Load balancing in dynamic networks by bounded delays asynchronous diffusion

Jacques M. Bahi[1], Sylvain Contassot-Vivier[2,3] and Arnaud Giersch[1]

[1] LIFC, University of Franche-Comté, Belfort, France
jacques.bahi@univ-fcomte.fr, arnaud.giersch@univ-fcomte.fr
Web page: http://info.iut-bm.univ-fcomte.fr/staff/
[2] LORIA, University Henri Poincaré, Nancy, France
sylvain.contassotvivier@loria.fr,
Web page: http://www.loria.fr/~contasss/homeE.html
[3] AlGorille INRIA Team, France

Abstract. Load balancing is a well known problem, which has been extensively addressed in parallel algorithmic. However, there subsist some contexts in which the existing algorithms cannot be used. One of these contexts is the case of dynamic networks where the links between the different elements are intermittent. We propose in this paper an efficient algorithm, based on asynchronous diffusion, to perform load balancing in such a context. A convergence theorem is proposed and proved. Finally, experimental results performed in the SimGrid environment confirm the efficiency of our algorithm.

Keywords: Load balancing, dynamic network, asynchronism, diffusion algorithm

1 Introduction

In the parallel computation domain, the load balancing is often a central issue to reach the optimal theoretical performances. As a consequence, that problem has been extensively studied since the beginning of parallelism in computer science [4, 2, 9, 12]. It can be observed that the evolution of the load balancing algorithms has rather closely followed the one of the parallel architectures. Hence, balancing algorithms have muted from static and centralized distributions [10, 7] to dynamic and/or decentralized ones [11, 1].

Although those mutations allow for load balancing in numerous parallel contexts, there are always emergent architectures which require new breakthroughs in the balancing schemes in order to fully benefit from the endlessly increasing computational power. The parallel systems are more and more complex, often including heterogeneous computational units and interconnection networks. Moreover, the modularity of those systems sharply increases the number of possible parallel contexts. So, it becomes less and less interesting to design balancing algorithms specific to a given context. Thus, there is a strong demand for fully

adaptive algorithms which are also as generic as possible, that is to say, which can be used on any kind of parallel architecture without sensible modifications.

Moreover, in addition to the complex architectures, the emergence of more and more dynamical systems has also been observed during the recent years. The dynamical aspect typically stands at the level of the communications as the links between the computing units are only intermittent.

The most suited strategies to such contexts are the neighborhood strategies based on diffusion algorithms [5, 1, 8]. Unfortunately, most of the current solutions are either synchronous or assume a static network. In the objective to respond to these two current issues, we propose in this paper a load balancing algorithm based on bounded delays asynchronous diffusion.

The following section presents the general computing model used to performed our theoretical study and design of our algorithm. In section 3, a detailed discussion on the balancing ratios to be used in our algorithm is given. Then, the algorithm is provided in section 4 as well as the proof of the balancing convergence in time in section 5.2. Finally, a quality evaluation of our algorithm, performed with the SimGird environment, confirms the very good performances of our approach in section 6.

2 Model

As our balancing algorithm is iterative, its convergence must be proven in order to ensure that the load will be balanced in finite time till there are no modification of the state of the system during the balancing phase. When the system configuration dynamically evolves during the running of the algorithm being balanced, no load stabilization may be observable although our balancing algorithm will follow the evolution of the computational power repartition. Hence, as soon as the system configuration is stabilized, the load repartition will follow with a slight delay. This behavior of our algorithm is proved in a convergence theorem given below. Nevertheless, that theorem and its proof require some notations and a description of the temporal evolution of the system state.

2.1 Notations

For the sake of clarity, we distinguish two kinds of features: those of the platform and the elements related to the application.

Platform characteristics:

$P = \{1, \ldots, n\}$ **:** the set of the n processors in the system.
$G(t) = (P, L(t))$ **:** the undirected connection graph of the links between the n processors at time t.
$N_i(t)$ **:** the set of processors directly connected to i at time t.
$d_j^i(t)$ **:** the delay of j according to i at time t. By definition, it verifies $d_j^i(t) \leq t$.
B **:** the bound of the delays, i.e. $\forall i, j \in P,\ \forall t \in \mathbb{N},\ t - B < d_j^i(t) \leq t$.

