A Practical Approach of Diffusion Load Balancing Algorithms

Emmanuel Jeannot[1] and Flavien Vernier[2]

[1] INRIA - LORIA, campus scientifique, BP 239, 54506 Vandoeuvre les Nancy, France
emanuel.jeannot@loria.fr
[2] UHP Nancy-I - LORIA, campus scientifique, BP 239, 54506 Vandoeuvre les Nancy, France
flavien.vernier@loria.fr

Abstract. In this paper, a practical approach of diffusion load balancing algorithms and its implementation are studied. Three problems are investigated. The first one is the determination of the load balancing parameters without any global knowledge. The second problem consists in estimating the cost and the benefit of a load exchange. The last one studies the convergence detection of the load balancing algorithm. For this last point we give an algorithm based on simulated annealing to reduce the convergence towards a load repartition in steps that can be done with discrete loads. Several simulations close this paper and illustrate the impact of the various methods and algorithms introduced.

1 Introduction

One of the most important problems in distributed processing consists in balancing the work load among all processors. The purpose of load (work) balancing is to achieve better performances of distributed computations, by improving load allocation. The load balancing problem was studied by several authors from different points of view [1,2,3,4,5,6,7].

In this paper we focus our study on the iterative load balancing algorithms introduced in [1]. These kinds of algorithms assume that a node manages its load only with its nearest neighbors. They are generic algorithms, useful when the system is decentralized or when some nodes cannot directly communicate with all the other nodes. However these algorithms face several problems. Firstly, the majority of studies about these algorithms use a global knowledge, like the network or the nodes properties, to determine the load balancing parameters. Secondly, most of these algorithms assume that balancing the load is always beneficial and leads to a reduction of the execution time. Thirdly, since the load is not infinitely divisible, the final load balancing (after convergence of the algorithm) can face a *step* problem.

In this paper we propose a practical approach of load balancing that solves the 3 above problems. They are 3 main problems that can appear in an implementation of these load balancing algorithms. To the best of our knowledge no load balancing algorithm of the literature can deal with these 3 issues at the same

W.E. Nagel et al. (Eds.): Euro-Par 2006, LNCS 4128, pp. 211–221, 2006.

time. We give methods to determine the diffusion parameters without any global knowledge. We propose an analysis of the cost and benefit of load exchange in order to determine when it is worth exchanging some load. The convergence of the load balancing algorithms with no infinitely divisible loads is also studied in this paper. Finally, the given methods are efficient and easy to implement.

It is important to note that, in this work, we make very few assumptions. We can deal with either static or dynamic load. The network topology can be of any type as long as it is connected. Nodes and networks can be homogeneous or heterogeneous. The notion of load is very abstract, it can be anything that just requires some time to be processed (data, etc.). The proposed methods deal with static networks but the adaptation to dynamic networks is straight forward. Finally, no global knowledge is required to process the algorithm. The knowledge is limited to the neighborhood. The obtained results give in the better case, performance gains greater than 100% and the algorithm does not always use all the available resources: it is able to find the right amount of resources that gives a good speed-up.

This paper is organized as follows. Section 2 presents the related works, we review the diffusion on any static network. In Section 3.1, we study the problem of the connection links heterogeneity. Section 3.2 presents a decentralized method to compute the load balancing parameters. Section 3.3 is dedicated to the not infinitely divisible loads and to the detection of the convergence of the load balancing algorithm. In Section 4 we illustrate the behavior of the load balancing algorithm according to the methods that we give by some experimentation.

2 Related Work

The studied algorithms are generally dedicated to static networks. A static network topology is classically represented by a simple undirected connected graph $G = (V, E)$, where V is the set of vertexes and E is the set of edges, $E \subseteq V \times V$. Each processor is a vertex of the graph and each communication link between two processors i, j is the edge $(i, j) \in E$ between the two vertexes i and j $(i, j \in V)$. Vertexes are labeled from 1 to n where n is the number of processors, hence $|V| = n$. Let m be the number of communication links ($|E| = m$). Let F be the vector of edge-weight and let us note $f_{i,j}$ the weight of edge (i, j) ($f_{i,j} = F_k | E_k = (i, j)$). Let C_n be the vector of node-weight such that the average of C_{n_i} is normalized $\frac{\sum_i C_{n_i}}{n} = 1$.

