A Practical Approach of Diffusion Load Balancing Algorithms

Emmanuel Jeannot, Flavien Vernier

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

A Practical Approach of Diffusion Load Balancing Algorithms.

Emmanuel Jeannot — Flavien Vernier

N° 5875

Mars 2006

Thème NUM

ISSN 0249-6399 ISRN INRIA/RR--5875--FR+ENG

A Practical Approach of Diffusion Load Balancing Algorithms.

Emmanuel Jeannot* , Flavien Vernier †

Thème NUM — Systèmes numériques
Projets AlGorille

Rapport de recherche n° 5875 — Mars 2006 — 20 pages

Abstract: In this paper, a practical approach of diffusion load balancing algorithms and its implementation are studied. Three problems are investigated. The first 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.

Key-words: load balancing, iterative algorithm, diffusion, parameter definition, simulated annealing, convergence detection.

* INRIA, LORIA, campus scientifique, BP 239, 54506 Vandoeuvre les Nancy, France. email: emanuel.jeannot@loria.fr

† UHP-Nancy 1, LORIA, campus scientifique, BP 239, 54506 Vandoeuvre les Nancy, France. email: flavien.vernier@loria.fr

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois, Campus scientifique,

Une approche pratique des algorithmes d'équilibrage de charge de type diffusion.

Résumé : Cet article présente une approche pratique des algorithmes d'équilibrage de charge de type diffusion et de leur implantation. Trois problèmes sont abordés. Le premier concerne la détermination des paramètres de l'algorithme sans aucune connaissance globale du système. Le second s'attache à l'estimation du coût et du bénéfice engendrés par les échanges de charge. Le dernier problème étudié est celui de la détection de convergence de l'algorithme d'équilibrage. Pour ce dernier point, nous proposons un algorithme basé sur le recuit simulé pour réduire une convergence en marches d'escalier qui peut apparaître avec des charges discrètes. Plusieurs simulations terminent cet article et illustrent l'effet des méthodes et algorithmes proposés.

Mots-clés : équilibrage de charge, algorithme itératif, diffusion, définition de paramètre, recuit simulé, détection de convergence.

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 many applications such as dense linear algebra, it is possible to statically compute the load and distribute it evenly among the processors. However, often it is not possible to know *a priori* the load of the problem and therefore a dynamic load balancing scheme is required.

In this paper we focus on the iterative load balancing algorithms introduced in [1]. These 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 global knowledge, like the network topology or node properties, to determine the load balancing parameters. It is suitable to give the optimal parameters if the network and the nodes properties are static and if this knowledge is not too expensive in terms of processing time. However, such global knowledge is not always available and determining the parameters then must be done without this knowledge.

Secondly, most of these algorithms assume that balancing the load is always beneficial and leads to a reduction of the execution time. However, this is not true. Since, transferring some load from one processor to an other requires some communication. If the network is slow the communication cost can be higher than the execution time gain.

Thirdly, since the load is not infinitely divisible, the final load balancing (after convergence of the algorithm) can face a *step* problem. This problem happens when the difference of load between each node is only one. If, for instance, we have a chain of n processors, the load can be distributed as $\{n, n-1, \ldots, 2, 1, \}$ and the diffusion load balancing algorithm can be locked by this configuration.

In this paper, we propose a practical approach of load balancing that solves the 3 above problems. To the best of our knowledge no load balancing algorithm of the literature can deal with these 3 issues at the same 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. Our approach is split into two. The former gives an algorithm based on simulated annealing to reduce or eliminate the steps that can be generated by the discrete load at the convergence of load balancing algorithms. The latter is the detection of the convergence of load balancing algorithms to stop it in static load context, or to manage the frequency of load balancing step in dynamic load context. The objectives are to stop the load balancing algorithm when it has converged and to not lose time to exchange information for the load balancing algorithm. 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 computation speed of the neighbor, to the link characteristics between a node and its neighbor, to the load of neighbors and to the computation time for one unit of load. All these information are supposed to be given, calculated or estimated.

