Diffusive Algorithms for Dynamic Load Balancing in Massively Parallel Architectures

Antonio Corradi, Letizia Leonardi, Franco Zambonelli

DEIS Technical Report No. DEIS-LIA-96-001, LIA Series No. 8, April 1996

Diffusive Algorithms for Dynamic Load Balancing in Massively Parallel Architectures

Antonio Corradi[1], Letizia Leonardi[2], Franco Zambonelli[1]

1) Dipartimento di Elettronica Informatica e Sistemistica - Università di Bologna
2, Viale Risorgimento
40136 Bologna - ITALY
Ph.: +39-51-6443083
Fax: +39-51-6443073
{acorradi, fzambonelli}@deis.unibo.it

2) Dipartimento di Scienze dell'Ingegneria - Università di Modena
214, Via Campi
41100 Modena - ITALY
Ph.: +39-59-378517
Fax: +39-39-370040
leonardi@dsi.unimo.it

DEIS Technical Report No. DEIS-LIA-96-001, LIA Series No. 8, April 1996

Abstract

The paper investigates the area of dynamic load balancing with the specific target of massively parallel architectures. The lack of centralisation makes the architectures cost effective and scalable but requires suitable simple system policies without centralisation and with decisions based on a limited amount of information. The paper analyses the class of load balancing policies inspired to diffusion and shows how they can lead a system to a load balanced configuration. The paper evaluates and compares the effectiveness of several diffusion-based policies depending both on the external environment (i.e., the properties of the system load) and on the internal parameters. All presented policies show a robust and scalable behaviour: they are able to reach a good load balancing quality with promptness, low intrusion and little dependence on the system size. Moreover, the paper shows that the enlargement of the scope of one diffusive policy can be effective only in case of slow load dynamicity. In any other case, policies with a limited scope are to be preferred.

Keywords

Massively Parallel Architectures, Load Balancing, Locality, Diffusion, Dynamicity

Diffusive Algorithms for Dynamic Load Balancing in Massively Parallel Architectures

Antonio Corradi, Letizia Leonardi, Franco Zambonelli

1. Introduction

Massively parallel architectures make available a large amount of execution and communication resources via replication [AlmG94]. Applications can achieve **high performance** only if their internal components exploits all available resources. The **allocation problem** consists in finding a wise physical resource - logical component assignment to achieve the above goal. Several areas are involved in the allocation problem and in strict relationship: application components not only need execution resources but also communication resources.

When a parallel application presents a static pattern in resource request, a good allocation can be statically determined by taking into account both execution and communication needs of the application components [NorT93]: this approach can be very expensive but tolerable before execution. Unfortunately, several applications present a dynamic behaviour that cannot be a priori identified; in this case, only **dynamic allocation policies** can achieve the goal of exploiting system resources by taking run-time allocation decisions [CasK88]. However, dynamic allocation policies cannot take into account all execution and communication resources: their scope must be limited to a less ambitious goal to reduce their intrusion and to grant prompt decisions.

The paper adopts the dynamic perspective and restricts to the **load balancing** area: it considers as relevant resources only the execution nodes with the goal of balancing their exploitation.

Main objective of an optimal load balancing policy is effectiveness in any situation either depending on the application or independently of it. In other words, several properties can be associated with the above goal:

- **generality**: the independence on the application characteristics;
- **stability**: the capability of not wasting resources in useless actions;
- **low intrusion**: the limited perturbation on the application part;
- **scalability**: the independence on the system size and its configuration.

The fully distributed nature of massively parallel architectures rules out the possibility of a centralised load balancing control. That could become a bottleneck and its performance worsen when the system size increases. Distributed implementations are needed but pay the cost of coordinating distributed decision centres toward a common goal, that can be reached either in a global or a local way. By following the **global approach**, each local decision centre can obtain both a global view of the controlled system and a global coordination in decisions [CorLZ92]. If the system is large, however, the required effort can be intolerable. The **local approach** restricts each decision centre to a local view of the system and limits coordination among them [LulMR93].

