A Genetic Exploration of Dynamic Load Balancing Algorithms

Mohammed Aldasht, Julio Ortega, Carlos G. Puntonet; Antonio F. Díaz

Department of Computer Architecture & Technology
University of Granada, E-18071 Granada, Spain
{mohammed, julio, carlos, afdiaz @atc.ugr.es}

Abstract – Evolutionary algorithms provide ways to explore wide search spaces. Thus, it is possible to get some conclusions about the characteristics of these spaces in order to aid in the determination of the best alternatives to solve the problem at hand. We have applied a genetic algorithm to assess the problem of distributed load balancing in parallel processing. To do that, we propose a classification of the space of design of distributed load balancing algorithms that takes into account the different alternatives for each dimension of the algorithm. This classification allows the codification of each load balancing strategy, thus making possible to apply a genetic search to determine the distributed load balancing procedure that provides the best performance for the type of parallel application at hand and the parallel platform where it is implemented. As an example, in this paper we provide the results corresponding to the parallel multiplication of matrices implemented in a cluster.

I. INTRODUCTION

Load balancing means to distribute the workload of a parallel application among the processors in the platform at hand according to their relative performance, in order to minimize the execution time of the program [10]. Load balancing algorithms can be classified into three main classes: static algorithms, dynamic algorithms, and adaptive algorithms [1]. *Static algorithms* decide how to distribute the workload according a prior knowledge of the problem and the system characteristics. *Dynamic algorithms* use state information to make decisions during program execution. Finally, *Adaptive algorithms* are a special case of dynamic algorithms. They dynamically change its parameters in order to adapt its behaviour to the load balancing requirements.

Moreover, dynamic load balancing strategies can be divided basically into two main classes: *centralized* dynamic load balancing and *distributed* dynamic load balancing. These strategies define where the load balancing decisions are made. In a centralized scheme, the load balancer is implemented on one master processor and all decisions are made there. In a distributed scheme, the load balancer is replicated on all processors [2]. In this paper we consider this last scheme. Thus, we present a taxonomy of the different distributed load balancing procedures that we call Distributed Load Balancing Classification (DLBC). It allows the description and codification of the distributed load balancing design space, thus making possible the use of genetic algorithms to find optimal procedures in this taking into account the class of parallel algorithms and platforms at hand.

In this paper, section II and III describe the way the DLBC has been defined and the genetic algorithm used to explore the space of distributed load balancing procedures, respectively. Section IV gives the experimental results obtained and the corresponding comments, and finally, section V provides the conclusions of the paper.

II. DESCRIPTION OF DLBC

The classification of the distributed load balancing algorithms, that we have called DLBC, is built by taking into account the four policies that have to be defined in any dynamic load balancing strategy [1]: *information* policy, *transference* policy, *location* policy, and *selection* policy. In the following we describe these policies:

1. Information Policy (IP)

This policy determines how information should be exchanged among the processing nodes. According to the information collected from other nodes, processing nodes decide the nodes where the tasks should be transferred. The exchange of information can be either *periodic* or *on demand*.

The information is exchanged periodically if the processing node interchanges load information according to a *load balancing frequency*. This policy will be coded as IP=1. On the other hand, the *on demand* information policy collects load information whenever load balancing is required. We have coded this policy as IP=0. In our distributed load balancing model, the information about the workload distribution is represented by a *load index* in each processor that indicates the work allocated to that processor.

2. Transference Policy (TP)

This policy determines when tasks should migrate. It is interrelated with the location policy because it uses the location policy to determine the processes that should take part in the task transference operation. The task transference policy can be *sender initiated* (coded as 0), in which, the node that initiates the transference operation is the one who needs to send work to others; *receiver initiated* (coded as 1), in which the node that starts the transference operation is the one who needs to receive work from other sender nodes; or s*ymmetrical initiated* (coded as 2) where the task transference operation is to be initiated by the sender and the receiver at the same time.

3. Location Policy (LP)

Determines which node is to be incorporated in the load balancing operation. This can be decided according to the information collected by the information exchange policy. There are three possible location policies: t*hreshold-based*, where the location of a node to send or receive work is done based on one or two thresholds. In the *one-threshold* case (coded as 0) a node is selected as sender or receiver if its load is more or less than the threshold value, respectively. In the *two-threshold* (coded as 1) case a node is selected as receiver if its load is below the lower threshold; on the other hand, it is selected as a sender if its load is above the higher threshold. The third type of location policy is *shortest-based* (coded as 2); where the sender node tries to find the node with lowest load to select it as a receiver. The fourth type is r*andom-based* (coded as 3), in which a node is randomly selected, to send or receive work.