We have implemented and tested the proposed features in order to evaluate their impact. Results show that in the worst case (depending on the network speed and total load), we do not face any degradation of performance compared to previous algorithms of the literature. In the best case the performance gain is more than 100%. An other interesting phenomena we have observed experimentally is that depending on the load, the speed of the network and the computational load, the load balancing 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. This is equivalent to studying the cost and the benefit of the load balancing algorithm. Section 3.2 presents a decentralized method to compute the load balancing parameters. These parameters include the ratio of load exchange and the difference of node speeds. 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 present the implementation of the load balancing algorithm and illustrate the behavior of the load algorithm according to the methods that we give by some experimentation and Section 5 concludes our work.

2 Related Works

The iterative load balancing 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 vertex and E is the set of edge, $E \subseteq V \times V$. 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-weights and let us denote $f_{i,j}$ the weight of edge (i, j) ($f_{i,j} = F_k | E_k = (i, j)$). Let C_n be the vector of node-weights 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. For a process

i, the load balancing step with all its neighbors is given by Equation (1) where $w_i^{(t)}$ is the work load done by process i at time t.

$$w_i^{(t+1)} = w_i^{(t)} + \sum_j \alpha_{ij}.\left(w_j^{(t)} - w_i^{(t)}\right) \quad (1)$$

Equation (1) is linear and thus it can be re-write in matrix form:

$$W^{(t+1)} = MW^{(t)}. \quad (2)$$

Here $W^{(t)}$ is the vector $(w_i^{(t)})$ and M is the diffusion matrix defined by

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

This algorithm has often been studied and derived. In [1, 2, 8] the authors introduced the Dimension Exchange (DE) and the Generalized Dimension Exchange (GDE). In these algorithms a processor balances its load with only one of its neighbors at each step. In [9, 4] the Second Order Scheme (SOS) and the Chebyshev method are introduced. These algorithms remembers the last load exchange to speed up the current exchange. More recently, the adaptation of all these algorithms on dynamic networks have been studied in [10, 11, 12].

In this paper, we focus our study on the determination of the FOS parameters on static and heterogeneous networks. Let us note that our study can be applied to the larger part of the algorithms derived from the FOS on static or dynamic networks. The general form of FOS on static and heterogeneous networks is given by:

$$w_i^{(t+1)} = w_i^{(t)} - \sum_j \alpha_{i,j}.f_{i,j}.\left(\frac{w_i^{(t)}}{C_{n_i}} - \frac{w_j^{(t)}}{C_{n_j}}\right) \quad (3)$$

In this case the diffusion becomes:

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

and the matrix form (2) becomes:

$$W^{(t+1)} = M^T W^{(t)}$$

In the literature, various methods can be found that determine these parameters $\alpha_{i,j}$ 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 needs a global knowledge of the network.

The Cybenko Choice needs a knowledge of the graph degree $D(G)$ and gives: $\alpha_{ij} = \frac{1}{D(G)+1}$, 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 load balancing 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 (3) the parameter $f_{i,j}$ corresponds to the weight of edge (i, j). This parameter favors the fastest edges. In other words, this parameter manages the cost of the LB algorithm. The cost of the LB algorithm is mainly due to the exchange of load. 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 the exchange of $L_{ij}^{(t)}$ 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 and the load $L_{ij}^{(t)}$ is exchanged between i and j. It can be noted that by this definition that $f_{i,j}$ 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,

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

Where the cost of a load exchange $L_{ij}^{(t)}$ is the time to prepare this load for the exchange ($\text{PreExcCost}(|L_{ij}^{(t)}|)$), plus the time of the exchange ($\text{ExcCost}(|L_{ij}^{(t)}|)$), plus the time to integrate it on the receiver ($\text{PostExcCost}(|L_{ij}^{(t)}|)$). 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).