The algorithms presented in this paper are based on the idea of load balancing the system only on the basis of space limited load information and by achieving a global goal by composing local actions. A similar principle is applied in the physical word: **diffusion** forces a system toward an homogeneous distribution only on the basis of the local state. Diffusion has inspired the implementation of the algorithms presented in the paper, with different coordination degree among the neighbourhood. The paper evaluates these algorithms and analyses their performance in balancing different load situation with different kind of unbalances: several parameters are introduced toward a metrics of the load.

All the implemented algorithms show themselves well-scalable, achieving good load balancing even for large sized systems. Moreover, all algorithms are robust, i.e. their dependence on the particular kind of unbalance is minimised. However, the paper shows that by enlarging the locality scope of the algorithms best load balancing performance can be obtained, unless the dynamicity of the applications increases. In the latter case, algorithms with a very limited locality scope perform better.

The paper is organised as follows. Section 2 introduces the load balancing problem in its general terms. Section 3 analyses the main issues of load balancing policies. On the basis of this analysis, section 4 introduces and analyses the diffusive approach to load balancing. Section 5 presents different diffusive-based algorithms and their implementation. The effectiveness of these algorithms is finally evaluated in section 6.

2. The Load Balancing Problem

A parallel application defines a set of active components in need of communicating to each other: one application can execute on a parallel architecture only after having allocated its components onto the available physical resources.

The allocation problem consists in finding the optimal logical component - physical resource assignment to achieve the **highest performance** from applications, i.e., the best speed-up compatible with the application properties and the available resources;

Under this general definition, several approaches can be outlined and several sub-problems identified. This section analyses them and comes to a precise definition of the dynamic load balancing problem.

2.1 Static versus Dynamic Allocation

The general problem of allocating N communicating entities to M processors is known to be **NP-hard** [GarJ79]: for large Ms, only sub-optimal solutions can be reached within an acceptable deadline.

If the resource needs of a parallel application are static - i.e., they can be accurately foreseen before the execution - a **static** approach is likely to be followed: a heuristic mapping algorithm decides the allocation of all active components of the application before the execution of the application itself [NorT93, Con95a]. All available data on the application behaviour - both execution and communication needs - can be used by the mapping algorithm to approach the optimal allocation of the application components to the system resources.

However, there are application areas where execution needs cannot be foreseen: for instance, molecular and fluid-dynamics [Cla94, ChuYL95], real-time programs [ShiS96], object-oriented parallel applications [CorLZ96]. In these cases, the static information available to the mapping algorithm, even if it exists, could be unreliable and produce ineffective allocations. Thus, only a dynamic approach should be adopted [CasK88, Con95b].

The **dynamic** approach defers the decisions about the allocation of the active components of an application to the execution time, and these decisions can be based on the current system state and the current application needs of resources. In the dynamic approach, the allocation problem can no longer be solved as an optimisation problem but it must achieve prompt decisions, because of its on-line nature [CanP95]. This is usually granted by restricting the allocation goal to one kind of resources (either execution or communication ones): when no interlaced effects between execution and communication resources are taken into account, to change the allocation of a component produces only linear effect on the global solution. The above change of perspective does not rule out NP-completeness but makes it possible to evaluate allocation decisions at run-time.

The dynamic allocation problem restricted only to execution resources is known as **dynamic load balancing**[1]. The goal of dynamic load balancing is to dynamically assign the active entities of an application to the system nodes to balance the execution load. This can achieved either by dynamically allocating an entity as soon as it appears in the system (**remote creation** [EagLZ86a]) or by moving already allocated entities from node to node during their execution (**migration** [Smi88, DouO91, Nut94]) or by both the above.

Communication issues are delegated to **routing** tools [Boa95].

2.2 Dynamic Load Balancing

Dynamic load balancing can be assimilated to a **system control problem**. The execution of an application must be monitored to detect its evolution and to identify load unbalances; on this base, one can decide and perform the actions needed to lead the system to a balanced configuration [CanP95].