In [1], Cybenko introduced a distributed load balancing (LB) algorithm for static networks called the diffusion algorithm or FOS (First Order Scheme). It assumes that a process i balances its load simultaneously with all its neighbors. To balance the load, a ratio α_{ij} of the load difference between the process i and j is swapped between i and j. In the general case - on heterogeneous networks - the LB step of a process i with all its neighbors is given by Equation (1) where $w_i^{(t)}$ is the work load done by process i at time t.

$$w_i^{(t+1)} = w_i^{(t)} - \sum_j \alpha_{i,j}.f_{i,j}.\left(\frac{w_i^{(t)}}{C_{n_i}} - \frac{w_j^{(t)}}{C_{n_j}}\right) \quad (1)$$

Equation (1) is linear and thus it can be re-written in matrix form: $W^{(t+1)} = M^T W^{(t)}$, where $W^{(t)}$ is the vector $(w_i^{(t)})$ and M is the diffusion matrix defined by

$$m_{ij} = \begin{cases} \frac{\alpha_{ij} f_{i,j}}{C_{n_j}} & \text{if } (i,j) \in E \wedge i \neq j, \\ 1 - \sum_k m_{ik} & \forall k | (i,k) \in E \wedge i = j, \\ 0 & otherwise. \end{cases}$$

This algorithm has often been studied and derived - Dimension Exchange Algorithm [1,2,8], Second Order Scheme [9,4], dynamic networks [10,11,12] ...

In the literature, various methods can be found that determine these parmeters $\alpha_{i,j}$ or $f_{i,j}$. There are three classical methods to compute α: Cybenko Choice [1], Boillat Choice [5] or optimal Choice [13]. The optimal Choice and the Cybenko Choice need a global knowledge of the network and the Boillat Choice only needs a knowledge of neighbors degree to determine α: $\alpha_{ij} = \frac{1}{\max(d(i),d(j))+1}$, where $d(i)$ is the degree of node i at time t. The parameter $f_{i,j}$ must be determined according to the constraints of the diffusion matrix M, M must be stochastic, irreducible and aperiodic [1].

3 A Decentralized Practical Approach

3.1 Cost and Benefit of Load Balancing

Let us start by defining the cost and the benefit of a LB algorithm. The cost is the time lost by exchanging the load, it is generally due to communication. The benefit is the time gained by exchanging the load, it is due to a better balance. In Equation (1) the parameter $f_{i,j}$ corresponds to the weight of edge (i,j). Hence, the parameter $f_{i,j}$ must be determined such that the cost of the LB algorithm is lower than the benefit given by the exchange of load. In our practical approach, $f_{i,j}$ is in $\{0,1\}$. If the cost of an exchange $L_{ij}^{(t)}$ between i and j is greater than its benefit, then $f_{i,j}$ is set to 0 and there is no exchange between i and j, otherwise $f_{i,j}$ is set to 1. It can be noted that by this definition $f_{i,j}$ depends on the time, hence it becomes $f_{i,j}^{(t)}$ and its corresponding vector F becomes $F^{(t)}$.

The cost and the benefit of an exchange depends on the size of this exchange. To determine the cost of an exchange we give the following equation,

$$\text{Cost}(L_{ij}^{(t)}) = \text{PreExcCost}(|L_{ij}^{(t)}|) + \text{ExcCost}(|L_{ij}^{(t)}|) + \text{PostExcCost}(|L_{ij}^{(t)}|).$$

The cost of a load exchange $L_{ij}^{(t)}$ is the time to prepare this load for the exchange (PreExcCost($|L_{ij}^{(t)}|$)), plus the time of the exchange (ExcCost($|L_{ij}^{(t)}|$)), plus the time to integrate it on the receiver (PostExcCost($|L_{ij}^{(t)}|$)). PreExcCost and PostExcCost completely depend on the application. ExcCost only depends

on the load $L_{ij}^{(t)}$ and on the edge (i,j), a good estimation of this cost can be: $\text{ExcCost}(|L_{ij}^{(t)}|) = \text{Lat}_{ij} + \frac{|L_{ij}^{(t)}|}{\text{Bw}_{ij}}$, where Lat_{ij} and Bw_{ij} are respectively the latency and the bandwidth of edge (i,j). Let us note that the communication can always be hidden some computation.

