Improving the Performance of a Dynamic Load Balancer Using a Classifier System

Jan M. Correa, Alba C. Melo

Department of Computer Science, University of Brasilia, Brazil
Campus Universitário – Asa Norte, Cx Postal 4466, CEP 70910-900 Brasília DF
{jan, albamm}@cic.unb.br

Abstract—

Processor scheduling in its general formulation is a NP-Complete problem. In the Dynamic Load Balancing problem the scheduler has to redistribute processes during their running lifetime trying to improve the performance according some optimization criterion. To tackle such a difficult problem is worth use heuristics to seek for better results. Among various heuristics, genetic algorithms are often used to handle problems with high complexity. In this paper we try to optimize the decisions taken by a dynamic load balancer with preemptive migration in a distributed environment using a Classifier System (CS). CS is an adaptive program that evolves decision rules applying genetic algorithms over a population of rules and selecting the best of them. CS has the ability of adapt to environment changes. The rules are rewarded or punished depending on their performance. This performance-driven behavior allows them to perform well even with few information. The results have been impressive and the classifier system was able to surpass, without previous knowledge of the properties of workload, the performance of a well designed analytic criterion.

Keywords— **Load Balancing, Processor Scheduling, Genetic Algorithms, Classifier Systems**

I. Introduction

There are several ways to improve the performance of a cluster of computers when doing Distributed Processing. A successful way is to perform a better process scheduling. Process scheduling is a NP-Complete problem [PAP98] where a scheduling algorithm has to decide which processes must execute on which processors. This decision is taken observing a particular optimization criterion. Load balancing is one of the most widely used optimization criteria since it is assumed that, if the total load of the system is divided more evenly between its processors, the performance of the whole system can be improved [CAS88].There are, basically, two classes of scheduling algorithms to perform load balancing : static and dynamic schedulers. Static load balancing works generally on problems that have a predictable structure and balances the work before execution time [WU97]. On the other hand, dynamic load balancers perform scheduling activities at execution time, using runtime system load information to adjust load distribution. Dynamic load balancing is suitable for a wide set of applications and requires process migration [WU97].

As we saw in [COR00], several attempts were made to apply the power of GAs to the static processor scheduling problem [DUS98], [GRA99], [SUN97], [COR99] and [SAN97] In this kind of problem, the scheduling algorithm generally has all information and time enough to do all computations needed. Despite the good results achieved in this field, very few attempts were made to apply GAs also to the dynamic scheduling problem. One reason for this is that it is difficult to evaluate the individuals of GA in a dynamic changing environment of the dynamic load balancing. It happens because the dynamic changing environment is very sensitive to the actions performed on it. Thus, it is nearly impossible to test the decisions about load balancing advocated for all individual of the population without disrupt the environment[COR01a].

A possible solution to avoid this problem and apply the power of GAs to perform dynamic load balancing could be to use a Classifier System (CS). A CS can evolve decision rules concerning load balancing with genetic algorithms in order to achieve better decision rules [BAU95] . A classifier system is composed by rules and relationships among these rules, enforced by a reward system. To find better relationships among rules, genetic operations of mutation and crossing-over can be applied [GOL89]. One type of classifier system is XCS (Special Classifier System) [WIL95] that was recently developed to generate more accurate and general decision rules than a ordinary classifier system.

In this paper, we extend the work presented in [COR01b]. We show that our modified XCS can achieve better results than other methods when achieving dynamic load balancing with process preemption on a cluster of computers. We investigated what is the impact of the network bandwidth over the methods and explained how our modified XCS take some scheduling decisions. We also show that the standard deviation of the memory size distribution can interfere with the system performance, subject that were not taken in account by the Methusela simulator developers [HAR96].

Our approach uses a function that is dynamically modified in order to adapt to the changing characteristics of the distributed environment. We claim that such a property is fundamental to achieve dynamic load balancing. Although XCS has been applied to some computational intensive problems such as Data Mining [SAX99] and Robotics [TOM99], as far as we know, this is the first time it is adapted to solve the dynamic load balancing problem.