4. Selection Policy (SP)

This policy determines which task is to be transferred from the current node. There are two alternatives. The first one is the *pre-emptive* task selection (coded as 0), where the tasks currently under execution can be selected for the transference. The second alternative is the *non-pre-emptive* task selection (coded as 1), in which only the tasks that are waiting to be executed can be selected for the transference.

Figure 1 shows the feasible combinations of the above three policies. Table 1 also provides the feasible combinations of the four policies (thus a different load balancing algorithm) and the load balancing parameters defined below: the threshold parameters, TS1 and TS2, the granularity parameter, GRN, and the load balancing frequency, LBF. These parameters need to be set the alternatives marked in the table. Thus, Table 1 has been derived tacking into account the different types of load balancing algorithms defined by the feasible combinations of different policies [5,6,10]. In this paper we are going to analyse the load balancing algorithms corresponding to the shaded rows.

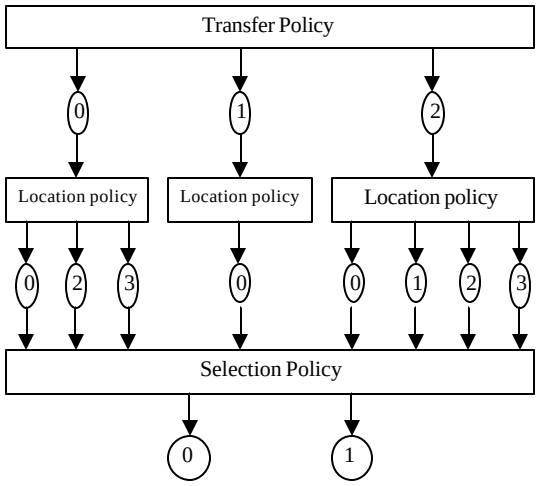

Figure 1. TP, LP, and SP Dependency Relation

TABLE 1. COMBINATIONS CORRESPONDING TO THE DIFFERENT POLICIES FOR ADAPTIVE DISTRIBUTED LOAD BALANCING

IP	TP	LP	SP	TS1	TS2	GRN	LBF
0	0	0	0	X		X	X
			1	X		X	X
		2	0			X	X
			1			X	X
		3	0			X	X
			1			X	X
	1	0	0	X		X	X
			1	X		X	X
1	2	0	0	X		X	X
			1	X		X	X
		1	0		X	X	X
			1		X	X	X
		2	0			X	X
			1			X	X
		3	0			X	X
			1			X	X

As Table 1 shows, there are three main groups of algorithms, defined by taking into account the transference policy (TP): sender initiative algorithms, receiver initiative algorithms and symmetrical initiative algorithms.

Sender initiative algorithms use on demand information exchange, and one of three location policies: one-threshold-based, shortest-based or random-based. The possible task selection policy can be one of two: pre-emptive or non-pre-emptive.

Receiver initiative algorithms also use on demand information exchange but the use only one location policy, a one-threshold-based location. They ca use one of two selection policies: pre-emptive or non-pre-emptive.

Symmetrical initiative algorithms use periodic information exchange and one of the four location policies: one-threshold-based location, two-threshold-based location, shortest-based location, or random-based location.

Definition of LB parameters: A complete definition the search space requires that the domains of values for the LB parameters TS1, TS2, GRN, and LBF are determined. The values of some of these parameters depend on the size of the problem at hand. In this work, the problem size is noted as N, and it is used to normalize the LB parameters, as it can be seen in the following definitions:

1. *Load Balancing Frequency* (LBF). This parameter can vary between 1 and N/P, where P is the number of processes. Thus, a process in the system can attend to other processes in the load balancing operation each LBF times during the problem execution.
2. *Granularity* of the problem (GRN). This parameter determines into how many tasks the problem at hand is divided. It can vary between 1 and N. Thus, a low GRN values indicates a fine granularity, and a large value indicates a coarse granularity.
3. *Threshold* parameter (TS1), determines the threshold at which a process could be considered over-loaded or under-loaded. It is frequently used by load balancing procedures, and can vary between 1 and N/P.
4. *Two-threshold* parameter (TS2), determines the two thresholds used by the corresponding location policy, LP = 1.

III. THE GENETIC ALGORITHM

Genetic algorithms (GAs) can be considered as optimization techniques based on the concepts of natural selection and genetics [4]. A simple GA consists of four main steps: *initialization, evaluation, exploitation* and *exploration*. We have designed a GA to aid in the exploration of the search space defined by the DLBC, in order to find common characteristics for the optimal load balancing procedures corresponding to different problems and platforms. In this paper we have considered the six one-threshold-based algorithms that appear shaded in Table 1. Thus, our search space will be defined by three parameters, load balancing frequency (LBF), granularity (GRN), and threshold (TS1).