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 $L_{ij}^{(t)} < 0$ - respectively on i and j if i gives load to j.

$$\text{Benefit}(L_{ij}^{(t)}) = \begin{cases} \max(\text{Cmp}(w_i^{(t)}, i), \text{Cmp}(w_j^{(t)}, j)) \\ -\max(\text{Cmp}(w_i^{(t)} - L_{ij}^{(t)}, i), \text{Cmp}(w_j^{(t)} + L_{ij}^{(t)}, j)) & \text{if } L_{ij}^{(t)} > 0, \\ \max(\text{Cmp}(w_i^{(t)}, i), \text{Cmp}(w_j^{(t)}, j)) \\ -\max(\text{Cmp}(w_i^{(t)} + L_{ij}^{(t)}, i), \text{Cmp}(w_j^{(t)} - L_{ij}^{(t)}, j)) & \text{if } L_{ij}^{(t)} < 0, \end{cases}$$

where $\text{Cmp}(w_i^{(t)}, i)$ is the computation time of $w_i^{(t)}$ on i. Let us note that in FOS, if $L_{ij}^{(t)} > 0$ then $\text{Cmp}(w_i^{(t)}, i) > \text{Cmp}(w_j^{(t)}, j)$ and if $L_{ij}^{(t)} < 0$ then $\text{Cmp}(w_i^{(t)}, i) < \text{Cmp}(w_j^{(t)}, j)$, hence the estimation of benefit can be rewritten as follows

$$\text{Benefit}(L_{ij}^{(t)}) = \begin{cases} \text{Cmp}(w_i^{(t)}, i) - \max(\text{Cmp}(w_i^{(t)} - L_{ij}^{(t)}, i), \text{Cmp}(w_j^{(t)} + L_{ij}^{(t)}, j)) & \text{if } L_{ij}^{(t)} > 0, \\ \text{Cmp}(w_j^{(t)}, j) - \max(\text{Cmp}(w_i^{(t)} + L_{ij}^{(t)}, i), \text{Cmp}(w_j^{(t)} - L_{ij}^{(t)}, j)) & \text{if } L_{ij}^{(t)} < 0. \end{cases} \quad (4)$$

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 (4) 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 at the convergence of the algorithm. When the difference of load between 2 homogeneous neighbors is 1 this system stops the exchanges. Let us consider the following example, a line with the load repartition $W^{(t)} = [11, 10, 9, 8, \ldots]$. An exchange that gives the following load repartition $W^{(t)} = [10, 11, 9, 8, \ldots]$ has a cost and a null benefit, therefore the cost/benefit system forbids this exchange. The load repartition in step is a classical problem when the load is not infinitely divisible, it is increased with the cost/benefit system ; 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, various methods

are given to determine $\alpha_{i,j}$ and C_{n_i}. But only the Boillat Choice does not need a global knowledge to compute $\alpha_{i,j}$. The other methods for $\alpha_{i,j}$ and C_{n_i} need a global knowledge.

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 $\delta_i = \frac{1}{\sum_{j(j\neq i)} C_{r_{i,j}}}$. 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, i.e. the graph G must be connected and non-bipartite.*

Proof *If M_r is stochastic, irreducible and aperiodic, 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 it is easy to show that $\mu = hC$ 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:

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

In this case, the relative diffusion matrix Mr is defined by:

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

with $\delta_i = \frac{1}{\sum_j C_{r_{i,j}}}$. This method warrants that the network is not bipartite because it warrants that the diagonal of M_r is not null. Hence, with this method the diffusion LB

algorithm converges if and only if the network is connected (which seems the minimum requirement).

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 $M^{(t)}$ is given by:

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

and the system load behavior is classically defined by:

$$W^{(t+1)} = {M^{(t)}}^T W^{(t)}.$$

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 or the number of neighbors is high. If the load or the network are dynamic, it can be interesting to detect the convergence to reduce the frequency of load balancing steps or to increase it if the unbalance increases.

The main problem to detect 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. Typically, a line of 4 processors with the load distribution $\{3, 2, 1, 0\}$ cannot reach a uniform load distribution with the classical diffusion algorithm. Moreover, this problem appear more often with the cost/benefit system presented in Section 3.1.