Our results on the Methusela simulator [HAR96] show that our modified XCS based on Barry's [BAR00] implementation can reduce the average slowdown of a collection of 500 processes in a wide range of conditions, when compared with the native Methusela load balancing algorithm. This reduction in the slowdown represents more accurate migration decisions. The remainder of this paper is organized as follows. Section II presents the genetic algorithms and classifier systems. The modifications made to XCS are described in section III. Experiments and results are provided in section IV. Finally, section V concludes the paper.

II. Genetic Algorithms and Classifier Systems

Genetic Algorithm (GA) is an heuristic search method that is based upon the principles of evolution and natural genetics [GOL89]. In a basic GA, a set of possible solutions is coded as chromosomes or individuals. Genetic operations such as mutation and crossing-over are applied over the population of chromosomes thus generating new chromosomes. An evaluation function is used to characterize a chromosome's performance to the problem. Such a function is applied to all existing chromosomes and generally only the fittest ones survive to compose a new population of chromosomes in which genetic operations will be applied again. This process continues until a certain number of generations or a reasonable level of fitness is reached.

Some properties distinguishes Genetic algorithms from the other probabilistic heuristic search methods [MIC96]. First, search is made from a set of solutions rather than from a single one. This leads to a more efficient search since many solutions are evaluated simultaneously. Moreover, the probability of attaining a local minimum (or maximum) is reduced. Second, the new solutions are created using information from the previous ones. In this way, GAs exploit historical information to find new search points that would probably lead to improvement.

Classifier Systems (CS) were firstly proposed by [HOL78]. They are a class of adaptive systems, i. e., systems that are able to modify themselves to reach a particular goal. A classifier is basically a decision rule that is composed by a condition and an action. The action is only executed if the condition is true. A classifier system is made of several classifiers that interact in order to associate the environment conditions with the most suitable actions. More than that, a classifier system is able to generate new rules from the existing ones and also make rules more generic or more specific [GOL89].

XCS (Special Classifier System) is a classifier system proposed by [WIL95] where rules are seen as individuals of a population. The goal of XCS is to generate more generic and efficient rules. To achieve this goal, XCS is composed by a population of very simple classifiers made of one condition and one action. Conditions and actions are received from and executed over the environment, respectively. The XCS condition is a ternary string [0,1,#(don't care)] and the action is a binary string [0,1]. Each classifier has an associated value that represents its relative strength on the population.

The execution of XCS is divided in trials. Each trial is either for exploration or exploitation. The exploration trial basically aims to refine an existing rule [BAR00]. Genetic algorithms are used at the exploitation trial and they search new and more efficient rules from the existing ones. As GAs are computationally intensive, they are not executed in every exploitation trial [BAR00].

III. Design of a Classifier System for Dynamic Load Balancing Schedulers

Our proposal consists in modifying an existing dynamic load balancer by adding a classifier system to it, in order to improve load balancing. We decided to use a processor scheduler simulator for distributed systems called Methusela. Methusela was developed by [HAR96] at the University of California at Berkeley. It simulates a network of interconnected computers where each computer receives processes to execute. [HAR96] first used this simulator to show the advantages of preemptive process migration on a distributed environment. In order to decide which process must migrate, Methusela uses information about process age distribution [HAR96].

A. Overview of the Methusela Simulator

Methusela does preemptive dynamic load balancing this way:

Periodically, the system load distribution is examined. In the case of load unbalancing, Methusela decides which processors are overloaded and underloaded by counting processes on their CPU queue. Having a migration source and a migration destination, Methusela chooses, among the migration source's processes, which ones would more take benefit from migration.

