Dynamic Load Balancing in Parallel Processing on Non-Homogeneous Clusters

Armando E. De Giusti, Marcelo R. Naiouf, Laura C. De Giusti, Franco Chichizola
Instituto de Investigación en Informática LIDI (III-LIDI)
Facultad de Informática – UNLP
{degiusti, mnaiouf, ldgiusti, francoch}@lidi.info.unlp.edu.ar

This project is financially supported by the CIC and the YPF Foundation.

ABSTRACT

This paper analyzes the dynamic and static balancing of non-homogenous cluster architectures, simultaneously analyzing the theoretical parallel Speedup as well as the Speedup experimentally obtained.
Three interconnected clusters have been used in which the machines within each cluster have homogeneous processors although different among clusters. Thus, the set can be seen as a 25-processor heterogeneous cluster or as a multi-cluster scheme with subsets of homogeneous processors.
A classical application (Parallel N-Queens) with a parallel solution algorithm, where processing predominates upon communication, has been chosen so as to go deep in the load balancing aspects (dynamic or static) without distortion of results caused by communication overhead.
At the same time, three forms of load distribution in the processors (Direct Static, Predictive Static and Dynamic by Demand) have been studied, analyzing in each case parallel Speedup and load unbalancing regarding problem size and the processors used.

Keywords: *Parallel Systems. Cluster Architectures. Parallel Algorithms. Dynamic and Static Load Balancing. Parallel Speedup. Homogeneous and Non-Homogeneous Processors.*

1. INTRODUCTION

1.1. Cluster and Multi-Cluster Architectures

A cluster is a type of parallel/distributed processing architecture consisting of a set of interconnected computers that can work as a single machine [20]. The machines that make up a cluster can be homogeneous or heterogeneous, this being an important factor for the analysis of performance that can be obtained from a cluster as a parallel machine [1] [4] [8].

A multi-cluster architecture consists in interconnecting two or more clusters to configure a new parallel machine. In this configuration, each intervening cluster can conceptually be seen as a multiprocessor machine with certain performance parameters, interconnected to other multiprocessor machines to obtain a single global architecture capable of carrying out parallel processing by combining the resources of each cluster. The characterization of global performance parameters of a multi-cluster is complex owing to the number of intervening clusters, the degree of heterogeneity of processors and the inter-cluster communication system [14][19]. On occasions, a combination of interconnected homogeneous clusters, configuring a heterogeneous multi-cluster is used. Although the processing model can be simplified resorting to a "super-cluster" with a processor with an interconnected cluster, the communication model is still complex and even inter-cluster communication can have a fixed band width or one that depends on the general flow of communications (e.g. clusters interconnected via the Internet) [26].

1.2. Master-Slave Scheme with Multi-Cluster Architecture

The use of a Master-Slave paradigm with Multi-cluster Architecture provides at least two possibilities:

- If a single Master M processor - part of one the clusters of the system - is used, both its performance and the communication time from any other multi-cluster node need to be characterized.
- If a Master M_i processor per cluster is used, an interaction model for the M_i should be defined so as to control information updating and communications among processors from different clusters. The scheme of the relation among M_i can be hierarchical or peer-to-peer. Again, the different communication times involved should be analyzed.

Data and processes dynamic migration among multi-cluster nodes will have a different scheme depending on the two models adopted.

1.3. Load Balancing in Heterogeneous Architectures

The load balancing of an application has a direct impact on the speedup to be achieved as well as in the performance of the parallel system [8][16].