The dynamic change of configuration can be viewed as the effect of a feedback on the application made possible by system observation (figure 1). A **monitoring tool** inspects the application and collects the load information of the current configuration. We can abstract these actions with one functional block:

- *Load Information = Monitor(Current Configuration).*

A **load balancing policy** analyses, on the basis of this information, whether the system is unbalanced and can decide the entities to move around in the system to achieve the load balancing goal. Again, we can express this step in terms of one functional block:

- *Movements = Policy(Load Information).*

By applying **mechanisms** to implement movements such as remote creation or migration, a new configuration of the application is produced. Therefore, by expressing the application of the mechanisms as another functional block:

- *New Configuration = Mechanism(Movements).*

If each block of the load balancing chain defines a precise interface, the design of each block can be clearly separated and one can concentrate only on load balancing policies. On the one hand, the monitor block is not interested in how the load information it collects will be used by the policy; on the other hand, the policy is not concerned neither in the details of information collection nor in the ones of mechanisms' implementation. The following of the paper focuses on load balancing policies, neglecting monitoring and mechanism issues. However, we claim that, even if the policy should not know **how** mechanisms are applied, it is important **which** mechanisms are available to define its decisions.

If migration mechanism is not available, a **creation policy** can only decide about the allocation of entities when they enter the system; once the allocation of an entity has been decided, no further decisions about its allocation are permitted. If **migration** is available, a **migration policy** can handle both the allocation of newly created entities and the movement of already allocated ones.

[1] For sake of readability, we should add that in the literature the term load balancing term often indicates the whole allocation problem [CasK88], though we adopt this restricted definition.

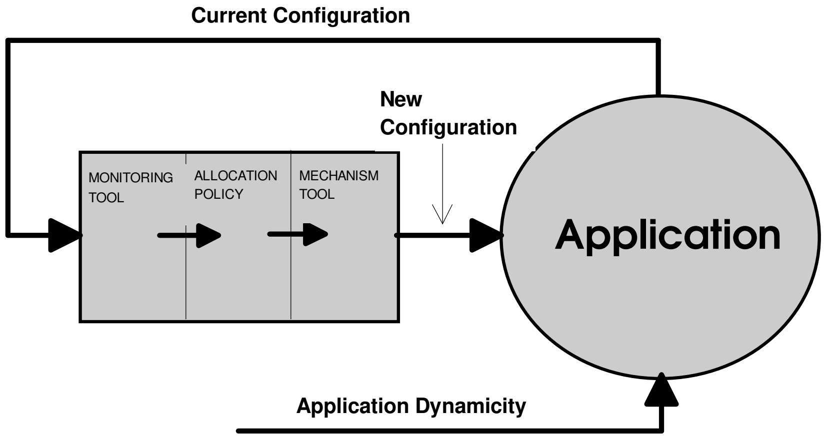


Figure 1. Dynamic Load Balancing as a Feedback on the application

3. Load Balancing Policies

This section analyses the requirements a load balancing policy should answer to be effective and outlines the main characteristics a load balancing policy should show to be effective for massively parallel architectures.

3.1 Requirements of a Load Balancing Policy

A load balancing policy aims to guarantee, as any control tool:

- **stability**: the capacity of moving monotonically toward the load balancing goal, i.e., not to waste resources in actions not effective towards the solution [Sta85];
- **generality**: the capacity of acting independently of the specific properties of the controlled system, i.e., of the behaviour of the applications;
- **low intrusion**: the capacity of acting with no intrusion on the controlled system, i.e., the policy should not subtract too many resources to applications and its action should not produce second-order effects on the system.

In the case of massively parallel architectures, a further requirement is

- **scalability**: the policy must work effectively even on large systems, i.e., it must depend minimally on the system size. In addition, the policy should also be able to act independently of different configurations of the architecture.

It is clear that a policy that answers the first three requirement but not the scalability is not likely to be implemented but in systems of very limited dimensions [CorLZ92].