Dynamic load balancing is implemented in Methusela by the algorithm presented in figure 1.

```
    Methusela_load_balancing()
Begin
 source_processor=  most overloaded processor();
 If (source_processor is sufficiently_overloaded)
  Begin
   dest_processor = processor_with_lowest_load();
   If (dest_processor is underloaded)
    Begin
     old_processes = Select old enough processes
     in (source_processor);
     source_processes = Select processes by age
     and migration cost (old_processes);
     Migrate(source_processes,source_processor,
     dest_processor);
   End
 End
End
```

Figure1. Dynamic Load Balancing Algorithm used in Methusela

The criterion used to choose processes for migration is process age and the older ones are chosen. To decide if a process is old enough, Methusela uses the following formula:

(1) process_age >= α* (migration_cost)

where α is a constant that specifies how much the process age must be greater than the migration cost. The migration cost is equal to

migration_cost = f + (m b)

where f is the fixed cost associated with the migration,
m is the total memory (in MB) that must be transferred and
b is the memory transfer rate (in seconds per MB)

Methusela provides also the so-called best analytical criterion. In this case, the formula (2) is used.

(2) process_age >= migration_cost / (n – m)

where n is the number of processes at the source node and
m is the number of processes at the destination node (including the potential migrant)

Processes that are considered old enough by formula (1) or (2) are placed in a list ordered by process_age / migration_cost and processes that have the highest value are chosen.

B. Integration between Methusela and XCS

One of the main problems in using Methusela is to determine the value of the α parameter [HAR96]. This is a very important parameter since it is used to decide if a process is old enough to be migrated. In order to determine the appropriate value of α, extensive experimentation must be made. Moreover, there is no guarantee that a value that is appropriate for a particular load distribution would be adequate for another one. In fact, [HAR96] suggests that each load distribution must have a different value for α.

Based on these considerations, we propose to use a classifier system to adapt the value of α at execution time to the current load balancing distribution. In order to do this, we integrated the XCS classifier system to the Methusela simulator. In this integration, the Methusela simulator has kept its independence and autonomy since all interaction between Methusela and XCS is made through a communication layer. The interaction between the systems is illustrated in figure 2.

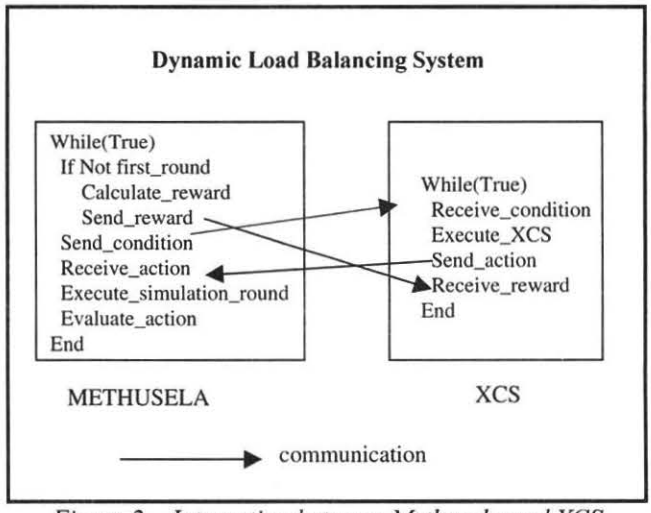

Figure 2 – Interaction between Methusela and XCS

To model the dynamic load balancing problem in a way that can be understood by a classifier system, we must answer many questions. The first one is which parameter must be modeled as a condition to XCS. We decided to use the process age as a XCS condition since it is one of the most important parameters used to decide if a process can migrate or not. The process age in fraction of second was mapped to several intervals. A function $f(x)= a * (x^b)$ where the current values are a=2 and b=1.1 The ceiling of this function is $2^{(\text{condition size})}$. The values are chosen empirically.

As can be seen, processes with small execution times were grouped together in few intervals and long processes were disposed in more intervals, since long processes have a greater probability to be migrated and is worth to allow more binary values for them so XCS can make a more accurate decision.

Based on this condition, XCS chooses which action must be executed on the environment. As the objective of our modified XCS is to find a good value for the α parameter, all the actions modify the value of α. The action is a 5-bit binary string where the first bit indicates if the value of α must be increased (1) or decreased (0). For instance, the action 00100 corresponds to $\alpha = \alpha * C2 * 4$ where $C2 = 1/ 2^{\text{action size}}$. This make the XCS to decrease α

from it's initial value to zero if needed. By a similar mechanism XCS can increase α up to it's double what is enough to prevent unnecessary migrations. After taking this action, Methusela evaluates the migration decisions taken due to the new value of α and generates rewards considering the remaining execution time. The XCS actions receive a reward in two cases. First, if the system decides to migrate a process and its execution time at the destination node was long enough to compensate the migration cost, the action is rewarded since migration was a good decision. Second, if Methusela decides not to migrate the process and the remaining execution time of the process was not sufficient to compensate the migration cost, the action is also rewarded.