The benefit given by the exchange of $L_{ij}^{(t)}$ can be estimated by the computation time on i and j without exchange minus the computation time on i and j after this exchange. Intuitively the benefit of a load exchange must be positive if the computation time is reduced by this exchange and negative in the other case. Let us recall that the computation time on i and j is given by the maximum between the computation time on i and the computation time on j. The following equation gives the benefit for the cases - $L_{ij}^{(t)} > 0$ and $L_{ij}^{(t)} < 0$.

$$\text{Benefit}(L_{ij}^{(t)}) = \begin{cases} \max(\text{Cp}(w_i^{(t)}), \text{Cp}(w_j^{(t)})) \\ -\max(\text{Cp}(w_i^{(t)} - L_{ij}^{(t)}), \text{Cp}(w_j^{(t)} + L_{ij}^{(t)})) \text{ if } L_{ij}^{(t)} > 0, \\ \max(\text{Cp}(w_i^{(t)}), \text{Cp}(w_j^{(t)})) \\ -\max(\text{Cp}(w_i^{(t)} + L_{ij}^{(t)}), \text{Cp}(w_j^{(t)} - L_{ij}^{(t)})) \text{ if } L_{ij}^{(t)} < 0, \end{cases} \quad (2)$$

where $\text{Cp}(w_i^{(t)} + ...)$ is the computation time of $(w_i^{(t)} + ...)$ on i.

In the iterative LB algorithms, the benefit of an exchange of load at a given iteration can increase with the next iterations. The estimation of the benefit that we give in Equation (2) is evaluated on only one iteration. Hence, a parameter k is introduced to estimate the benefit on the k successive iterations after an exchange. Indeed, $f_{i,j}^{(t)}$ is equal to 1 if and only if $\text{Cost}(L_{ij}^{(t)}) < k * \text{Benefit}(L_{ij}^{(t)})$. The parameter k can be constant or not (in Section 4 the impact of both cases are compared).

One limit of this cost/benefit system appears when the algorithm converges to a load repartition in step. This problem is studied in Section 3.3.

3.2 Parameter Computation

From the general equation of FOS we have determined the parameter $f_{i,j}^{(t)}$ in the previous section. Now, let us study the parameters $\alpha_{i,j}$ and C_{n_i}. In Section 2 we have seen that only the Boillat Choice does not need a global knowledge to compute $\alpha_{i,j}$, but this method is limited to homogeneous networks.

In this section, a method that only needs a local knowledge is given to determine the relation $\frac{\alpha_{i,j}}{C_{n_i}}$. Let us denote C the vector of the processors speeds. Let C_r be the matrix of relative speeds defined by $C_{r_{i,j}}$, the relative speed of j compared to i:

$$C_{r_{i,j}} = \begin{cases} \frac{C_j}{C_i + C_j} & (i,j) \in E, j \neq i \\ 0 & otherwise. \end{cases}$$

Thus the unit of C is not important, it can be MHz, Mflops or any other. With this definition of a relative speed matrix a diffusion matrix that we denote M_r can be given. M_r is defined such that:

$$M_{r_{ij}} = \begin{cases} \min(\delta_i C_{r_{i,j}}, \delta_j C_{r_{i,j}}) & j \neq i \\ 1 - \sum_{j(j \neq i)} M_{r_{ij}} & j = i \end{cases}$$

with $\delta_i = \frac{1}{\sum_j C_{r_{i,j}}}$. By construction, it is easy to show that $M_{r_{ij}} \geq 0$ and $\sum_j M_{r_{ij}} = 1$, in other words the matrix M_r is stochastic.