Application related values:

$x_i(t)$ **:** the load of processor i at time t.
$x_j^i(t) = x_j(d_j^i(t))$ **:** the load of processor j at time $d_j^i(t)$. That information represents the evaluation at time t on processor i of the load on processor j.
$s_{ij}(t) = \alpha_{ij}(t)(x_i(t) - x_j^i(t))$ **:** the amount of load sent by processor i to processor j at time t. Concerning $\alpha_{ij}(t)$, we have the following constraints: $\forall i, j \in P,\ \alpha_{ij}(t) \in [0,1]$ and $\sum_{j=1}^{n} \alpha_{ij}(t) = 1$. Also, $s_{ij}(t) = 0$ if $j \notin N_i(t)$ or $x_i(t) \leq x_j^i(t)$.
$r_{ij}(t)$ **:** the amount of load received on processor j from processor i at time t.
$v_{ij}(t)$ **:** the amount of load sent by processor i before time t and not yet received by processor j at time t.

2.2 General load balancing scheme

First of all, we consider that we have an initial total load L such that

$$\sum_{i=1}^{n} x_i(0) = L \tag{1}$$

and that there is a conservation of the load in the sense that it is either on the processors or in transit in the interconnection network. In that context, we use the following decentralized scheme to balance the load in the system.

Algorithm 1. *At each time step t, each processor:*

1. *Compares its load to the loads of its connected neighbors*
2. *Determines the load quantities to send to its less loaded neighbors*
3. *Sends those amounts of load to the corresponding nodes*
4. *Potentially receives some load from its more loaded neighbors*

2.3 Dynamical evolution of the system state

In the scope of that study, the main issue addressed is the temporal evolution of the interconnection network of the system. Contrary to classical parallel systems, we consider dynamic links between the different processing units. However, some constraints are necessary to ensure the diffusion of the loads through the system.

Concerning the network, we define the extended neighborhood of a processor i at time t as the set

$$\overline{N}_i(t) = \{j \mid \exists t' : t - B < t' \leq t \text{ such that } j \in N_i(t')\}$$

This means that j has been connected at least one time to i during the time interval $\{t - B + 1, \ldots, t\}$.

Assumption 1. *There exists $B \in \mathbb{N}$ such that $\forall i, j \in P \times P$ and $t \geq 0$, $\max(0, t-B) \leq d_j^i(t) \leq t$ and the union of the communication graphs $\cup_{\tau=t}^{t-B+1} G(\tau)$ is a connected graph.*

This assumption, known as the jointly connected condition [8], implies that information can be exchanged between any couple of nodes i and j within any time interval of length B, and that the delay between two nodes cannot exceed B.

The example given in Fig. 1 shows the effect of that assumption. In the two consecutive times, the connection graphs $G(t)$ are not fully connected. However, their fusion yields a virtual graph which is actually connected.

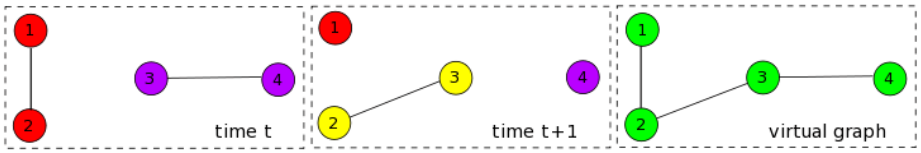

Fig. 1. jointly connected graph

It is interesting to point out that the load balancing in such a context is similar to a kind of information percolation in an intermittent network [6].

Assumption 2. *$\forall t \geq 0$, $\forall i \in P$ and $\forall j \in N_i(t)$, when $x_i(t) > x_j^i(t)$, there exists $\alpha_{ij}(t) > 0$ such that $\alpha_{ij}(t)(x_i(t) - x_j^i(t)) \leq s_{ij}(t)$.*

Assumption 2 indicates that as soon as two nodes are connected, the more loaded sends a non negligible ratio of its load excess to the other node.

Assumption 3.

$$x_i(t) - \sum_{k \in N_i(t)} s_{ik}(t) \geq x_j^i(t) + s_{ij}(t), \quad \forall j \in N_i(t) \ s.t. \ s_{ij}(t) > 0$$

Assumption 3 is essential to avoid the starvation and the ping-pong phenomena. It ensures that the remaining load on the sending processor will not become smaller than the loads on the receptors. A famine occurs when a node has no more workload. The ping-pong state is established when two nodes continually exchange load between each other without reaching an equilibrium.

