLUTIONARY ALGORITHMS FOR THE CLUSTER ALLOCATION PROBLEM

In a distributed system, users migrate to different machines, users invoke different programs and users and programs need distinct data files to satisfy their expectations. 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.

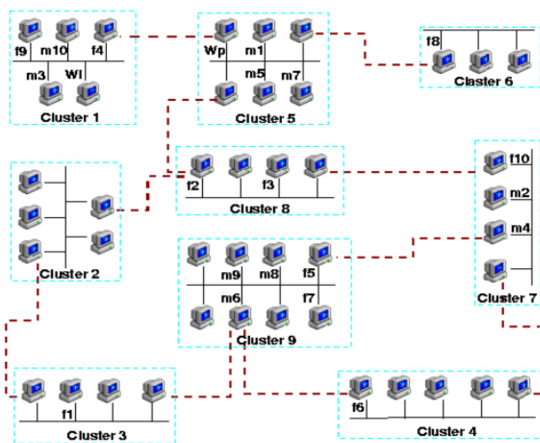

Fig. 4. A possible layout of parallel tasks within clusters.

The model

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. Figure 4 shows the final allocation of parallel tasks after the strategy was 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 allocated to nine available clusters following the minimization criteria.

It is assumed that:

- 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 with each parallel program:

- *M:* Modules. A scalar describing the number of modules comprising the program.
- *CB*: Code Blocks (including executing environment). An *M*-vector indicating the length of the 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:* Inter Module 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 any of the nodes of cluster *j*. $FD_{i,j} = 0$ indicates that file *i* is not stored in any of the nodes 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.

Our objective function is the Total Intercluster Traffic Cost (TITC). If $C_A = \{c_{A_1},..., c_{A_N}\}$ is the set of available clusters in the network, our goal is to find an execution cluster distribution

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

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's request to run the program, plus migration of parallel tasks to available clusters.
- *Intermodule Communication Cost (IMCC):* Includes the cost of transfering 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.

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

4.1. 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 problem depending on *N* and *M*. To contrast results and as a first approach we decided to experiment with three

optimization algorithms: Branch and Bound, Hillclimbing, Simulated Annealing and Evolutionary Algorithms.The algorithms were applied on six different scenarios with diverse number of modules, clusters and files each (see table 2), and ten series of twenty runs, with diverse values for the corresponding input parameters were accomplished.

Parameter	Scenario					
	1	2	3	4	5	6
# of modules	15	20	15	15	20	15
# of clusters	6	10	30	5	10	30
# of files	5	5	5	10	10	10

Table 2.Different scenarios for the cluster allocation problem

Branch and Bound could be tested only for problem sizes of at most 10^{10} in reasonable time, providing optimal solutions. For greater problem sizes the running time increased exponentially or the program run out of memory. After founding a suitable chromosome representation the MGA outperformed Hillclimbing and Simulated Annealing in quality of results and speed of convergence. Values in table 3 represent the best values found using different parameter settings and each entry in the table accommodates the solution value (TITC) and the time (t) consumed to get it, expressed in seconds (on a SUN Sparc 10). Detailed information can be seen in [2].

Scenario	Technique					
	HC		SA		GA	
	TITC	t	TITC	t	TITC	t
1	13.35	7	16.23	5	11.72	5
2	10.11	12	11.05	5	9.72	4
3	16.41	4	16.49	5	12.18	8
4	7.24	3	9.38	2	6.75	1
5	10.12	12	10.59	6	10.04	4
6	7.22	11	9.41	9	7.22	8

Table 3. TITC values and corresponding running time under each optimizing algorithm.

4.2. USING THE MULTIPLICITY FEATURE

Following new trends in evolutionary computation [4], [5], another set of experiments using multirecombinative approaches were performed:

- MCPC-FPCS. Multiple crossovers per couple with fitness proportional couple selection.
- MCMP. Multiple crossovers on multiple parents, inserting the best offspring in the new population. Parameters of this method are n_1 (number of parents) and n_2 (number of crossovers).
- MCMP_R. A variant of the method indicated above, inserting a random selected offspring in the new population.

A particular benchmark case with known optimum was chosen for experimentation. Here we discuss the case of 10 modules, 9 clusters and 10 files. Ten series of twenty runs, with diverse values for the corresponding input parameters were accomplished. Different setting of parameters, such as crossover and mutation probabilities, population size and maximum number of generations, were used. The following relevant performance variables were chosen:

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.

For this set of experiments, mean values of the performance variables *Ebest*, *Epop*, *Gbest* (as above defined), and absolute error values ε were obtained.