3.2 Centralised versus Distributed Policies

A load balancing policy is **centralised** if it presents a single decisional component, to evaluate the information collected by the monitoring on each node and to issue the allocation mechanisms in all given nodes. It is **distributed**, instead, if it presents several autonomous decisional components, replicated and distributed over the system nodes: each component is in charge of the allocation decisions about the node it is localised to, eventually by cooperating and coordinating with other decisional components. Even **semi-distributed** load balancing policies could be conceived, by partitioning the decisional components in a multi-level hierarchy [AhmG91, FurTI90].

Centralised policies are intrinsically not scalable: they can be effectively implemented in small sized systems [NiH85, LinR92, GosDI93], but they are likely become a bottleneck as soon as the system size increases up to some dozens of nodes. Scalability, then, suggests distributed implementations.

Even semi-distributed policies can show good performance in very large system but are less general: they generally require to be tuned to the particular system configuration.

3.3 Adaptive versus non Adaptive Policies

As a general rule, any load balancing decision follows the identification of an unbalance and of the involved system nodes. In other words, the policy need to recognise which nodes in the system either can move part of their load or can accept other application load.

When the analysis is based on a predefined loading factor, without any analysis of the current system state, the policy is **non adaptive** [ShiKS92]: a node is considered underloaded, overloaded or neuter on the basis of an a priori known threshold value. For examples, a node can be considered underloaded if it is executing less than TL processes, overloaded if it executing more than TH processes, neuter otherwise [Zhou88, EagLZ86a], where TL and TH are two threshold values statically determined. In **adaptive** policies, the identification is based on loading factors that must take into account the current system state. In particular, the state of a node is dynamically calculated on the basis of the available knowledge about the load of the other nodes of the system[2].

In this paper, we neglect non adaptive policies: they suit better for **load sharing** than for load balancing [EagLZ89]. In other words, non adaptive policies strive not to equalise the load of the system nodes but, instead, to avoid idle processors within the system so to efficiently exploit all available execution resources. Only adaptive policies can achieve load balancing.

3.4 Local versus Global Policies

The amount of load information the decisions take into account and the scope of the actions characterise a load balancing policy.

On the one hand, **global** policies consider information about the load of all nodes of the system and extend their actions over the whole system [HanS91, CorLZ92, BurM93, XuH93]. On the other hand, **local** policies limit the load information to a system size-independent set of nodes. Typically, this set identifies a neighbourhood with respect to the communication network [CorLZ92, Can93, DutM94].

Global policies present several drawbacks, even if they can lead to optimal state identifications:

- to build a global point of view requires high coordination, making the intrusion of the policy unacceptable in large systems [CorLZ92];
- the time needed to obtain load information from all other nodes could be too high: the obtained information can be already obsolete and can lead to unstable decisions.
- the global scope of the actions is likely to produce second-order effects on other component of the system, such as routing [Boa95] or distributed memory management tools [NitL91].

The above reasons make local policies preferred over global ones in large sized systems.

A drawback of the local perspective is that the decision could not reflect the global system state: local policies can only reach a local goal, while one require to achieve the global goal of load balancing. For example, a globally heavily loaded node could be identified as underloaded by a local policy just because its neighbours are more loaded than it. Next section analyses those problems and gives the definition of diffusive algorithms.

---

[2] More detailed taxonomies can be found about the amount of information exploited by load balancing policies, as in [Wu95].

4. Diffusive Load Balancing Strategies

The goal of scalability restricts choices in the design of a load balancing policy for massively parallel architectures. In particular, a policy that effectively works in large parallel system must be:

- **distributed**;
- **adaptive**;
- **locally informed**.

The first property requires that decisional components are replicated in the system; the second requires that decisions must be taken on the basis of the current system state; the third property means that the information available at any decisional component are restricted to a limited scope.

One can doubt this kind of policies could reach the global goal of load balancing because all components act by following a local goal. Apart from a formal proof - to be found in the next sub-section - we suggest a simple analogy: the policies presented in the paper are inspired to the physical phenomenon of **diffusion** to achieve the global result. Diffusion drives any non homogeneous distributions to homogeneous ones, only on the basis of the local state, by moving elementary items on the direction of decreasing energy.