In our approach, a genetic algorithm is executed every 25 trials and acts only on the rules activated by the Methusela condition. Our GA executes crossing-over and mutations on the conditions but only mutations on the actions. This choice was made because the genetic algorithm is not searching for better actions but for better mappings between conditions and actions. By applying crossing-over on the conditions, the GA is able to generate more general or more specific rules. Crossing-over is done by the roulette-wheel method [MIC96]. Mutating conditions is necessary to refine the classifier activation by testing different Methusela conditions over it. The crossing-over and mutation rates used in our GA are 80% and 2%, respectively. The values are chosen empirically.

After applying the genetic operations, new individuals are generated. New and old individuals are evaluated according to the accuracy of their actions. The most accurate classifiers survive. This is a important issue because we want classifiers to achieve the results they claim.

IV. Experiments and Results

This section evaluates the gains obtained by our Methusela + XCS approach to the dynamic load balancing problem. The metric used was the slowdown which describes how much migration and process scheduling contributed to augment the process' total execution time. Slowdown is defined to be: *(time_process_ends-time_process_starts) / CPU_time.*

The performance of the system was measured by the average slowdown, the slowdown standard deviation and the number of severely slowed processes. The average slowdown measures how much the average execution time was augmented due to migration and scheduling. The slowdown standard deviation reflects the difference between slowdowns of processes with distinct duration times. This metric can give some hints about how much processes with a big slowdown can interfere in other processes. The number of severely slowed processes measures how many processes had a slowdown equal or greater than 3. Policies that allow a great number of severely slowed processes must be avoided.

In the original Methusela [HAR96] the memory size of each process was chosen randomly, with the same random seed. This yielded the problem that every time Methusela executed, the same memory size was associated to a given process, and decisions that were taken randomly were always the same. This is fixed in this new version. Further to this we added extra parameters to generate a memory size gaussian distribution [CAR94] with a given average and standard deviation. Surprisingly, it had a great impact on the system's measures, subject that were not treated in [HAR96]. The results presented here were obtained with a Pentium II 350 MHz with 64 MB of RAM. The processing cost of each XCS trial was 4.2 milliseconds on average and the migration decisions were taken in intervals of one second. Each value obtained here is an average of 10 runs.

Methusela receives as input a tuple of parameters that will conduct the simulation. This tuple is presented below:

<file_name><migration_policy><placement_policy>
<load_threshold><n_processes><α><r_exec>
<fixed_cost> <mem_transf_cost>

Table 1 gives an explanation about each parameter.

Parameter	Value
file_name	Name of the file that contains the processes and its associated execution times
migration_policy	0 – no migration; 1 – non-preemptive migration; 2 – age-based preemptive migration only when a new process arrives at a heavily-loaded node 3 – periodical check for heavily-loaded nodes and preemptive migration
placement_policy	0 - always choose the node with the lowest load 1 – choose the node with the lowest process total age
load_threshold	0 – does not consider a threshold in the migration policy 1 - otherwise
n_processes (only used if the former parameter is set to 1)	If the node has more than number of processes in its CPU queue, it is considered overloaded
α	Constant that multiplies the migration cost. If negative, the best analytical criterion is used
r_exec	Cost of remote execution in seconds
fixed_cost	Fixed migration cost in seconds
mem_transf_cost	Memory to memory transfer cost in seconds per MB

Table 1 – List of Methusela parameters

In our experiments, we used a selection of 500 processes (file t_trace) that were collected randomly by [HAR96] on a real educational environment. Despite having various types of processes, only batch and independent processes were considered eligible for migration.