The three most common representations for a GA correspond to the use of binary, integer, or real number codes [6]. We use a real representation. In our GA, the number of genes in each individual is three: one represents the LBF parameter, another represents the GRN parameter, and the third represents the TS1 parameter. The pseudo-code of our GA is shown below:

Step1. Initialize parameters, population size, and probability of crossover and mutation
Step2. Random generation of an initial population such that all the genes are in the interval] 0...1[.
Step3. Evaluate the fitness (i.e. the average execution time of the parallel procedure when the load balancing strategy associated to each individuals) of the current population.
Step4. Select the new population using the roulette wheel method. The algorithm is forced to keep the best individual in the population.
Step5. Apply crossover (in a point) and mutation (Gaussian-distributed) with the corresponding probability.
Step6. Repeat Step3, Step4 and Step5 until the end condition is reached.

This genetic algorithm implements the searching operation in the space of the three parameters, LBF, GRN, and TS1 in order to determine their best values for the six dynamic load balancing algorithms that we considered here (shaded rows in Table 1). To give values to the genes, the size of the problem, N, has been taken into account. In our experiments, the population size is 20 individuals. The crossover transformation uses two parents and produces two children by exchanging the two parts of the parents that are defined by a random selection of a point to divide each individual. The probability to achieve the crossover operation between two parents has been set to 0.8 in our experiments.

Mutation has been applied with probability 0.3, by using a Gaussian distribution with mean 1 and standard deviation 0.4. These values take into account the range and the behaviour of the three parameters, LBF, GRN, and TS1.

IV. EXPERIMENTAL RESULTS

As a benchmark, in this paper we have used the matrix multiplication algorithm because of its simplicity and scalability. The product of two matrices A and B is defined by $c_{ij} = \sum_{k=1}^{n} a_{ik} \times b_{kj}$, where a_{ij}, b_{ij}, and c_{ij} is the element in the i'th row and j'th column of the matrix A, B, and C respectively, and C is the result matrix. For simplicity, square matrices are used in the simulations. Thus, the matrices A, B, and C are $n \times n$ matrices. The sequential algorithm of this matrix multiplication requires n^3 multiplications and additions, therefore its time complexity is $O(n^3)$.

The parallel platform we have used is cluster of six dual-processor SMP nodes. All executions have been done by considering a matrix size of 1000×1000 and P=12 processors.

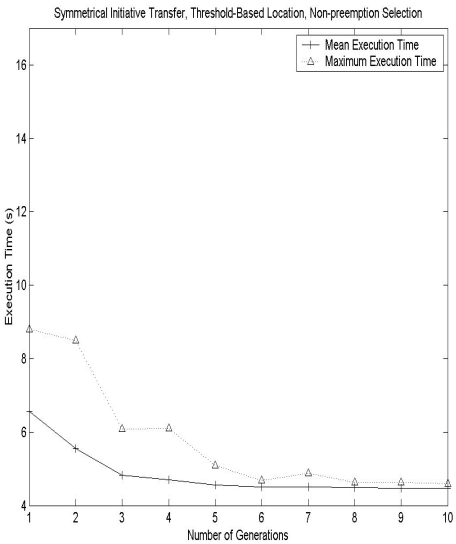

Figure 2. The Evolution of the GA in the case of Symmetrical Initiated with Task Non-pre-emption LB[1] Algorithm

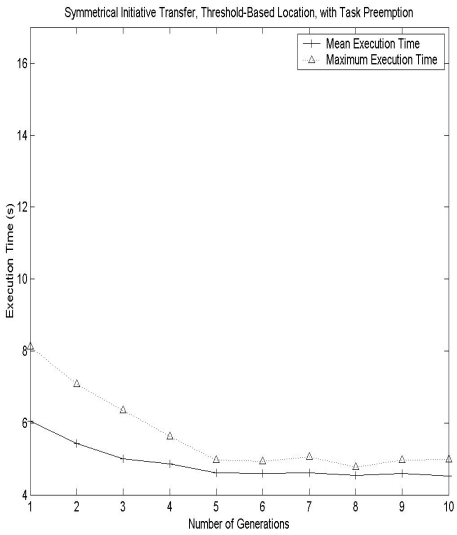

Figure 3. The Evolution of the GA in the case of Symmetrical Initiated with Task Pre-emption LB Algorithm