Usually, when working with heterogeneous clusters, the different calculation powers of the intervening machines are a factor that can be computerized to analyze the distribution of the work to be done.
For the type of known work problems (e.g. matrix multiplication) a "predictive" static load balancing considering the calculation power of the multi-cluster processors can be obtained; however, many real problems have a variable or dynamic workload depending on the data [2][9][18][27]. In these cases, it is necessary to adjust data or processes allocation dynamically while the application is being executed.
Note that any "predictive" load balancing formula should compute not only the calculation power, but also other factors proper of the architecture, such as the size and access time at the different levels of memory of each processor. [17][28]. In this paper, only the calculation power of each processor has been taken into account.
Besides, in a multi-cluster scheme in which applications are resolved with the Master-Slave paradigm, any dynamic balancing solution used, implies a communication overhead that will be affected by the complexity of the communication scheme among the nodes of the different clusters, as mentioned before.

1.4. Types of Problems with Variable Workload.

There are certain types of data parallelism problems for which it is possible to perform a static balancing allocation of the total workload. In these cases, provided there is a heterogeneous architecture, it will be possible to define a predictive $F(P_i,Wt)$ function where P_i is the calculation power of processor i and Wt the total work This function allows to distribute data "a priori" among processors [21].
If there is a variable workload due to the data particular characteristics (e.g. data arrangement, identification of image patterns), it is not possible to have a predictive function that assures load balancing among processors. Thus, it will be necessary to have a dynamic allocation policy that can be combined with a predictive initial distribution of a percentage of the total data [6][13][18].
Any dynamic allocation policy used implies some overhead degree of communication, which will be more complex to model and predict in a heterogeneous multi-cluster architecture.

2. CHARACTERIZATION OF TYPE OF APPLICATION OF INTEREST.

As analyzed in the introduction, there are different research axes on dynamic load balancing problems in multi-cluster architectures.
An architecture model in which heterogeneity appears only in machines with different clusters and can be compared to a calculation power function of the machines of each cluster has been determined.
Besides, it has been chosen to work with the Master-Slave paradigm with one Master (usually located at the cluster with better processing services).
Finally, the focus of this experimental work has been put on one type of the problems in which communication time among Tc processes is not significant, considering Tp ($Tp >> Tc$) local processing time. This restriction allows to identify the differences among the static and dynamic load balancing schemes more clearly without overlapping an important communication overhead.

3. LOAD DISTRIBUTION MODELS TO BE STUDIED AND THEORETICAL SPEEDUP TO BE ACHIEVED

Three ways of data parallelism implementation will be used:

- *Direct Static Distribution (DSD)* where the total workload Wt will be allocated to the architecture B processor in a homogeneous manner, so that each processor will have Wt/B, regardless the $F(P_i,Wt)$ function.
- *Predictive Static Distribution (PSD)* where the total workload Wt will be allocated to the architecture B processor at the moment of starting the application, according to the prediction $F(P_i,Wt)$ function.
- *Dynamic Distribution upon Demand (DDD)* where a Li percentage of the total Wt workload will be allocated to the architecture B processor at the moment of starting the application, according to the prediction $F(P_i,Wi)$ function and then, each processor will demand more work on the part of the Master, as its task is being completed.

The Li value and the amount of additional work to be allocated to each processor on demand are experimental research parameters that depend on the application and the relation between Tp and Tc.

The theoretical Speedup to be achieved by multi-cluster architecture will be a $G(Pi)$ function. The experimental measuring of the real Speedup should directly correlate with the degree of balancing achieved with the total Wt work allocation during the execution of the application.

4. CONTRIBUTION OF THIS WORK

A Master-Slave model with 3 heterogeneous clusters among them, each one having 8 homogeneous machines operating as a (B=24) multi-cluster with an additional processor as Master has been studied. The processors heterogeneity was analyzed, considering one of the functions of their calculation power.

One problem case was studied, which responded to the hypothesis *Tp* >> *Tc*, with the three load distributions proposed (*DSD*, PSD, *DDD*) to carry out the data parallelism, specially analyzing the theoretical parallel Speedup. This Speedup was achieved in view of the calculation power of the processors, and the load unbalancing taking into account the parameters *B*, *Wt*, P_i y *Li* mentioned before.