Besides Plain Methusela (PM), Best Analytic Criterion (BAC), XCS+Methusela (XCS) we established a new policy Best Result (BR). This policy uses *a priori* knowledge about the process duration to migrate the process that will execute longer to a processor where it can execute without sharing a processor. BR provides the best result that can be obtained by the Methusela algorithm. We also use the Theoretical Limit (TL) that is the same of BR with fixed migration cost equal to zero and infinite network bandwidth. Therefore, TL provides the best results for Methusela algorithm when migration is for free. Thus, TL offers the theoretical limits of the performance measures that Methusela algorithm can achieve. TL is used as the lower values of the figures and is independent of simulator parameters.

In our first experiment we will show that the process memory size standard deviation has a impact over the performance. We used migration fixed cost of 0.002 seconds as measured in a real system MOSIX for LINUX [BAR99].For instance, the input for Methusela in this first experiment was:

t_trace 3 0 1 2.0 1.0 0.1 0.002 0.1

The same input (500 files) was used for Methusela_algorithm, Best_analytical_criterion and XCS+Methusela. The network bandwidth was 80Mb/s (0.1 second for MB) and the mean of memory size is 1 MB.

The average slowdown of PB, BAC, XCS and BR are shown in figure 3. When standard deviation of processes size is zero means that all processes have the same size. A non-zero value means a bell shaped curve with a given deviation on its average. Zero and 0.5 are respectively the minimum and maximum for the gaussian random number generator.

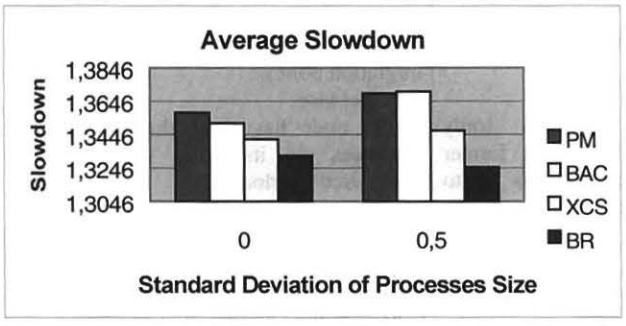

Figure 3 – Average Slowdown of the policies for the values of standard deviation of processes size

In figure 3 we can see that the average slowdown increased for almost all policies. For instance, the slowdown for BAC with 0.5 of deviation is 39.3% bigger than with deviation 0, when compared to TL. It possibly happens because as standard deviation of processes size increases, so does the complexity of the search space. It does not affect negatively the BR policy because it always does the right choice. We can also see that XCS+Methusela had better results than BAC e PM, showing less dependency on the conditions of the environment. When compared to TL, XCS showed an improvement of 19,7% over BAC for deviation zero and 36,3% over BAC for deviation 0.5.

One important factor to take in account concerning these policies of preemptive migration is the bandwidth of the network. The greater the network bandwidth is , less will be the migration cost and more processes will be eligible for migration. The choice to migrate these processes will probably improve the system performance. Thus, it is worth to investigate how the performance of the policies will vary depending on the network bandwidth.

In our trials we used fixed migration cost of 0.002 seconds, process average size of 1MB and standard deviation of 0.25, a reasonable number for a big set of processes [COU96]. In figures 4, 5 and 6 we show the results for average slowdown, standard deviation of slowdown and severely slowed processes for network bandwidth of 80, 160 and 320Mb/s.

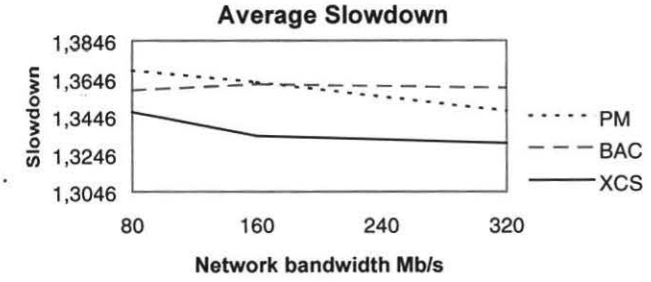

Figure 4 – Average Slowdown of the policies for the values of network bandwidth

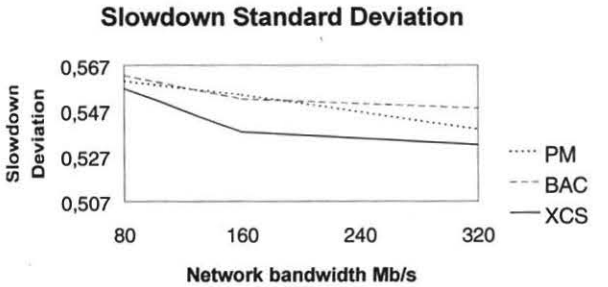

Figure 5 – Slowdown Standard deviation of the policies for the values of network bandwidth