3.3.1 The unit size tokens problem

Let us start by eliminating this discretisation problem. In the literature, the load balancing 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 perturbe 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 respect to 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:

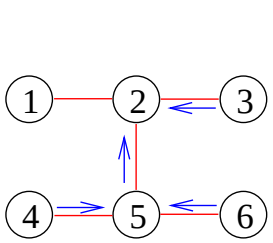
$$\text{alea} < e^{(-\kappa * U_{ij})},$$

a part of load is exchanged. U_{ij} denotes the number of successive load balancing 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.

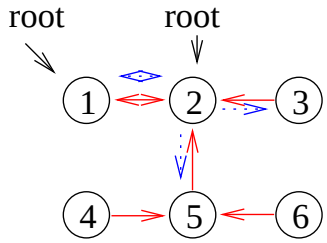
3.3.2 Convergence detection problem

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. Various algorithms exist to generate a spanning tree, from the simple token(s) algorithm to the more sophisticated algorithms that give a minimum-weight spanning tree [16, 17, 18, 19]. The goal of the algorithm is to detect a binary state: the LB algorithm has converged or not.

For the load balancing algorithm, an adaptation of the algorithm given in [15] is used. This adaptation is synchronous and dedicated to binary state detection. The idea of this algorithm is as follows (see Figure 1): 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 the leaves that send their state to their father. Naturally, the state of a sub-tree of node i is defined by $\bigwedge_j S_{t_j}, j \in \{i\} \cup \Delta_i$, where S_{t_j} is the state of j and Δ_i is the set of children of i. Observe that the state of a child j is the state of its sub-tree and the convergence state of a node i is true if it has not exchanged load with its neighbor, and false otherwise. 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 to its children state and they do likewise with their children and so on. Thus the information of the global state goes through all the network. With this algorithm, two root nodes can be defined, that means



(a) A possible state of the first stage of the convergence detection algorithm. The arrows correspond to the subtree state sent from a node to its father.



(b) A possible state of the second stage of the convergence detection algorithm with 2 roots generated by the first stage. The dashed arrows correspond to the global state sent from a node to its children.

Figure 1: These figures illustrate two possible states of the stages of the convergence detection algorithm. They also illustrate the generation of two roots in the tree.

these two nodes have detected the global state but it is not a problem for the convergence detection.

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 Implementation and Simulations

4.1 Implementation

Let us start to introduce a classical structure of an application including an iterative LB algorithm. This structure is given by Algorithm 1.

Two steps appear in this algorithm, the first one is the initialization and the second one is the computation. In the former, the detection of convergence and the load balancing algorithm are initialized. The initialization of the convergence detection - initLBDetecConv - consists in generating a spanning tree of the network used by the application and initializing the parameters and the neighborhood of the convergence detection algorithm. The load balancing initialization - initLB - consists in getting static information - degree of neighbors, power of neighbors and the α_{ij} computed - from the neighbors to initialize the parameters of the LB algorithm. This initialization corresponds to Section 3.2. In the latter - the computation step - if the step is load balancing, the algorithm starts to send its quantity of load and receive the neighbors quantity of load with "getWj". Note that in "getWj" we

Algorithm 1 These algorithms describe the classical structure of prog. with iterative LB algorithm (on the left) and the function that determine the α parameters with a local knowledge used in step initLB (on the right).

```
{classical structure of prog.}
{with iterative LB algorithm}

initApp
initLBDetecConv
initLB
while App condition do
  if is LBStep then
    getWj
    computeLij
    sendRecvLoad
    LBDetecConv
  end if
  //***************
  //* App Job
  //***************
end while
```

```
{distributed computation of α}

for i = 0 to myDegree do
  send my load to neighbor[i];
end for
for i = 0 to myDegree do
  receive load from neighbor[i];
end for
compute Mr_ij Mr_ji
for i = 0 to myDegree do
  send Mr_ij to neighbor[i];
end for
for i = 0 to myDegree do
  receive RMr_ji from neighbor[i];
end for
if RMr_ji < Mr_ji then
  Mr_ij = RMr_ji * Mr_ij/Mr_ji
  Mr_ji = RMr_ji
end if
```