5. APPLICACIÓN TO PARALLEL SOLUTION ON A HETEROGENEOUS MULTI-CLUSTER OF THE N-QUEENS PROBLEM

The *N*-queens problem consists in placing *N* queens on an *N*x*N* board in such a way that they do not attacks one another [5][7][12]. A queen attacks another one if they are in the same diagonal, row or column.

5.1 Sequential Solution

An initial solution to the *N*-queens problem, using an sequential algorithm, consists in trying all possible location combinations of the queens on the board, keeping those that are valid and disrupting the search whenever this is not achieved.
Considering that a valid combination can generate up to 8 different solutions, which are rotations of the same combination, the number of distributions to be evaluated can be reduced. The best sequential algorithm found for this problem is based on this fact [3][23][24]. There follows a brief description of said algorithm on an *N*x*N* board.

The algorithm performs *N*/2 iterations, and each one places the queen in a different position on the first row. The remaining *N*/2 positions are not evaluated for they are symmetric combinations of the previous ones.

The vector of valid positions for the following row is determined from the queen placed on the first row, and for each of them, the solutions that they themselves generate are determined (Figure 1a). To determine the number of solutions as from row i (in such a way that the whole row j with j ≤ i has its queen placed), the vector of valid positions for row i+1, where the same step is repeated for each of them, is determined. This continues until a queen is placed on the last row, or until no more valid positions are left on a certain row (Figure 1b). When placing a queen on the last row, the number of different solutions that such combination and its symmetric one generate when rotated at 90º, 180º y 270º are estimated (Figure 1c).

```
main ()
{ quantSol:=0
  for (pos= 1..N/2)
     placeQueen (1,pos,board)
     detPosValid (posValid,board,2)
     quantSol:=quantSol + detSol (2,posValid,board)
}
```

(a)

```
function quantSolutions (b)
{ if (rot(b,90)=b) or (rot(b,90)=sym(b)) then return (2)
  else if (rot(b,180)=b) or (rot(b,180)=sym(b)) then return (4)
        else return (8)
}

rot(b,d): returns the board b rotated in d degrees.
sym(b): returns the symmetrical board of b.
```

(b)

```
function detSol (row, posValid, board)
{ i:= positionValid (posValid)
  if (row = N) and (i < N ) then
     placeQueen (row,i,board)
     return (quantSolutions (board))
  else
     total:=0
     while (i < N)
        placeQueen (row,i,board)
        detPosValid (newPosValid,board, row+1).
        total:= total + detSol (row+1,newPosValid,board)
        i:= positionValid (posValid)
     return total
}

detPosValid (p,b,r): determines the set p of valid positions for
the row r in the board b.
positionValid (p): returns the first valid position in p.
```

(c)

Figure 1. Sequential Solution Pseudocode

5.2. Parallel Solution proposed based on the Function of the Load Distribution Models

For the parallel solution of this problem, the queen is placed on one or more rows, and all the solutions for that initial arrangement are obtained. Each processor is in charge of solving the problem for a subset of said solutions, in this way, the whole system works with all the possible combinations of those rows. Two things are to be taken into account:

- How to make the combinations.
- How to distribute the combinations among machines.

5.2.1. How to make the Combinations

When working with a heterogeneous architecture, the amount of work (combinations) that each processor must solve vary according to the existing relation regarding calculation power. To be able to distribute the work in a balanced way, it is convenient to use "fine grain", that is, many combinations of little work each, so as to level up the work done by each machine, and resolve several of them [10]. To this aim, the first three rows are used to form each of the combinations to resolve.
In this way, different N^3 combinations are obtained to be distributed among all the heterogeneous processors, *N* being the board size.

5.2.2. How to Distribute Combinations

5.2.2.1- Direct Static Distribution (DSD)

In this type of distribution, each processor m_i calculates the combinations to be resolved at the beginning of its execution. These are distributed in a cyclic manner among the machines, so that each m_i resolves *Wt/B* combinations distributed cyclically.