These last two assumptions are similar to assumption 4.2 in [2].

So, according to these assumptions, we have $\forall i \in P$, $\forall t \in \mathbb{N}$, the following load evolution:

$$x_i(t+1) = x_i(t) - \sum_{j \in N_i(t)} s_{ij}(t) + \sum_{j \in \overline{N}_i(t)} r_{ji}(t) \tag{2}$$

where:

- $s_{ij}(t)$ is given by $\alpha_{ij}(t)(x_i(t) - x_j^i(t))$ with the constraints given above. The values $\alpha_{ij}(t) \in [0, 1]$ define the strategy of the load balancing algorithm. As already mentioned, they must be carefully chosen to ensure the convergence of the algorithm. Their determination is detailed in the following section,

– the last term corresponds to the total amount of load received by processor i from the processors in its extended neighborhood.

Although that equation describes the load evolution on the processors, it does not give any information on the loads in transit. For this, we have:

$$v_{ij}(t) = \sum_{s=0}^{t-1} (s_{ij}(s) - r_{ij}(s))$$

and $v_{ij}(0) = 0$. Moreover, the constraint of load conservation discussed above implies that:

$$\sum_{i=1}^{n} \left(x_i(t) + \sum_{j \in \overline{N}_i(t)} v_{ij}(t) \right) = L, \quad \forall t \geq 0 \tag{3}$$

Once again, it is worth noticing that although the convergence of our balancing is proved in the context of load conservation, our algorithm should provide also interesting results in a more general context of intermittent load evolutions. In fact, inside each time interval where the global load stays constant, our algorithm will tend to balance the current global amount of load among the processors. Thus, an overall gain in performance may be expected in numerous contexts of dynamic loads. However, it is obvious that the detailed behavior and applicability of our algorithm in such cases would require another complete study.

3 Choice of the load ratios

As seen above, the values of the $\alpha_{ij}(t)$ must be chosen such that the amount of load on every node converges to $\frac{L}{n}$.

Let's denote by j^*, the processor satisfying $x^i_{j^*} = \min_{k \in N_i(t)} x^i_k(t)$. It clearly appears that j^* depends on both time and processor i.

In order to correctly choose the $\alpha_{ij}(t)$, Assumptions 2 and 3 are used to deduce the constraints. Assumption 2 can be carried out by fixing an arbitrary constant $\beta \in [0, 1[$ and choosing:

$$\begin{cases} \sum_{k \neq j^* \in N_i(t)} \alpha_{ik}(t)(x_i(t) - x^i_k(t)) \leq \beta(x_i(t) - x^i_{j^*}(t)) \\ \alpha_{ij^*}(t) = \frac{1}{2} \left(1 - \frac{\sum_{k \neq j^*} \alpha_{ik}(t)(x_i(t) - x^i_k(t))}{x_i(t) - x^i_{j^*}(t)} \right) \end{cases} \tag{4}$$

And we deduce

$$\alpha_{ij^*}(t) \geq \frac{1 - \beta}{2} = \alpha$$

Finally, it comes that $\forall i, j^*, t$ such that $j^* \in N_i(t)$ and $x^i_{j^*}(t) = \min_{k \in N_i(t)} x^i_k(t)$,

$$s_{ij^*}(t) = \alpha_{ij^*}(t)(x_i(t) - x^i_{j^*}(t)) \geq \alpha(x_i(t) - x^i_{j^*}(t)).$$

We can observe that the load sent cannot exceed the halve of the load difference between the sender and the receiver.

Furthermore, Assumption 3, avoiding ping-pong effects, implies to choose $\alpha_{ij}(t)$ such that $\forall t \geq 0$, $\forall i \in P$, and $j \neq j^* \in N_i(t)$ satisfying $x_i(t) > x_j^i(t)$,

$$0 \leq \alpha_{ij}(t) \leq \frac{1}{2}\left(1 - \frac{\sum_{k \neq i} \alpha_{ik}(t)(x_i(t) - x_k^i(t))}{x_i(t) - x_j^i(t)}\right) \tag{5}$$

4 Load balancing algorithm

The algorithmic scheme of our load balancing is given below.