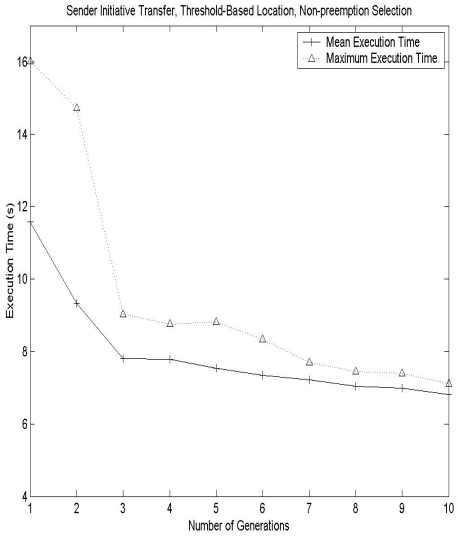

Figure 4. The Evolution of the GA in the case of Sender Initiated with Task Non-pre-emption LB Algorithm

[1] LB stands to Load Balancing

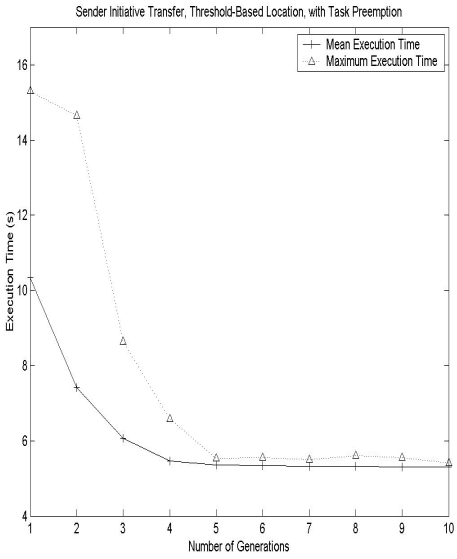

Figure 5. The Evolution of the GA in the case of Sender Initiated with Task Pre-emption LB Algorithm

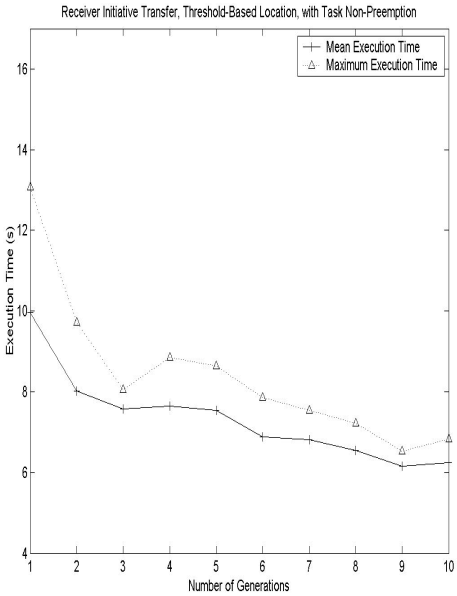

Figure 6. The Evolution of the GA in the case of Receiver Initiated with Task Non-pre-emption LB Algorithm

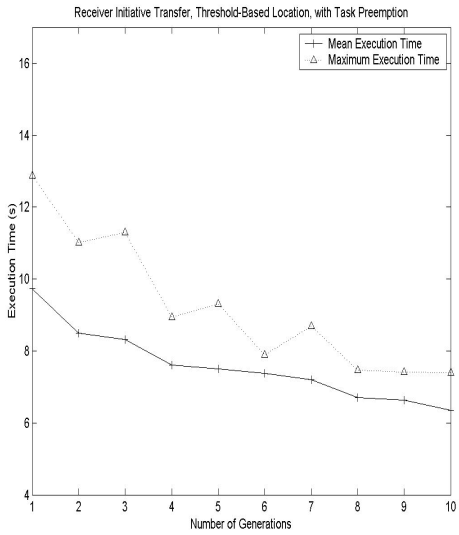

Figure 7. The Evolution of the GA in the case of Receiver Initiated with Task Pre-emption LB Algorithm

Figures 2 through 7 show the evolution of the GA in the six different algorithms, considered here. Table 2 shows the best values for the parameters LBF, GRN, and TS1. We have set the parameters in each LB procedure as a function of the problem size and number of processes, taking into account the limits indicated in section II, and by using the following equations:

$$LBF = \lceil FP * N / P \rceil$$
$$GRN = \lceil GP * N \rceil$$
$$TS1 = \lceil TP * N / P \rceil$$

where FP is the load balancing frequency parameter, GP is the granularity parameter, TP is the threshold parameter, N is the problem size and P is the number of processes.