Theorem 1. *The diffusion LB algorithm with M_r as diffusion matrix converges toward a load distribution relative to the node speed if and only if M_r is irreducible and aperiodic - the graph G must be connected and non-bipartite.*

Proof. If M_r is stochastic, irreducible and aperiodic, thus the Perron-Frobenius Theorem can be applied, i.e. $\exists \mu$ (μ is a fixed point vector) such that $M_r^T \mu = \mu$. By construction of M_r is stochastique and $M_r^{T^\infty}$ tends to the matrix in which each column is $\frac{1}{\sum_i C_i} C$, thus $\mu = hC$ where h is such that $\sum_i w_i^{(0)} = h \sum_i C_i$. Thus for a given $W^{(0)}$ the invariant distribution μ is proportional to C.

As shown by Theorem 1, the LB algorithm converges if the network G is connected and non-bipartite. The connectivity of the network depends on the set E and the network is not bipartite if M_r is well constructed. The previous method does not ensure that the network is not bipartite, to ensure that we can use the following definition to compute $C_{r_{i,j}}$:

$$C_{r_{i,j}} = \begin{cases} \frac{C_j}{C_i + C_j} & (i,j) \in E, j \neq i \\ \frac{C_i}{2C_i} & j = i \\ 0 & otherwise. \end{cases}$$

To build the diffusion matrix M with one of these two methods and the cost/benefit defined in Section 3.1, the vector $L_r^{(t)}$ of load exchange prediction must be defined to compute $f_{ij}^{(t)}$. $L_r^{(t)}$ is given by $L_{r_{ij}}^{(t)} = M_{r_{ij}} w_i^{(t)} - M_{r_{ji}} w_j^{(t)}$. With $F^{(t)}$ and M_r defined, the diffusion matrix $M^{(t)}$ - as F depends on t, M depends on t - is given by:

$$m_{ij}^{(t)} = \begin{cases} f_{ij}^{(t)} M_{r_{ij}} & j \neq i, \\ 1 - \sum_{k(k \neq i)} f_{ik}^{(t)} M_{r_{ik}} & j = i. \end{cases}$$

3.3 Convergence Detection with Unit Size Tokens

The last step that we study in this paper is the termination of the LB algorithm. This step consists in detecting the end of the LB algorithm to stop it and avoid the cost of exchange of information done by the LB algorithm. This cost can be important if the network is slow and if the number of neighbors is high.

The main problem to detect the convergence is that the load is not infinitely divisible for the real applications. This implies that the LB algorithm cannot always reach a uniform load distribution, hence it does not always reach the convergence point. Some steps of load can appear in the system that can block the LB algorithm.

The unit size tokens problem. Let us start by eliminating this step problem. In the literature, the LB problem of indivisible unit-size tokens is studied in [9] where the authors introduced the "I Owe You" (IOU) unit on each edge, and in [14] where the authors introduced a randomized algorithm that deals with heterogeneous networks. In this section, a new approach based on simulated annealing algorithms is used. The objective is to shake the system to move the load of the most loaded nodes toward the least loaded nodes when the classical LB algorithm is blocked. Hence, the algorithm operates as follows: if a node i is unbalanced with its neighbor j and no load is exchanged between these two nodes, a random value denoted alea is drawn between 0 and 1 ($0 <$alea< 1), and if alea $< e^{(-\kappa * U_{ij})}$, a part of load is exchanged. U_{ij} denotes the number of successive LB iterations during which the neighbors nodes i and j are unbalanced and do not exchange load. The parameter κ defines the probability to exchange load and can be defined by $\kappa = \frac{ln(p)}{\tau}$ where p is the probability to exchange load at the iteration τ of U_{ij}. For example if 50% of probability to exchange is wanted at the second iteration $\kappa = \frac{ln(0.5)}{2}$. Let us note that this method does not ensure to reach the uniform load distribution but it can reduce the unbalance.

Convergence detection problem. Let us recall that the first problem presented in this section is the convergence detection of the LB algorithm. Hence, we must detect that no more load is exchanged in the network. In [15] the authors give a decentralized convergence detection algorithm dedicated to parallel iterative asynchronous algorithms. This algorithm is based on the leader election on the IEEE-1394 (FireWire) protocol, and this base can be used to detect a global state in synchronous algorithms without any centralization. These algorithms operate on a tree, hence a spanning tree of the network must be defined [16,17,18,19].