Algorithm 2. *At each time step t, each processor:*

1. *Compares its load to the loads of its connected neighbors*
2. *Determines the $\alpha_{ij}(t)$ and deduces the $s_{ij}(t)$*
3. *Sends those amounts of load to the corresponding nodes*
4. *Receives some loads from more loaded nodes*

Although it does not appear directly, the heterogeneity of the processors can be taken into account in this algorithm, for example by introducing virtual processors of the same power (GCD of the actual powers) and distributing them among the actual processors according to their relative speeds. Finally, at each time t and on each node i, the load update is given by (2) and the global behavior of that algorithm is depicted by the following theorem.

Theorem 1. *Under Assumptions 1, 2 and 3, the asynchronous load balancing Algorithm 2 converges to $x^* = \frac{1}{n}\sum_{i=1}^{n} x_i(0)$.*

5 Proof of the load balancing convergence

5.1 Technical results

Let $m(t) = \min_i \min_{t-B<\tau\leq t} x_i(\tau)$. Note that $x_j^i(\tau) \geq m(t)$, $\forall i, j, t$. Lemma 1 and 2 below can be proven similarly to the lemma of pages 521 and 522 in [BT89].

From Assumption 1 we can conclude that the amount of load $v_{ij}(t)$ in the network before time t and not yet received consists in workloads sent in the time interval $\{t - B + 1, ..., t - 1\}$, so $v_{ij}(t) \leq \sum_{\tau=t-B+1}^{t-1} s_{ij}(\tau)$, $\forall i \in P, \forall j \in N_i(t)$.

Lemma 1. *The sequence $m(t)$ is monotone, non-decreasing and converges and $\forall i \in P, \forall s \geq 0$,*

$$x_i(t+s) \geq m(t) + \left(\frac{1}{n}\right)^s (x_i(t) - m(t))$$

Let $i \in P$, $t_0 \in \mathbb{N}$ and $t \geq t_0$, we say that the event $E_j(t)$ occurs if there exists $j \in \overline{N}_i(t)$ such that

$$x_j^i(t) < m(t_0) + \frac{\alpha}{2n^{t-t_0}}(x_i(t_0) - m(t_0)) \quad \text{and} \quad s_{ij}(t) \geq \alpha\left(x_i(t) - x_j^i(t)\right),$$

where α is deduced from (4).

Lemma 2. *Let $t_1 \geq t_0$, if $E_j(t_1)$ occurs, then $E_j(\tau)$ doesn't occur for any $\tau \geq t_1 + 2B$.*

Lemma 3. $\forall i \in P, \forall t_0 \in \mathbb{N}, \forall j \in \overline{N}_i(t)$,

$$t \geq t_0 + 3nB \quad \Rightarrow \quad x_j(t) \geq m(t_0) + \eta \left(\frac{1}{n}\right)^{t-t_0} (x_i(t_0) - m(t_0)).$$

where $\eta = \frac{\alpha}{2}\left(\frac{1}{n}\right)^B$.

Definition 1. *We say that a node j is l-connected to a node i if it is logically connected to i by l communication graphs, i.e. if there exists a minimal sequence (without redundancy) $\{i_0(t_0),\ i_1(t_1),\ \ldots, i_l(t_l)\}$ such that $i = i_0(t_0)$, $i_{j-1} \in N_{i_j}(t_j)\ \forall j \in \{1, ..., l\}$, $i_l = j$ and $t_1 < t_2 < \cdots < t_l$.*

Lemma 4. *If node j is l-connected to node i then*

$$\forall t \geq t_0 + 3nlB,\ x_j(t) \geq m(t_0) + \eta^l \left(\frac{1}{n}\right)^{(t-t_0)^l} (x_i(t_0) - m(t_0)).$$

5.2 Proof of Theorem 1

Consider a node i and a time t_0. Assumption 1 implies that node i is B-connected to any node $j \in P$ and Lemma 4 gives: $\forall t \in [t_0 + 3nMB, t_0 + 3nMB + B]$, $\forall j \in P$,

$$x_j(t_0 + 3nMB + B) \geq m(t_0) + \delta\,(x_i(t_0) - m(t_0)),$$

where $\delta > 0$. This inequality being true for all $i \in P$, it follows that

$$m(t_0 + 3nMB + B) \geq m(t_0) + \delta \left(\max_i x_i(t_0) - m(t_0)\right).$$