5.2.2.2. Predictive Static Distribución (PSD)

In this type of distribution, each processor m_i determines the combinations to be resolved at the beginning of its execution. To that purpose, the following steps are carried out:

- Obtain the relative number of combinations cr_j $\forall$ j = 1..*B*.

$$cr_j = \frac{P_j}{P_{lessPowerfulMachine}}$$

- Determine the number of combinations *A* that form a block.

$$A = \sum_{j=1}^{B} cr_j$$

- Calculate for each block, the combinations to be done, considering that they are allocated cyclically to each machine m_j that has not carried out any cr_j combinations in that block.

Figure 2 shows how the distribution is carried out for a six heterogeneous machine (two for each cluster) system in which the relative number of combinations are m_1=3, m_2=3, m_3=2, m_4=2, m_5=1 and m_6=1.

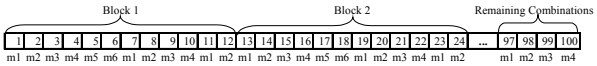


Figure 2: Predictive Static Distribution

Figure 3 shows the complete algorithm pseudocode for this distribution:

```
main ()   // processor 1
{ quantSol:=0
  while  (processor 1 has combinations)
    determines the location for row 1 (p1)
    determines the location for row 2 (p2)
    determines the location for row 3 (p3)
    quantSol:=quantSol+detSolPartial (p1,p2,p3,board)
  for (i=2..B)
    recv(quantOtherProc,i)
    quantSol:=quantSol+quantOtherProc;
 }
```

(a)

```
main ()   // processor i, where i = 2..B
{ quantSol:=0
  while  (processor i has combinations)
     determines the location for row 1 (p1)
     determines the location for row 2 (p2)
     determines the location for row 3 (p3)
     quantSol:= quantSol+detSolPartial (p1,p2,p3,board)
  send(quantSol,1)
 }
```

(b)

```
function detSolPartial (posRow1, posRow2, posRow3, board)
{ placeQueen (1,posRow1, board)
  placeQueen (2,posRow2, board)
  placeQueen (3,posRow3, board)
  detPosValid (posValid, board,4).
  return detSol  (4, posValid, board)
}

detPosValid (p,b,r): determines the set p of valid positions for
the row r in the board b
```

(c)

Figura 3. Pseudocode for PSD_p. The detSol function is that of Figure 1 (b).

5.2.2.3. Dynamic Distribution upon Demand (DDD)

Given *C* combinations to be calculated by *B slave* processors, and a *master*, the algorithm carries out the following steps:

- The master processor (m_0) distributes the initial combinations:
  - m_0 calculates the number of combinations (*Ci*) to be initially distributed.

$$Ci = \frac{Wt \times Li}{100}$$

  - m_0 obtains the number of initial cc_i combinations that correspond to the m_i processor according to P_i

$$powTotal = \sum_{i=1}^{B} p_i$$

$$cc_i = \frac{Ci \times P_i}{powTotal}$$

  - m_0 distributes the initial *Ci* combinations allocating consecutive cc_i combinations to m_i, with i = 1..*B*.

- The master processor (m_0) distributes the remaining *Cr* combinations, where *Cr* = *C* – *Ci*.
  - m_i requests m_0 more combinations to resolve when its work has finished. For any i = 1..*B*.
  - m_0 sends consecutive *Cb* combinations to the m_i processor that has made the request if there is still work to be done.
  - If *Cb* > 1, m_0 decreases its value in one point

Figure 4 shows how the distribution is done for a system consisting of six slave heterogeneous machines (two for each cluster), in which the relative calculation powers are m_1=3, m_2=3, m_3=2, m_4=2, m_5=1 and m_6=1. *Li* = 12 %, and *Cb* = 5.