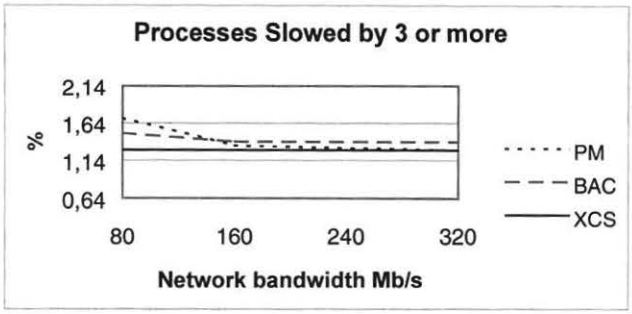

Figure 6 – Processes Slowed by 3 or more of the policies for the values of network bandwidth

In figure 4 we can see that as network bandwidth increases, the average slowdown decreases. It happens because migration gets less costly and then more processes migrate, yielding a better load balancing and therefore better results. As the network bandwidth increases, we expected that PM, BAC and XCS would present very similar results because too many processes would be considered eligible for migration, making the differences between PM, BAC and XCS to disappear. That was not the case. XCS performed better than PM and BAC for all values of network bandwidth. For 320Mb/s XCS performed 40,4% better than the others in relation to TL. BAC was designed to perform better than PM [HAR96] but it was not the case when migration is less costly as here.

In figures 5 and 6 we can see that XCS performed better than PM and BAC for these metrics too, showing that XCS is dividing the overhead of process migration more evenly between the processes.

It is difficult to forecast how long a process will last based on its actual age. Figure 7 shows the relation between the process duration and the age of the process by the time the process was evaluated.

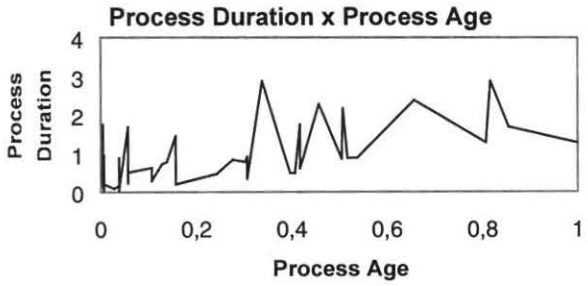

Figure 7 –Process Duration related to its age by the time of the evaluation (in seconds)

As we can see in figure 7 Process Duration seems to increase when Process Age increases, but not in a predictable fashion. One measure of the XCS action is the ratio alfa / Process Age. The alfa parameter is associated to a process evaluated for migration by XCS based on its age. Figure 8 shows how XCS varies alfa according to the process age for processes greater than 0.5 second, average process size of 1MB and parameters

t_trace 3 0 1 2.0 1.0 0.1 0.002 0.1

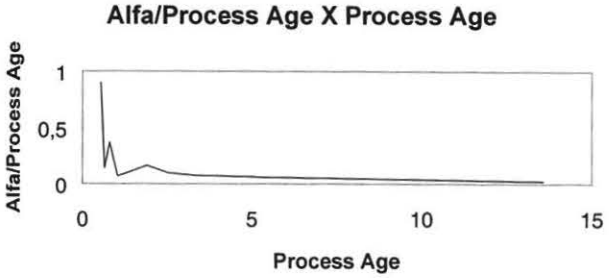

Figure 8 – Ratio Alfa / Process Age given by Methusela+XCS for processes greater than 0.5 second

As we can see, the greater is the process age the smaller is the alfa associated to it. It results that processes with greater ages have their probability of migration increased by XCS. When migration costs are very low, migration becomes very attractive. So, long processes tend to migrate continuously once they reach the migration threshold. Methusela+XCS was able to understand this situation, augmenting the α parameter to avoid early migrations

In order to see why Methusela+XCS was performing better than the analytic criterion we compared how the decision if a process is old enough for migration was related to process age. In figure 9 we compare the results of criterion of decision of Methusela+XCS and Analytic Criterion with the same parameters.