just exchange a number that describes the quantity of load and not a "real" load. Next, it computes the load that it must send to each neighbor according to the conditions given in Sections 3.1 and 3.2. If it appears that two neighbors are not balanced and do not exchange load, the shaker system can force an exchange of load. This step is done in "computeLij". When the load that must be exchanged is computed, each node sends and/or receives the exchanges in "sendRecvLoad". Finally, the convergence detection algorithm checks if the load balancing algorithm has converged to stop it. "initApp", "App condition" and "App Job" correspond to the application that must be balanced.

4.2 Simulation Results

The following simulations are realized with SimGrid. SimGrid is a toolkit that provides core functionality for the simulation of distributed applications in heterogeneous distributed environments [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 - the global load of the system is the same during the computation $\sum_i w_i^{(0)} = \sum_i w_i^{(t)}$ for all t - hence no load can be added or retrieved in the system. A static load is just used to illustrate the convergence of the algorithm, but the LB algorithm can deal with dynamic load. The load is also considered homogeneous, the time to compute a given load does not depend on this load, it only depends on the quantity of load and on the speed of the processor in which it is computed.

In a first part, the behavior of the FOS algorithm is studied on the worse case configuration: a line topology with 64 homogeneous nodes such that all the load is on the first node. 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. This study is realized for two cases, first a LAN network and, second, a DSL network.

4.2.1 Fast network

In the former, a bandwidth of 100Mb/s is used with 0.15ms of latency on each edge. Figure 2 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. Hence, if the gain is negative, the $Algo2$ slows down the execution of the program and a gain of 0.5 means that the computation time of the program is divided by two. 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 load balancing steps. If the load balancing step is 1, the load balancing algorithm is applied only one time at the first iteration of the program. The results on Figure 2 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 the load balancing algorithm itself: when it has converged, its cost is constant

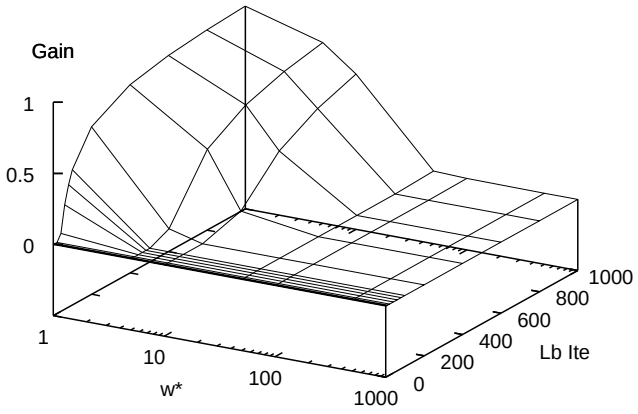


Figure 2: Gain given with the cost/benefit parameter: $k^{(t+1)} = k^{(t)} - 1$ and with the convergence detection algorithm on a LAN network.

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 expensive with a LAN network: the gain on Figure 2 is never negative when w^* is high.

4.2.2 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 3, 4 and 5 show the program computation times depending on the load average w^* and on the number of load balancing steps. Figure 3 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 load balancing algorithm give a gain and that after some iterations of load balancing, the computation time increases and the time to compute the program becomes much greater than a sequential computation. This problem has two complementary reasons. The first is that the classical FOS algorithm does not verify if a load exchange is more costly than beneficial. The second is that in this implementation the FOS algorithm runs until the end of the program, and if the load repartition has converged, the FOS algorithm does not exchange load but it exchanges some information. In a LAN network the cost of these