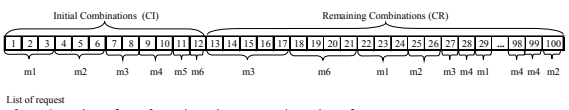


Figure 4: Dynamic Distribution upon Demand

Figure 5 presents the algorithm pseudocode for *DDD*.

```
main ()   // processor 0
{ quantSol:=0
  for (i=1..B)
     determines the first combinations (first)
     determines the last combinations (last)
     send(first,last,i)
  while  (has combinations)
    recv(request,i)
    determines the first combinations (first)
    determines the last combinations (last)
    send(first,last,i)
  for (i=1..B)
    send(end,end,i)
    recv(quantOtherProc,i)
    quantSol:=quantSol+quantOtherProc;
}
```

(a)

```
main ()     // processor i,  where i  = 1..B
{ quantSol:=0
  recv(first,last, 0)
  while  (first<>end)
     for (comb= first..last)
        determines the location for row 1 (comb, p1)
        determines the location for row 2 (comb, p2)
        determines the location for row 3 (comb, p3)
        quantSol:=quantSol+detSolPartial (p1,p2,p3, board)
     send(request,i,0)
     recv(first,last, 0)
  send(quantSol,0)
}
```

(b)

Figura 5. Pseudocode for DDD_d. The detSolPartial function is that of Figure 4.

6. EXPERIMENTAL RESULTS OBTAINED

In this section, the tests carried out are presented together with the results obtained, regarding the Speedup metrics and the unbalancing described below.

6.1. Metrics used

To measure the load unbalancing among the processors that intervene in a parallel application, the relative work difference obtained is calculated from the following formula [4][5][24].

$$Unbalance = \frac{\max_{i=1..B}(Trabajo_i) - \min_{i=1..B}(Work_i)}{average_{i=1..B}(Work_i)}$$

where $Work_i = machine\ time_i$.

The Speedup metrics is used to analyze the algorithm performance in the parallel architecture:

$$Speedup = \frac{SequentialTime}{ParallelTime}$$

In the case of a heterogeneous architecture, the "*Sequential Time*" is given by the time of the best sequential algorithm executed in the machine with the greatest calculation power [1][8][11].

To evaluate how good the speedup obtained is, it is compared with the theoretical speedup of the architecture upon which work is being carried out. The speedup considers the relative calculation power of each machine with respect to the power of the most powerful machine [25]. The theoretical Speedup is calculated following this formula:

$$TheoreticalSpeedup = \sum_{i=1}^{B} P_i$$

where

B is the number of machines of the architecture used.

P_i is the relative calculation power of the machine i regarding the best machine power. This relation is expressed in the following formula:

$$P_i = \frac{sequentialTime(powerfulMachine)}{sequentialTime(m_i)}$$

6.2. Experiments

The experiments were done on a multi-cluster architecture consisting of three clusters:

- an 8 Celeron 2 Ghz homogeneous cluster of 128 Mb memory.
- an 8 Duron 800Mhz homogeneous cluster of 256 Mb memory.
- an 8 Pentium III 700 Mhz cluster homogeneous cluster of 256 Mb memory.

Communication within each cluster is done via an Ethernet web, using a switch for communication among clusters.

The language used for the implementations is C together with the MPI library to improve communications among processors [22].

Tests were carried out using 24 machines, adding one for the dynamic distribution, acting as master, and with different board sizes. (N=17,18,19,20,21).

In the case of dynamic distribution, it was experimented with different percentages of initial distribution (*Li*) and with a different initial number of combinations in the blocks to be distributed (*Cb*).
Li = 0, 25, 50 and 75.
Cb = 1, 5, and 10.