Figure 9 – How Methusela+XCS and Analytic Criterion decide if a process is old enough for migration.

Figure 9 shows that Methusela+XCS is more likely to consider processes with greater ages suitable for migration. It is important to note that at the bandwidth of 10MB/s (80Mb/s) and average process size of 1MB, an average migration would cost 0.1 seconds. Thus processes with very small age are now eligible for migration because

migration is less costly when compared to their total execution time. That is why processes with less than 1 second have great values by Old Enough Criterion in figure 9. The problem now is that such small duration processes are much more difficult to have their distribution modeled [HAR96]. So, it is more difficult to predict for how long a process will continue to execute based on its age and this is the basis of most non adaptive preemptive load balancing algorithms.

One important feature of XCS is to adapt to environment changes. Methusela+XCS is able to perform better despite having less parameters to take decisions than BAC. It happens because rules in XCS are rewarded when doing something right and punished when doing wrong. Further to this XCS has the ability of generalize rules or turn one more specific if needed. So the same decision of migrate a process of a given age may be rewarded differently depending on the behavior of the remaining processes of the system. Its allows XCS to take advantage of temporary situations.

It is important to point that XCS learned to do load balancing indirectly since it did not access the average information. Another point is that it was learned during the runtime since XCS has no 'a priori' knowledge of the workload.

One of the weaknesses of our model is that I/O is not specifically modeled. Concerning this issue, some considerations must to be made. First the traces are taken from a real academic environment and processes that are known to be interactive (such as Mail and Emacs) are not considered eligible for migration by Methusela. Further to this, the decisions of migration were taken in intervals of 1 second which means at most one migration per second. Thus, with a process average size of 1MB and network bandwidth of 10MB/s as used here, an average migration would take 0.1 second. It means that the network use of preemptive migration is only 10% on average and can be much less in faster networks.

V. Conclusions and Future Work

The preliminary results of Methusela with XCS to solve the dynamic load balancing problem showed that it can achieve similar results and even perform better than a well designed analytic criterion. This is done even without the same amount of information of the analytic criterion. It is probably achieved because XCS is performance-driven allowing it to discover when certain evaluation parameter is useful and when is not. . Further to this XCS has the ability of generalize rules or turn one more specific if needed. It results that XCS can evolve rules to explore temporary situations that the analytic criterion is not able to use. As future work we intend to extend the number of parameters used by XCS, deciding what parameters are useful and if the cost of obtaining it is worthwhile, allowing the creation of even more effective decision rules. Those rules can be used to design better migration policies and new analytic criteria. We also intend to implement a XCS scheduler in a real environment aiming to achieve better results exploring temporary behavior of the system.

Acknowledgments

The authors would like to thank Harchol-Balter and Downey for their Methusela source code and Alwyn Barry for his XCS source code. We also would like to thank the anonymous reviewers for their helpful comments.

References

[BAU95] BAUMGARTNER , J., COOK , D. J., and SHIRAZI, B. Genetic solutions to the load balancing problem, Proc of ICPP 95 Workshop, pp 72-78.

[BAR99] BARAK, A., LA'ÁDAM O. and SHILOH A. Scalable Cluster Computing with MOSIX for LINUX, Proc .Linux Expo '99, pp. 95-100, Raleigh, U.S.A., May 1999.

[BAR00] BARRY , A.M. "XCS Performance and Population Structure within Multiple-StepEnvironments", PhD Thesis, Queens University Belfast, Sept 2000.