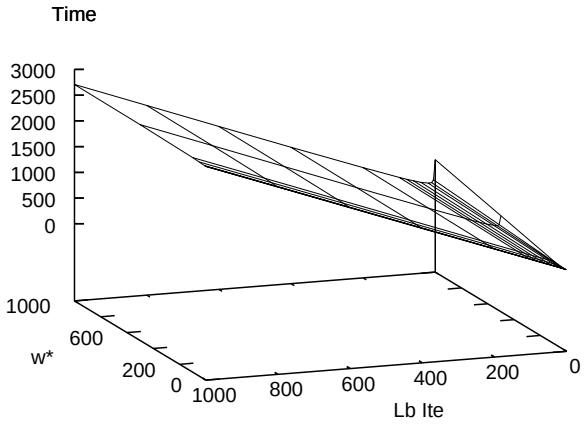


Figure 3: Classical load balancing.

exchanges are negligible but they become very important if the network is slow as for a DSL. Hence, the cost/benefit system and the convergence detection algorithm can be interesting for any size of w^*. In Figure 4 the convergence detection algorithm is implemented and the same cost/benefit system as in the LAN configuration is used: the parameter k is defined by $k^{(t+1)} = k^{(t)} - 1$. We can see that the result is better than in Figure 3. The computation time of the problem does not increase infinitely, the cost/benefit system does not exchange a load if it estimates that the cost is too big and the convergence detection algorithm detects that there is no more exchange and stops the FOS algorithm. But the computation time always increases much more than for a sequential computation. This is due to the estimation done by the cost/benefit system: the cost and the gain of an exchange is computed locally. In a distributed system the gain of an exchange is significant only if it is realized by the slowest processor, it is this processor that dominates the computation time. Hence, a given processor cannot compute an exact gain of an exchange without having a global knowledge, but it can estimate the global impact of the load balancing algorithm with the computation time of an iteration. In Figure 5 the same systems as in Figure 4 are implemented but in this case k also depends on the computation time of an iteration. The computation is synchronous, hence if the computation time of an iteration on the slowest processor increases, the computation time of an iteration on any other processor increases too. Therefore, 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 5 shows that with this system, the load balancing algorithm is stopped after a few iterations in which the computation

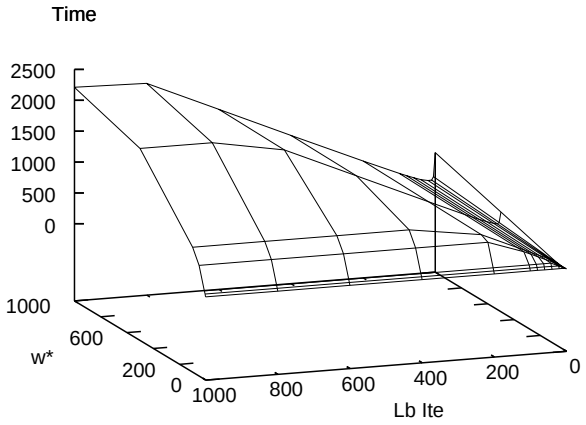


Figure 4: Load balancing with convergence detection and cost/benefit system with $k^{(t+1)} = k^{(t)} - 1$.

time has increased. Thus the load balancing algorithm is beneficial to the program in almost all configurations. When the global load is small - when a parallel computation is more expensive than a sequential one - the load balancing 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 load balancing algorithm with this cost/benefit system does not use all the processors. Table 1 shows the number of processors used and the optimal number of processors. The optimal value - opt1 - is the number of processors to reach the

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	1/64	1/64	1/64	1/64	3/64	5/64	10/64

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

minimum 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 very close to the optimal.

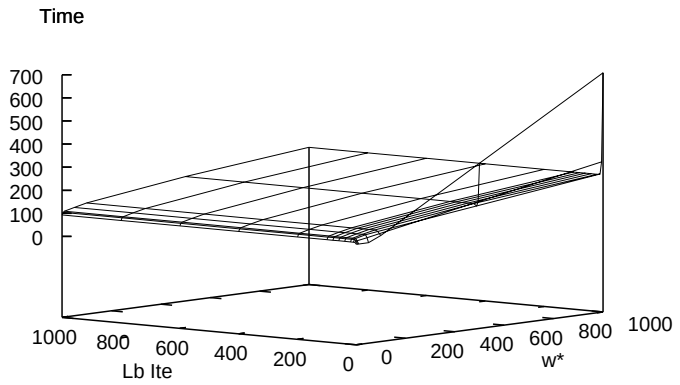


Figure 5: Load balancing with convergence detection and cost/benefit system with k depends on the computation time of an iteration.