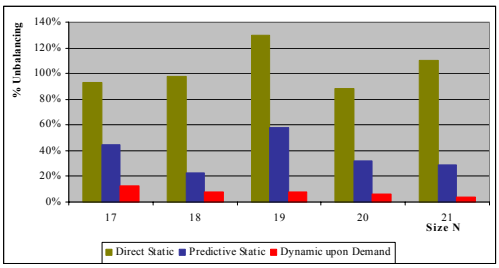
6.3. Results

The data of Table 1 shows the percentage of load unbalancing produced by the algorithm for the *Direct Static, Predictive Static* and *Dynamic upon Demand* distributions with different *Li* and *Cb* values. Some of these results can be seen in Graph 1.

Size	Direct Static	Predictive Static	Dynamic upon Demand					
			0% - 1	0% - 5	0% -10	25% - 1	25% - 5	25% -10
17	93%	45%	11%	11%	10%	11%	9%	61%
18	98%	23%	9%	10%	11%	11%	11%	11%
19	130%	58%	8%	9%	9%	9%	9%	9%
20	88%	32%	6%	6%	6%	8%	7%	37%
21	110%	29%	7%	6%	6%	5%	6%	4%

Size	Dynamic upon Demand					
	50% - 1	50% - 5	50% -10	75% - 1	75% - 5	75% -10
17	8%	13%	113%	39%	39%	47%
18	11%	8%	67%	35%	35%	40%
19	9%	8%	68%	20%	20%	37%
20	8%	6%	49%	36%	36%	43%
21	7%	4%	36%	30%	30%	31%

Table 1: Percentage of unbalancing for each test.



Graph 1: Load Unbalancing of Direct Static, Predictive Static and Dynamic upon Demand Distributions (with *Li*=25 and *Cb*=1). For *N*=17, 18, 19, 20, 21.

Table 2 presents the speedup obtained for each test mentioned before together with the optimal speedup (or theoretical) calculated for this machine combination. Table 3 shows the total time for each test.

Size	Optimum	Direct Static	Predictive Static	Dynamic upon Demand				
				0% - 1	0% - 5	0% -10	25% - 1	25% - 5
17	16	9.42	13.49	14.78	14.82	14.80	14.83	15.16
18	16	9.28	14.25	15.12	14.98	14.89	14.89	14.89
19	16	8.09	12.80	15.13	15.12	15.15	15.22	15.14
20	16	9.82	13.61	15.39	15.27	15.32	15.26	15.30
21	16	8.96	13.56	15.36	15.40	15.36	15.42	15.45

Size	Dynamic upon Demand						
	25% -10	50% - 1	50% - 5	50% -10	75% - 1	75% - 5	75% -10
17	10.22	15.16	14.61	7.72	12.19	12.19	12.19
18	14.91	14.90	15.12	9.82	12.43	12.43	12.34
19	15.15	15.17	15.20	9.85	13.89	13.89	12.27
20	12.02	15.25	15.43	11.09	12.39	12.38	11.89
21	15.68	15.39	15.54	12.03	12.73	12.73	12.67

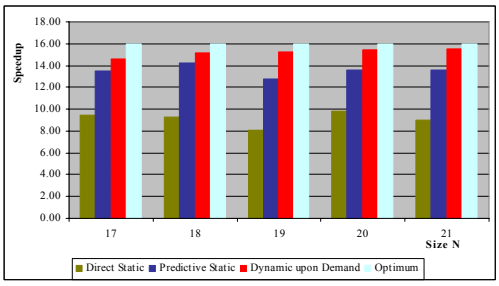
Table 2: Speedup.