For the LB algorithm, an adaptation of algorithm given in [15] is used. This adaptation is synchronous and dedicated to binary state detection. The idea of this algorithm is as follows: each node i defines k channels where k is the number of neighbors of i. In the first stage of the algorithm, if a node has only one channel that is not associated to a neighbor, it associates this channel to its neighbor that has no channel and defines this neighbor as its father and sends to its father the state of its sub-tree. If a node receives the state of a sub-tree from a neighbor, it associates a channel to this neighbor and defines this neighbor as one of its children. It is obvious that the leaf nodes of the spanning tree have exactly one such channel at the start of the protocol. Hence, the algorithm is started by a leave that sends its state to its father. In the second and last stage of the algorithm, when a node i has all its channels associated to all its neighbors and that all its neighbors are its children, this node i is the root of the tree. Hence, the state of its sub-tree is the state of the tree, in other words, this node detects the global state of the system. It sends this global state to its children and they do likewise with their children and so on. Thus the information of the global state goes through all the network.

To finish with, if the convergence is detected, the LB algorithm can be stopped if the load is static. In the other case - dynamic load, dynamic networks or other ...

- the convergence detection algorithm can be used to reduce the frequency of LB steps if the system became stable or increase it if it became unbalanced.

4 Simulations

The following simulations are realized with SimGrid [20]. The application that is balanced is represented by an integer that corresponds to the load of the application. Let us recall that the load is static to illustrate the convergence of the algorithm and it is considered homogeneous.

The behavior of the FOS algorithm is studied on the worse configuration - a line topology with all the load on the first node - with 64 homogeneous nodes (2000MFlops). The program that is balanced can be viewed as a parallel and iterative numerical solver that computes 1000 iterations where the topology is virtual and depends on the data dependency - communication for data dependency are simulated. This study is realized for two cases, a first one when the network is a LAN and a second one when the network is a DSL.

4.1 Fast Network

In the former, a bandwidth of 100Mb/s is used with 0.15ms of latency on each edge. Figure 1 shows the gain given by the FOS algorithm with the cost/benefit system and with convergence detection ($Algo2$) compared to the FOS algorithm without cost/benefit system and without convergence detection ($Algo1$). Let us note that in $Algo2$ the cost/benefit parameter k is given by $k^{(t+1)} = k^{(t)} - 1$ with $k^{(0)} = 1000$. The gain is given by $\frac{T1-T2}{T1}$, where $T1$ and $T2$ are the computation time of $Algo1$ and $Algo2$, respectively. In this figure, the gain depends on the load average w^* - the global load is given by $64 \times w^*$ - and on the number of LB steps. The results on Figure 1 show that the gain is significant when w^* is low and also show that the gain is null when w^* is high. This is due to the cost of

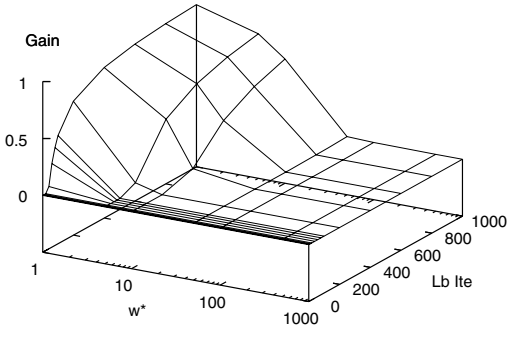

Fig. 1. Gain given with the cost/benefit parameter: $k^{(t+1)} = k^{(t)} - 1$ and with the convergence detection algorithm on a LAN network

the LB algorithm itself: when it has converged, its cost is constant and it only depends of the network. Hence, when the computation time is low - when w^* is low - the cost of load balancing is relatively high and when the computation time is high - when w^* is high - it is negligible. If the cost of load balancing is negligible, the cost/benefit system and the convergence detection are not useful but it can be noted that they are not costly with a LAN network: the gain on Figure 1 is never negative when w^* is high.

4.2 Slow Network

In the latter, the same problem is deployed on a DSL network where the bandwidth is 1Mb/s and the latency is 40ms. Figures 2 and 3 show the program computation times depending on the load average w^* and on the number of LB steps.