Table 2 also provides the average execution time obtained at the end of the genetic optimization process for each load balancing algorithm. As it is shown, the symmetrical initiated load balancing procedures provide the best times, although their values of GRN and TS1 are quite different. The receiver initiated procedures provide similar results although they also have different values of TS1. With respect to the sender initiated procedures, they give different results depending whether task pre-emption or non-pre-emption is used. In this case, it is better to use pre-emption.

TABLE 2 BEST VALUES OF THE LBF, GRN, AND TS1.

	LBF	GRN	TS1	Exec. Time (s)
Alg. 1	4	341	3	4.31
Alg. 2	3	630	24	4.35
Alg. 3	11	390	18	6.47
Alg. 4	1	397	8	5.13
Alg. 5	2	153	15	5.63
Alg. 6	4	169	5	5.66

Alg. 1: Symmetrical initiated with task non-pre-emption LB algorithm.
Alg. 2: Symmetrical initiated with task pre-emption LB algorithm.
Alg. 3: Sender initiated with task non-pre-emption LB algorithm.
Alg. 4: Sender initiated with task pre-emption LB algorithm.
Alg. 5: Receiver initiated with task non-pre-emption LB algorithm.
Alg. 6: Receiver initiated with task pre-emption LB algorithm.

V. CONCLUSION

We have introduced a suitable codification of the distributed dynamic load balancing strategies according to the taxonomy of the DLB algorithms we have described. From the experimental results shown, we can see that the GA converges faster to the optimum solution in the case of DLB algorithms that use pre-emption of the migration task. Moreover, in general, the symmetrical initiated algorithms provide the best execution time for the problem and parallel platform we have considered. The reason of this behaviour comes from the use of a periodic information exchange policy in the symmetrical initiated algorithms. This information policy maintains the load information corresponding to the other processes always updated, thus helping the load balancer to select a suitable process for the task transference. Although this information policy presents more communication overheads, the communication network in our parallel platform (Gigabit Ethernet) provides a sufficiently good bandwidth. It is foreseen that on parallel platforms with lower bandwidth networks, this type of policy does not present this behaviour compared with the other alternatives of Table 2.

In the case of sender initiated algorithms, the pre-emption of migration tasks has a positive effect on the execution time.

We can note in table 2, that the best LBF and TS1 are closed and around 1% of the problem size, while the best GRN is between 30% and 40% of the problem size.

Acknowledgements. This paper has been supported by the Spanish *Ministerio de Ciencia y Tecnología*, under grant TIC2001-2845.

BIBLIOGRAPHY

[1] Luis Paulo Peixoto Dos Santos, Load Distribution: A Survey, Technical Report, Universidade do Minho, Departamento do Informática Oct. 1996.

[2] Shahzad Malik, Dynamic Load Balancing in a Network of Workstations, 95.515F Research Report, Nov. 2000.

[3] Jeffrey Moyer, Transparent Process Migration on the Beowulf System, Copyright © 1993, 1994, 1995, 1996, 1997, Nikos Drakos, Computer Based Learning Unit, University of Leeds. http://segfault.dhs.org/ ProcessMigration/Proposal/ Jan. 1999.

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

[5] Albert Y. Zomaya, Chris Ward and Ben Macey, Genetic Scheduling for Parallel Processor Systems: Comparative Studies and Performance Issues, IEEE Trans. Parallel and Distributed Systems, Vol. 10, no. 8, pp. 795-812, Aug. 1999.

[6] Albert Y. Zomaya, and Yee-Hwei Teh, Observations on Using Genetic Algorithms for Dynamic Load-Balancing, IEEE Trans. Parallel and Distributed Systems, Vol. 12, no. 9, pp. 899-911, Sep. 2001.

[7] L. Davis, Handbook of Genetic Algorithms, van Nostrand Reinhold, New York, 1991.

[8] Thyagaraj Thanalapati and Sivarama Dandamudi, An Efficient Adaptive Scheduling Scheme for Distributed Memory Multicomputers, IEEE Trans. Parallel and Distributed Systems, Vol. 12, no. 7, pp. 758-768, July 2001.

[9] Yair Wiseman and Dror G. Feitelson, Paired Gang Scheduling, IEEE Trans. Parallel and Distributed Systems, Vol. 14, no. 6, pp. 581-592, June 2003.

[10] Mohammed Javeed Zaki, Wei Li, and Srinivasan Parthasarathy, Customized Dynamic Load Balancing for a Network of Workstations, Journal of Parallel and Distributed Computing 43, pp. 156–162 (1997).