We show that

$$\lim_{t_0 \to \infty} \max_i x_i(t_0) - m(t_0) = 0,$$

otherwise we would have $\lim_{t_0\to\infty} m(t_0) = +\infty$. As $\lim_{t\to\infty} m(t) = c$ and $m(t) \leq x_j(t) \leq \max_i x_i(t)$, we deduce that

$$\forall j \in P,\ \lim_{t\to\infty} x_j(t) = c,$$

which implies that

$$\lim_{t\to\infty} s_{ij}(t) = 0.$$

Thanks to Assumption 1, we deduce that

$$\lim_{t\to\infty} v_{ij}(t) = 0,$$

and thanks to (1) and (3), we deduce that

$$nc = \lim_{t\to\infty} x_i(t) = \sum_{i=1}^{n} x_i(0),$$

i.e.

$$c = \sum_{i=1}^{n} x_i(0)/n,$$

which leads to

$$\lim_{t\to\infty} x_i(t) = \frac{1}{n}\sum_{i=1}^{n} x_i(0) = \frac{L}{n},$$

proving Theorem 1.

6 Experimental evaluation

In order to evaluate the efficiency of our load balancing algorithm, we have implemented it in the SimGrid environment [3]. This is a simulation-based complete framework for evaluating cluster, grid and P2P algorithms and heuristics. Among its numerous interests, let's point out realistic computations and communications models. So, the results presented here are fully representative of real results that should be obtained with a similar parallel architecture.

As mentioned in the introduction, our load balancing algorithm is quite generic. However, it should be more interesting in the context of parallel iterative algorithms in which a pool of tasks is repeatedly executed. In that context, we model the iterative process by associating to each task a number of iterations to be performed. Thus, a same task with a constant number of operations is repeatedly executed until its associated number of iterations becomes null. So, as the load balancing takes the form of the migration of those tasks from one node to one of its neighbors, a task may accomplish its iterations on different nodes.

Temporal dependencies between the tasks only occur in synchronous iterative algorithms and when there are some data dependencies between the tasks. Classically, we say that a task A depends on another task B if the computations of A require the knowledge of the data processed in B (typically at the previous iteration). In such a case, when two data dependent tasks are migrated on different nodes, this implies a dependency between those two nodes. However, in the context of use of iterative algorithms, such dependencies very often already exist due to the domain decomposition induced by the parallel treatment of the problem. Indeed, the notion of neighbor between nodes is commonly related to those data dependencies.

In asynchronous iterative algorithms, there are no temporal dependency between the tasks, even if there are some data dependencies. This comes from the fact that each task performs its computations without waiting for the last version of its data dependencies but by using the version of those dependencies which are locally available at that time. In that way, asynchronous algorithms are much more flexible and provide better performances than synchronous ones in numerous parallel contexts.

Due to that last remark and to the fact that our balancing algorithm is also asynchronous by nature, the evaluations presented below take place in the case

of asynchronous iterative algorithms. Moreover, we consider that the domain decomposition is regular and that the tasks all have the same amount of workload.

Before presenting the results, it is necessary to explain how the efficiency of the balancing is evaluated.

6.1 Efficiency evaluation

In the scope of this study, we evaluate the efficiency of the balancing by comparing the performance gains obtained with our algorithm and with a near optimal scheduling. That optimal performance is deduced from the nodes speeds and the tasks workload. As the tasks are composed of a given number of iterations whose workload is always the same, the problem is reduced to the choice of the correct node for each iteration of each task. Thus, the optimal makespan is obtained by successively placing each iteration of each task on the node which is able to offer the soonest ending.