4.2.3 Cluster of clusters

In the last experiment, we deploy the diffusion load balancing algorithm on a (10×10) mesh mapped onto two clusters of 50 nodes. In the first cluster the nodes have a power of 3000Mflops and in the second they have a power of 2000Mflops. Both clusters have a network with a bandwidth of 1Gb/s and with a latency of 0.15ms on each edge. The backbone between the clusters has a bandwidth of 1Gb/s and a latency of 15ms. At the initialization, the load is allocated by steps to illustrate the impact of the "shaker", see Section 3.3. This impact is studied with various granularities from 615 units of load that need 100Mflops per load per iteration, to 615.10^6 units of load that need 0.0001Mflops. For the experiment, κ is set to 0.01, a small value to increase the probability to reach a uniform load distribution relative to the power of nodes. The results are given in Figure 6. As the shaker system uses a random values, Figure 6 shows the maximum, the minimum and the average gain given by the experimental results. At first, this figure confirms the gain given by the cost/benefit system with convergence detection. In this configuration the gain is always positive and is between 0.03 and 0.1. Second, it shows that the "shaker" is only beneficial when the granularity is coarse and can be extremely expensive when the granularity is fine. This result is linked to the ratio between the cost of an exchange and the benefit given by this exchange. When the "shaker" moves a load, it does not take cost or benefit into account,

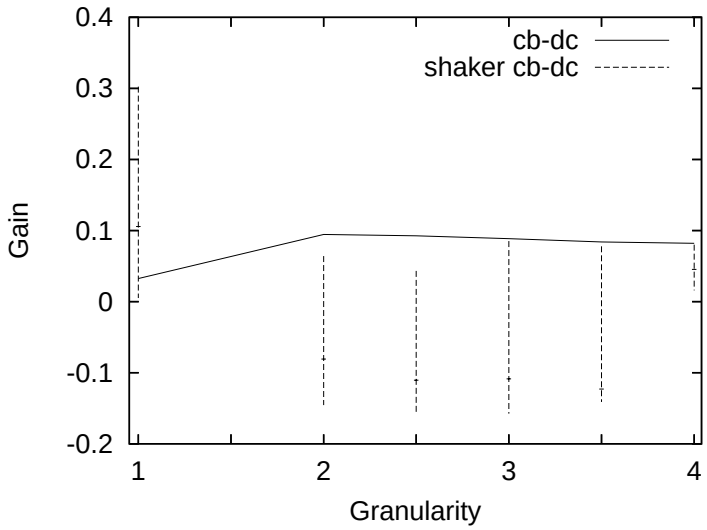


Figure 6: Comparison of gains between FOS with cost/benefit system and convergence detection and FOS with cost/benefit system, convergence detection and shaker system. The algorithm of reference is the classical FOS. Granularity 1 corresponds to 615 units of load that need 100Mflops per load per iteration, 2 to 61500 units of load that need 1Mflops, 3 to 615.10^4 units of load that need 0.01Mflops and 4 to to 615.10^6 units of load that need 1.10^{-4}Mflops

it just tries to unlock the load repartition. Therefore the "shaker" is only beneficial when the cost is negligible in comparison to the benefit.

These experiments show that reaching a uniform load distribution - relative to the nodes power - is not necessarily the objective of a load balancing algorithm. It is very important to manage this to stop the load balancing algorithm before it cost more than it is beneficial.

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 prerequisites are required. 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, 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. 7th 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. Elsässer, B. Monien, and S. Schamberger. Load balancing of indivisible unit size tokens in dynamic and heterogeneous networks. In *Proc. of 12th 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.

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399