Figure 2 corresponds to the program with a classical FOS algorithm without cost/benefit system and without convergence detection algorithm. Here, we can see that the first iterations of the LB algorithm give a gain and that after some iterations of load balancing, the computation time increases and the time to

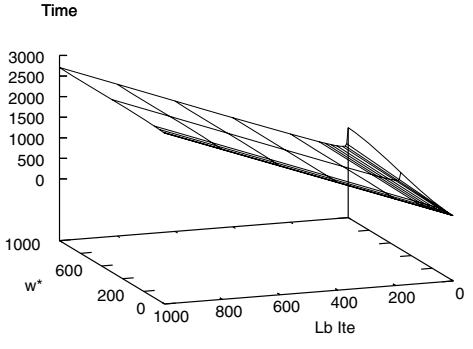

Fig. 2. Classical load balancing

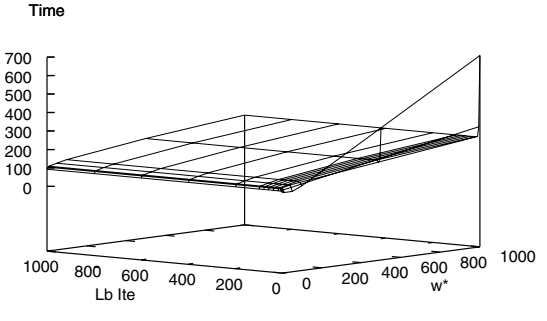

Fig. 3. Load balancing with convergence detection and cost/benefit system with k depends on the computation time of an iteration

compute the program becomes much greater than a sequential computation. This problem has two complementary reasons: the cost of load exchanges and the cost of information exchanges after the convergence of the algorithm. Hence, the cost/benefit system and the convergence detection algorithm can be interesting, in particular also for for small w^*.

An implementation of the convergence detection algorithm and the cost/ benefit system as in the LAN configuration - k defined by $k^{(t+1)} = k^{(t)} - 1$ - showed us that this definition of k is not effective in a DSL network.

Figure 3 shows the results obtained with the convergence detection algorithm and the cost/benefit system with k depending on the computation time of an iteration. For a given node, when the computation time of its iteration is greater than the computation time of its previous iteration, it divides its value of k by 2. Figure 3 shows that with this system, the LB algorithm is stopped after a few iterations in which the computation time has increased. Thus the LB algorithm is beneficial to the program in quasi all configurations. When the global load is small - when a parallel computation is costlier than a sequential one - the LB algorithm is not beneficial but it is stopped fast enough for its cost to be negligible. Moreover, it can be noted that with this extreme configuration the LB algorithm with this cost/benefit system does not use all the processors, see Table 1. The optimal value is the number of processors to reach the minimum

Table 1. This table shows for a given load, in line 2 and in line opt, the number of nodes used and the optimal number of nodes with the cost/benefit system

load nxw^*		64x1	64x5	64x10	64x50	64x100	64x500	64x1000
number of	used	3/64	5/64	6/64	7/64	8/64	9/64	10/64
	opt	1/64	1/64	1/64	1/64	3/64	5/64	10/64

computation time with the cost/benefit system. This optimal value is computed using a global knowledge. We see that without global knowledge, we find a result close to the optimal.

5 Conclusion

In this paper we have studied a practical approach of diffusion load balancing. We have proposed an analysis of the cost and benefit of a load exchange. Based on this analysis we are able to decide wherever or not to exchange the load. This cost and benefit mechanism increases the well-known *step* problem. In order to tackle this problem, we propose a new feature based on simulated annealing that shakes the load when required. Finally, we have enhanced the classical convergence detection to take into account these new elements.

In this work very few assumptions are made. We can deal with static or dynamic load, with any kind of network topology, with heterogeneous nodes and networks and with any type of load. Furthermore, no global knowledge is required to perform the algorithm.

Results show that the proposed features do not degrade the performance of the load balancing algorithm and can lead (in the best case) to 100% of performance increase. Furthermore, in case of slow networks, the algorithm does not use all the available resources in order to give a good speed-up.