6.2 Experimental contexts

In the following, the efficiency of our algorithm is evaluated for three common topologies: a line, a ring and a complete graph. Although the line is a bit easier to manage from the algorithmic point of view, it is actually the worst case in terms of performances as the load diffusion will be the longest in that case.
The experiments have been conducted in the following conditions:

Cluster	
Size	10 and 50 machines
Powers	homogeneous or heterogeneous (ratio 10 between slowest and fastest)
Links	homogeneous
Initial distribution of the tasks	
or	All on a single node
	evenly distributed over the processors
Communications	
or	Always active
	Intermittent while ensuring Assumption 1
Tasks	
Number	10000
Data size	80 bytes per task
Iterations	random in the range [100,500]
Flops	1600 per iteration

6.3 Results

For the sake of clarity, we present our results in different tables for each topology used. In each cell of the tables, there are two percentages. The first one (on top) gives the relative overhead of our balancing relatively to the theoretically optimal one. That reference time only includes the computation time of all the tasks but

not any scheduling or task migration overhead. As mentioned in Section 6.1, that theoretical makespan is computed by using a best choice list algorithm at the level of the iterations inside the tasks. It is quite obvious that this time is not always attainable in practice but it gives a good reference for the evaluation of our load balancing algorithm. So, 10% indicates that our balancing makes the whole iterative process terminate in a time 10% greater than the optimal one. The second value, in italic, indicates the gain of our balancing relatively to the iterative process without any balancing.

Table 1. Results obtained with a linear topology

	Initial tasks distribution	Homogeneous processors		Heterogeneous processors	
		10	50	10	50
Constant links	All tasks on one node	31.38 *86.86*	387.82 *90.24*	34.96 *92.09*	367.5 *83.9*
	Even distribution	0.44 *0.97*	2.33 *3.13*	16.25 *80.64*	46.26 *75.31*
Intermittent links	All tasks on one node	55.58 *84.44*	967.35 *78.65*	146.73 *85.54*	832.17 *67.89*
	Even distribution	0.48 *0.93*	3.18 *2.33*	52.78 *74.56*	99.89 *66.26*

Table 2. Results obtained with a ring topology

	Initial tasks distribution	Homogeneous processors		Heterogeneous processors	
		10	50	10	50
Constant links	All tasks on one node	11.55 *88.85*	292.48 *92.15*	23.43 *92.76*	370.14 *83.8*
	Even distribution	0.26 *1.15*	2.08 *3.37*	2.78 *82.89*	44.39 *75.63*
Intermittent links	All tasks on one node	23.75 *87.63*	1187.76 *74.24*	127.99 *86.64*	1116.72 *58.09*
	Even distribution	0.54 *0.87*	3.45 *2.07*	34.94 *77.53*	80.62 *69.51*

Table 3. Results obtained with a complete graph topology

	Initial tasks distribution	Homogeneous processors		Heterogeneous processors	
		10	50	10	50
Constant links	All tasks on one node	6.12 *89.39*	811.01 *81.78*	15.24 *93.25*	791.51 *69.29*
	Even distribution	0.4 *1.01*	7.45 *-1.72*	2.8 *82.89*	108.62 *64.79*
Intermittent links	All tasks on one node	28.11 *87.19*	4101.52 *15.97*	46.96 *91.39*	1085.86 *59.15*
	Even distribution	0.31 *1.09*	6.74 *-1.04*	7.93 *82.03*	331.93 *27.09*

Table 1 provides the results for the line topology. That topology is the most difficult case of load balancing as every node has at most two neighbors and this is the communication network with the largest diameter. Hence, that case will yields the longest load diffusions. It can be observed that the results are better for the smaller cluster. This could be expected as a load diffusion will always be longer in a larger system. Moreover, in large systems, each processor has less work to do and for the same initial amount of work, the makespan will be much smaller. This explains the higher ratios according to the optimal makespan. Also, the results are quite different according to the initial load distribution and it is interesting to see that our balancing does not imply any overhead in simple cases like the even distribution on homogeneous nodes. Finally, concerning the intermittent links, our balancing is farther from the optimal time, but this is normal for two reasons. The first one is that the load diffusion is more difficult and naturally longer in such contexts. The second one is that, as mentioned above, the optimal time is computed without taking into account the scheduling and migration costs, which are much more important with intermittent links. Moreover, the absolute performances of our algorithm stay very good in that context as large gains are still obtained relatively to the unbalanced version.

The results for the ring topology are presented in Table 2. As expected for a topology with a smaller diameter, the gains are better than with the linear topology in all the contexts with the small cluster. With the larger cluster, the results are similar or better except for the intermittent links with the initial distribution of the work on only one node. This probably comes from the tuning of the parameters of our load balancing algorithm which have not been optimized. However, the results stay globally satisfying.