Size	Direct Static	Predictive Static	Dynamic upon Demand					
			0% - 1	0% - 5	0% -10	25% - 1	25% - 5	25% -10
17	6.36	4.44	4.05	4.04	4.05	4.04	3.95	5.86
18	46.88	30.51	28.75	29.02	29.20	29.21	29.21	29.16
19	413.18	261.28	220.96	221.11	220.63	219.63	220.80	220.67
20	2735.21	1973.48	1745.32	1759.14	1752.54	1759.64	1755.23	2234.18
21	25296.55	16710.35	14752.70	14720.81	14750.63	14699.02	14673.81	14450.14

Size	Dynamic upon Demand					
	50% - 1	50% - 5	50% -10	75% - 1	75% - 5	75% -10
17	3.95	4.10	7.76	4.91	4.91	4.91
18	29.18	28.76	44.30	35.00	34.99	35.23
19	220.45	219.94	339.58	240.72	240.72	272.56
20	1760.48	1740.82	2422.17	2168.29	2169.02	2259.31
21	14721.65	14585.20	18846.13	17809.43	17810.01	17886.58

Table 3: Algorithm Total Time.

Graph 2 shows the speedup obtained with each of the distribution algorithms for some of the tests in Table 2, together with the optimal speedup of this architecture.



Graph 2: Speedup of the Direct Static, Predictive Static and Dynamic upon Demand Distributions (with *Li*=50 and *Cb*=5) and Optimum, for *N*=17, 18, 19, 20, 21.

From the results obtained, it can be seen that the speedup achieved with the dynamic upon demand distribution slightly increases as the size of *N* increases, in all cases being closer to the optimum. Besides, it can be seen that there is a significant difference regarding the speedup achieved by the algorithm that distributes in a direct static manner, and a little less for the predictive static distribution.

7. CONCLUSSION AND WORK GUIDELINES

Analyzing the results obtained from the experimental work, we can come to the following conclusions:

- Pure parallel solution (without considering work distribution) for the type of problems where *Tp*>>*Tc*, mainly the *N*-Queens requires a minimal communication among machines, thus making essential the choice of data distribution among clusters, to achieve an almost optimal Speedup.

- Naturally, algorithms that take into account the calculation power of each machine for work distribution have a better behavior than *Direct Static* distribution. This improvement is clearly expressed in the Load Balancing and the Speedup.

- Among the algorithm that take into account the calculation power, it can be seen that the algorithms that distribute dynamically can distribute work in a more balancing way among the machines (as seen in Graph 1), without much affecting the final time of execution (as shown by the speedup en Graph 2 and the data of Table 3).

- In dynamic distribution, the Speedup obtained is quite close to the optimum according to the parallel architecture used in this case.

At present, work is being done with a fourth cluster, using more powerful and sensitive machines than those of the three clusters used in this experimental work.
Likewise, tests are being done with clusters outside the UNLP, particularly at the UNSur (Bahía Blanca), UNComahue (Neuquen), UA Barcelona(Spain) and the Universidad Católica del Salvador (Brasil).

8. REFERENCIAS

[1] Al-Jaroodi J, Mohamed N, Jiang H, Swanson D. "Modeling Parallel Applications Performance on Heterogeneous System". IEEE Computer Society, 2003.

[2] Baiardi F, Chiti S, Mori P, Ricci L. "Integrating Load Balancing and Locality in the Parallelization of Irregular Problems". Future Generation Computer Systems, Elsevier Science B.V., Vol 17, 2001, pp 969-975.

[3] Bernhardsson B. "Explicit Solution to the n-queens Problems for all n". ACM SIGART Bulletin,2:7,1991.

[4] Bohn C, Lamont G. "Load Balancing for Heterogeneous Clusters of PCs". Future Generation Computer Systems, Elsevier Science B.V., Vol 18, 2002, pp 389-400.

[5] Bruen A, Dixon R. "Then n-queens Problem. Discrete Mathematics". 12:393-395, 1997.

[6] Campos L, Scherson I. "Rate of Change Load Balancing in Distributed and Parallel System". Proceeding of the 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing, San Juan, Puerto Rico, Pages 701-707. 1999.

[7] De Giusti L, Novarini P, Naiouf M, De Giusti A. "Parallelization of the N-queens problem. Load unbalance analysis". Workshop de Procesamiento Paralelo y Distribuido (WPPD), Congreso Argentino de Ciencias de la Computación (CACIC'03), 2003.