[CAR94] CARTER, E.F, 1994; The Generation and Application of Random Numbers , Forth Dimensions Vol XVI Nos 1 & 2, Forth Interest Group, Oakland California

[CAS88] CASAVANT, Thomas L., KUHL Jon G .,A Taxonomy of Scheduling in General-Purpose Distributed Computing System, IEEE Transactions on Software Engineering, Vol 14, No. 2, Febuary 1988.

[COR00] CORRÊA, Jan M., MELO, Alba C. , Algoritmos Genéticos para Escalonamento de Processadores, Workshop em Sistemas Computacionais de Alto Desempenho, São Pedro, Brasil 2000

[COR01a] CORRÊA, Jan M., MELO, Alba C., Using a Classifier System in a Dynamic Changing Environment :An Application to Dynamic Load Balancing to appear in Encontro Nacional de Inteligência Artificial, Fortaleza, Julho, 2001

[COR01b] CORRÊA, Jan M., MELO, Alba C., Using a Classifier System to Improve Dynamic Load Balancing, to appear in 30ª- International Conference on Parallel Processing, Valencia, Spain, September, 2001

[COR99] CORRÊA, Ricardo C.; FERREIRA, Afonso; REBREYEND, Pascal Scheduling Multipocessors Tasks with Genetic Algorithms, IEEE Transaction On Parallel and Distribute Systems, Vol 10, No.8, pp 825-837 August 1999

[COU96] COUCH, Alva L. Visualizing Huge Tracefiles with Xscal,Tenth USENIX System Administration Conference Chicago, IL, USA, Sept. 29 - Oct. 4,1996.

[DUS98] DUSSA-ZIEGER, K. and SCHWEHM, M. Scheduling of Parallel Programs on Configurable Multiprocessors by Genetic Algorithms, Journal of Approximate Reasoning, Special Issue on `Approximative Methods in Scheduling', Vol 19 (1-2) 23-38, 1998.

[GOL89] GOLDBERG, David E., Genetic Algorithms in Search, Optimization, And Machine Learning, Addison-Wesley, 1989.

[GRA99] GRAJCAR, M. Genetic List Scheduling Algorithm for Scheduling and Allocation on a Loosely Coupled Heterogeneous Multiprocessor System; Proceedings of the 36th Design Automation Conference (DAC), New Orleans, 1999

[HAR96] HARCHOL-BALTER, Mor and DOWNEY, Allen B. Exploiting process lifetime distributions for dynamic load balancing. In Proceedings of the 1996 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, pages 13--24, May 1996.

[HOL78] HOLLAND, J. H. and REITMAN, J. S.. Cognitive systems based on adaptive algorithms, Pattern-directed inference systems. Academic Press, New York, 1978.

[MIC96] MICHALEWICZ, Z. Genetic Algorithms + Data Structures = Evolution Programs, Springer-Verlag, 1996, .

[PAP98] PAPADIMITRIOU, C. and STEIGLITZ, K. Combinatorial Optimization: Algorithms and Complexity, Dover Publications Inc.,1998

[SAN97] SANDNES, F.E. and MEGSON, G.M. Improved Static Multiprocessor Scheduling Using Cyclic Task Graphs - A Genetic Approach. IPPS'97 Workshop on Randomised Parallel Computing, 1997.

[SAX99] SAXON, S, and BARRY A. (1999). XCS and the Monk's Problem. Second International Workshop on Learning Classifier Systems (IWLCS-99), Orlando, FL, USA, July 13, 1999.

[SUN97] SUNG-HO Woo; SUNG-BONG Yang; SHIN-DUG Kim; TACK-DON Han, "Task scheduling in distributed computing systems with a genetic algorithm ",Proceedings of the High-Performance Computing on the Information Superhighway, HPC-Asia '97, 1997.

[TOM99] TOMLINSOM, A. and BULL, L. A corporate XCS. In Proceedings of the1999 Genetic and Evolutionary Computation Conference Workshop, pages 298--305, Morgan Kaufmann, San Francisco, California.1999

[WIL95] WILSON, S. Classifier fitness based on accuracy . Evolutionary Computation, 1995.

[WU97] WU, M. On Runtime Parallel Scheduling for Processors Load Balancing, IEEE Transactions on Parallel and Distributed Systems, v.8, n.2, February, 1997