In Table 3 are given the results for the complete graph topology. Here again, a good behavior can be observed for the small cluster whereas the algorithm gives rather deceptive results for the larger one. Here again, the local strategy of work distribution seems to play a major role. That strategy gives the rules of how a node distributes its overload to its less loaded nodes while respecting the constraints given in Section 3. So, it defines the β and α_{ij} values. For example, the use of a slightly different strategy taking into account the load average among the node itself and its less loaded neighbors to compute those parameters produces slightly better results in the context of the complete graph. In particular, there are no more loss of time in the already balanced cases as we obtain an overhead of only 4.67% and a gain of 0.92% in the case of an evenly distributed load on homogeneous processors with constant links and an overhead of 3.31% and a gain of 2.21% in the same context with intermittent links.

Finally, all those results are very encouraging but they point out that a single distribution strategy is not adapted to all the contexts of parallel systems. So, a deeper study on the behavior of our algorithm according to its inner parameters will be necessary to precisely analyze the potential causes of inefficient results. Our future investigations on the optimization of the inner parameters of our algorithm (the β value in particular) should provide good results in every contexts of use.

7 Conclusion

An asynchronous decentralized load balancing algorithm has been presented. Its main advantages are to be usable on dynamic networks where the links are intermittent. Moreover, it is quite generic and can be applied to numerous computational algorithms.

The convergence of the balancing has been proved in the context of load conservation. Also, it has been pointed out that in case of variable load, the algorithm will implicitly tend to balance the load during the time intervals in which the load stays constant and will thus globally follow the load variations.

Some simulations have been conducted within the SimGrid environment in the context of an asynchronous parallel iterative algorithm. Globally, the experiments confirm the good behavior of our algorithm, even in the most difficult case of a linear topology. In most of the cases, our algorithm does not induce any sensible overhead in already balanced contexts and provide sharp improvements in the other contexts.

However, as has been pointed out by the simulations, there is still some room for a finer tuning of our algorithm and for the enhancement of its adaptability to large systems. By this way, our next investigation will focus on the automatic determination of the optimal inner parameters of our load balancing scheme in order to be efficient in every context of parallel systems.

References

1. Bahi, J., Couturier, R., Vernier, F.: Accelerated diffusion algorithms on general dynamic networks. In: Fifth International Conference PPAM. pp. 77–82. Poland (2004)
2. Bertsekas, D., Tsitsiklis, J.: Parallel and Distributed Computation. Prentice Hall, Englewood Cliffs, New Jersey (1999)
3. Casanova, H., Legrand, A., Quinson, M.: SimGrid: a Generic Framework for Large-Scale Distributed Experiments. In: 10th IEEE International Conference on Computer Modeling and Simulation (Mar 2008)
4. Cybenko, G.: Dynamic load balancing for distributed memory processors. Journal of Parallel and Distributed Computing 7, 279–301 (1989)
5. Elsässer, R., Monien, B., Preis, R.: Diffusion schemes for load balancing on heterogeneous networks. Theory of Computing Systems 35, 305–320 (2002)
6. Fatès, N.: Directed percolation phenomena in asynchronous elementary cellular automata. In: 7th International Conference on Cellular Automata for Research and Industry (ACRI 2006). pp. 667–675. LNCS 4173, Perpignan, France (2006)
7. Genaud, S., Giersch, A., Vivien, F.: Load-balancing scatter operations for grid computing. Parallel Computing 30(8), 923–946 (Aug 2004)
8. J.M.Bahi, R.Couturier, F.Vernier: Synchronous distributed load balancing on dynamic networks. Journal of Parallel and Distributed Computing 65(11), 1397–1405 (2005)
9. Kumar, V.: Introduction to Parallel Computing. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2002)

10. Miguet, S., Robert, Y.: Elastic load-balancing for image processing algorithms. In: Proceedings of the First International ACPC Conference on Parallel Computation. pp. 438–451. Springer-Verlag, London, UK (1992)
11. Willebeek-Lemair, M.H.: Startegies for dynamic load balancing on highly parallel computers. IEEE Transactions on Parallel and Distributed Systems 4(9), 979–993 (1993)
12. Yawei, L., Zhiling, L.: A survey of load balancing in grid computing. LNCS Computational and Information Science 3314, 280–285 (2005)