[8] Donaldson V, Berman F, Paturi R. "Program Speedup in a Heterogeneous Computing Network". Journal of Parallel and Distributed Computing 21:3 (6/1994), 316-322.

[9] Dongarra J, Foster I, Fox G, Gropp W, Kennedy K, Torczon L, White A. "The Sourcebook of Parallel Computing". Morgan Kauffman Publishers. Elsevier Science, 2003.

[10] Goldman. "Scalable Algorithms for Complete Exchange on Multi-Cluster Networks". In: CCGRID'02, IEEE/ACM, Berlín, p.286-287, 2002.

[11] Grama A, Gupta A, Karypis G, Kumar V. "Introduction to Parallel Computing". Second Edition. Pearson Addison Wesley, 2003.

[12] Hedetniemi S, Hedetniemi T, Reynolds R. "Combinatorial problems on chessboards: II". Chapter 6 in Domination in graphs: advanced topic, pag 133-162, 1998.

[13] Hui C, Chanson S. "Improve Strategies for Dynamic Load Balancing". IEEE Concurrency, pages 58-67. 1999.

[14] Jiang, Yeung. "Scalable Inter-Cluster Communication System for Clustered Multiprocessors". 1997.

[15] Jordan H, Alaghband G. "Fundamentals of Parallel Computing". Prentice Hall, 2002.

[16] Leopold C. "Parallel and Distributed Computing. A survey of Models, Paradigms, and Approaches". Wiley Series on Parallel and Distributed Computing. Albert Zomaya Series Editor, 2001.

[17] Menascé D, Almeida V. "Cost-Performance Analysis of Heterogeneity in Supercomputer Architectures". Proc. ACM-IEEE Supercomputing'90 Conference, New York, Nov 1990.

[18] Naiouf M. "Procesamiento Paralelo. Balance Dinámico de Carga en Algoritmos de Sorting". Tesis Doctoral. Universidad Nacional de La Plata, 2004.

[19] Ogura S, Nakada H, Matsuoka S. "Evaluation of the inter-cluster data transfer on Grid environment". Proceedings of CCGrid 2003 , pp. 374-381, May 2003.

[20] Pfister G. "In Search of Clusters". Prentice Hall, 2nd Edition, 1998.

[21] Ross K, Yao D. "Optimal Load Balancing and Scheduling in a Distributed Computer System". Journal of Association for Computing Machinery, 38 (3): 676-690.1991.

[22] Snir M, Otto S, Huss-Lederman S, Walker D, Dongarra J. "MPI: The Complete Reference". Cambridge, MA: MIT Press, 1996. Available in web site: http://www.netlib.org/utk/papers/mpi-book/mpi-book.html.

[23] Somers J. "The N Queens Problem a study in optimization". www.jsomers.com/nqueen_demo/nqueens.html.

[24] Takaken, "N Queens Problem (number of Solutions)". http://www.ic-net.or.jp/home/takaken /e /queen/.

[25] Tinetti F. "Cómputo Paralelo en Redes Locales de Computadoras". Tesis Doctoral. Univ. Autónoma de Barcelona, 2004. https://lidi.info.unlp.edu.ar/~fernando/publis/publi.html

[26] Vaughan F, Grove D, Coddington P. "Communication Performance Issues for Two Cluster Computers". Proceedings of the twenty-sixth Australasian computer science conference on Conference in research and practice in information technology, p.171-180, February 01, 2003, Adelaide, Australia.

[27] Watts J, Taylor S. "A Practical Approach to Dynamic Load Balancing". IEEE Transactions on Parallel and Distributed Systems, 9(3), March 1998, pp. 235-248.

[28] Zhang X, Yan Y. "Modeling and Characterizing Parallel Computing Performance on Heterogeneous Networks of Workstations". Proceeding of the 7th Symposium on Parallel and Distributed Processing. 1995.