References

1. G. Cybenko. Dynamic load balancing for distributed memory multiprocessors. *Journal of Parallel and Distributed Computing*, 7:279–301, 1989.
2. S.H. Hosseini, B. Litow, M. Malkawi, J. McPherson, and K. Vairavan. Analysis of a graph coloring based distributed load balancing algorithm. *Jour. of Para. and Dist. Comp.*, 10:160–166, 1990.
3. B. Litow, S.H. Hosseini, K. Vairavan, and G.S. Wolffe. Performance characteristics of a load balancing algorithm. *Jour. of Para. and Dist. Comp.*, 31:159–165, 1995.
4. R. Diekmann, A. Frommer, and B. Monien. Efficient schemes for nearest neighbor load balancing. *Parallel Computing*, 25:289–313, 1998.
5. J.E. Boillat. Load balancing and poisson equation in a graph. *Concurrency: Practice and Experience.*, 2(4):289–313, 1990.
6. J.M. Bahi and J. Gaber. Load balancing on networks with dynamically changing topology. In *Europar 2001 conference, Lecture Notes on Computer Science*, pages 175–182, Manchester, UK, 2001.
7. D.P. Bertsekas and J.N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods.* nglewood Cliffs NJ, Prentice-Hall, 1989.
8. C.Z. Xu and F.C.M. Lau. Analysis of the generalized dimension exchange method for dynamic load balancing. *Journal of Parallel and Distributed Computing*, 16(4):385–393, 1992.
9. B. Ghosh, S. Muthukrishnan, and M.H. Schultz. First and second order diffusive methods for rapid, coarse, distributed load balancing. In *Proc. of the 8^{th} Annual ACM Sympo. on Para. Algo. and Archi.*, pages 72–81. ACM NY, 1996.
10. J.M. Bahi, R. Couturier, and F. Vernier. Synchronous distributed load balancing on dynamic networks. *Journal of Parallel and Distributed Computing*, 65(11):1397–1405, 2005.
11. R. Elsässer, B. Monien, and S. Schamberger. Load balancing in dynamic networks. In *Proc. 7^th Inter. Sympo. on Para. Archi., Algo. and Net.*, 2004.
12. F. Vernier. *Algorithmique itérative pour l'équilibrage de charge dans les réseaux dynamiques.* PhD thesis, Univ. de Franche-Comté (France), 2004.
13. C.Z. Xu, B. Monien, R. Lüling, and F.C.M. Lau. An analytical comparison of nearest neighbor algorithms for load balancing in parallel computers. In *Proc. of 9th Inter. Para. Proc. Sympo.*, pages 472–479. IEEE CSP, 1995.
14. R. Elssser, B. Monien, and S. Schamberger. Load balancing of indivisible unit size tokens in dynamic and heterogeneous networks. In *Proc. of 12^{th} Annual Euro. Symp. (ESA'04)*, volume 3221, page 640. Springer, 2004.
15. J.M. Bahi, Contassot-Vivier S., R. Couturier, and F. Vernier. A decentralized convergence detection algorithm for asynchronous parallel iterative algorithms. *IEEE Trans. on Para. and Dist. Sys.*, 16(1):4–13, 2005.
16. R.G. Gallager, P.A. Humblet, and P.M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Transactions on Programming Languages and Systems*, 5(1):66–77, 1983.

17. I. Lavallee and G. Roucairol. A fully distributed (minimal) spanning tree algorithm. *Information Processing Letters*, 23(2):55–62, 1986.
18. B. Awerbuch. Optimal distributed algorithms for minimum weight spanning tree, counting, leader election, and related problems. In *The 19th Annual ACM Conf. on Theo. of Comp.*, pages 230–240. ACM NY, 1987.
19. I. Lavallee and C. Lavault. Yet another distributed election and (minimum-weight)spanning tree algorithm. Rr-1024, INRIA - Rocquencourt, 1989.
20. H. Casanova, A. Legrand, and L. Marchal. Scheduling Distributed Applications: the SimGrid Simulation Framework. In *Proc. of the 3rd IEEE Inter. Sympo. on Clust. Comp. and the Grid (CCGrid'03)*. IEEE CSP, may 2003.