Method	n_1	n_2	ε					*Mean Ebest*	*Mean Epop*	*Mean Gbest*
			zero	**< 0.01**	**< 0.1**	**< 0.2**	**≥ 0.2**			
MGA	-	-	5.8	25.8	75.8	90.6	9.4	1.908219	9.969379	44.61904
MCMP-B	3	3	**33.3**	**86.7**	**100.0**	**100.0**	**0.0**	**0.151398**	**0.151398**	**6.666667**
MCMP-R	3	3	11.1	32.2	84.4	97.8	2.2	1.168116	14.90672	45.65000
MCMP-B	6	3	**31.1**	**93.3**	**100.0**	**100.0**	**0.0**	**0.082201**	**0.084306**	**5.857143**
MCMP-R	6	3	8.9	36.1	94.4	96.7	3.3	0.949496	14.76170	41.75000
MCMP-B	8	3	**30.0**	**93.3**	**100.0**	**100.0**	**0.0**	**0.080126**	**0.081913**	**5.222222**
MCMP-R	8	3	8.9	33.3	89.4	95.0	5.0	1.092489	14.63671	41.25000
MCMP-B	**4**	**6**	**51.1**	**94.4**	**100.0**	**100.0**	**0.0**	**0.063877**	**0.063877**	**4.804348**
MCMP-R	4	6	10.0	32.2	88.9	96.7	3.3	1.060883	15.01473	37.94444

Table 3. Absolute error and mean values of the performance variables under each evolutionary approach

These results indicate that:

- Evolutionary algorithms always ensure finding near-optimal solutions and in varying percentage of cases they find optimal solutions.
- Worst results correspond to MGA, with 5.8% of optimal solutions and 90.6% of near-optimal solutions with an absolute error level of less than 0.2 .
- Best results are found under MCMP-B (for $n_1 = 4$, $n_2 = 6$) with 51.1% of optimal solutions, 94.4% of near-optimal solutions, with an absolute error level of less than 0.01. More detailed analysis revealed that all solutions have an absolute error level of less than 0.03.
- Both MCMPs variants outperform MGA.
- Mean values of *Ebest* and *Epop* show again the effect of clustering the population around the optimum under MCMP-B for any (n_1, n_2) association.
- When observing mean *Gbest* values, it is clear that the optimum is found in earlier generations when MCMP-B is applied.

5. Conclusions

In this proposal we have demonstrated that evolutionary algorithms are efficient applicable tools for management of resources in computer systems, and have provided experimental evidence of this claim through diverse applications in the field, two of them are described here. For the cluster allocation problem, evolutionary algorithms were used to provide also, not a single optimal solution but a set of timely near optimal solutions. For the load balancing problem, mean response time improved when a good load balancing strategy was applied. And it was shown that when an evolutionary approach is used, results better than those obtained with traditional algorithms can be expected. Searching for a traffic reduction in the system the evolutionary strategy was enriched by incorporating a prediction function to the load balancing module. Hence, by using the knowledge gathered by each node, the number of nodes to be consulted when overload occurred was drastically reduced. The experimental results obtained through simulation gave a clear indication about the efficiency of the proposed hybrid strategy. Finally, it was shown that when appropriately designed they outperform conventional approximative approaches in those problems where strict time constraints are not the main issue. As they are population-based algorithms, when optimal solutions are sought a set of near-optimal solutions in the final population provide fault tolerance to possible changes in system configuration as in the cluster allocation problem. As they are adaptive, they can

give appropriate response to changes in system environment and improved through hybridization as in the load balancing problem.

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. BIBLIOGRAPHY

[1]. Blair G. S., Mariani J. A., Nicol J. R. and Shepherd D. - *A Knowledge-based Operating System* - The Computer Journal, Vol 30, No 3, 1987.

[2]. 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.

[3]. Esquivel S., Leguizamón G., Gallard R., - *A Hybrid Strategy for Load Balancing in Distributed Systems Environments*- Presentado y publicado en Proceedings of the Fourth IEEE International Conference on Evolutionary Computation (ICEC'97), pp. 127-132, ISBN 0-7803-3949-5, Indianapolis, USA, April 1997.

[4]. Esquivel S., Leiva A., Gallard R., - *Multiple crossover per couple in genetic algorithms*. Proc. of the 4th IEEE International Conf. on Evolutionary Computation (ICEC'97), pp 103-106, Indianapolis, USA, April 1997.

[5]. Esquivel S., Leiva H.,.Gallard R., *Multiple Crossovers Between Multiple Parents To Improve Search In Evolutionary Algorithms.* Proceedings of the 1999 Congress on Evolutionary Computation (IEEE). Washington DC, pp 1589-1594.

[6]. Goldberg, D.E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA, 1989.

[7]. Michalewicz Z. - *Genetic Algorithms + Data Structures = Evolutions Programs* -Springer-Verlag, third, revised edition, 1996.

[8]. Munetono M., Takai Y. y Sato Y. - *A Gemetic Approach to Dynamic Load Balancing in a Distributed Computer System*- , 1st IEEE CEC, Junio 1994, Vol. I, pags. 419 - 421.

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

[10]. Nicol J. R., Blair G. S. and Walpole J. - *Operating Systems Design: Towards a Holistic Approach?* - ACM Press, Operating System Review, Vol. 21, No 1, Jan. 1987.

[11]. Stallings William - *Operating Systems*, MacMillan Publishing Company, New York, 1992.