The conditions that make diffusion succeed in balancing a scalar field (for example a temperature or a gas pressure) are:

- the same physic law acting in each point of the field;
- the locality of the goal: the above law aims to find, in each point, the minimal energy level with respect to immediate neighbour values of the field, independently of the global level of the field;
- the locality of the decision environment: in any given point, only the values of the field in that point and in its immediate neighbourhood influence the changes of the scalar field;
- the locality of the effects: any change in the scalar field can immediately influence only the neighbourhood;
- the continuity of the scalar field, defining a seamless space.

The paper deals with load balancing policies strategies that behave in the same way: load balancing is obtained only on the basis of local load information and by locally moving entities along the directions of decreasing load [Cyb89, Xu95]. In particular, a load balancing policy can be defined as diffusive when:

- it is based on **replicated decision** components that behave all in the same way; in other words, their actions are the same for any replicated component, whenever it is allocated in the system;
- the **load balancing goal is locally followed**; replicated components try to balance the part of the system they are allowed to access as if it were the whole system;
- each component bases its decision on **limited load information** coming from a subset of nodes of the system;
- for each replicated decision component, the **scope of its actions is bound** to a local area of the system; in other words, replicated components cannot extend their actions - moving entities - to too far parts of the system;
- the domains of control of each replicated component must provide a **seamless coverage** of the controlled system; in other words, the domain of nodes controlled by any component must have a non null intersection with the domain of nodes controlled by at least one different other component and the whole system must be covered by the union of these

domains. In addition, this coverage must guarantee that it is not possible to split the system into two or more subsets of nodes without splitting at least one domain.

Several different policies can be implemented following the above guidelines; all of them can follow the same formal description for proving their convergence toward a globally balanced system load in a simple and operative way.

4.1 Formal Description

In general, a replicated component of a diffusive algorithm acts within a limited area of control D composed of N_D nodes: we define **domain** this subset of nodes. Any replicated component holds load information about all the nodes of the domain and aims to balance their load with respect to the average domain load. Not only the goal is local, but even actions are limited to this subset of nodes: load can be exchanged by a component only within its domain.

It is possible to show that one diffusive policy achieves the goal of balancing a system in static load situations, i. e., when the load of the system nodes can only change because of load balancing actions. Let *i* be the *i-th* node in a system of *N* nodes. Let its load at time *t* be *Li(t)*, and let it be infinitely divisible, i.e., the granularity of the load be 0. The average system load at time *t* is:

$$\mu(t) = \frac{1}{\mathbf{N}} \sum_{i=1}^{N} L_i(t) \qquad (4.i)$$

Since load balancing policies are load conservative, i.e., they do not neither create nor destroy load but only move it around the system, in static load situation we have that m does not change in time:

$$\mu(t) = \mu(0) = \mu \qquad \forall t$$

While the load of single nodes can change because of load balancing actions.

One can measure the unbalance of the system load at time *t*, with a synthetic indicator, the variance of the load of the nodes, i.e., their quadratic deviation from μ:

$$\sigma^2(t) = \frac{\sum_{i=1}^{N} (L_i(t) - \mu)^2}{N} \qquad (4.ii)$$

Our goal is to show that:

<u>Theorem:</u>

In static load situations, the execution of a diffusive load balancing policy nullifies any load unbalance in a system, i.e.,

$$\forall \varepsilon > 0, \exists T > 0 \Rightarrow \sigma^2(T) = \frac{\sum_{i=1}^{N} (L_i(T) - \mu)^2}{N} < \varepsilon \;\; where \;\; \mu = \frac{1}{\mathbf{N}} \sum_{i=1}^{N} L_i(0) \qquad (4.iii)$$

To prove theorem 1, a lemma is helpful:

<u>Lemma:</u>

If a local load balancing contributes to diminish the local variance of the load in a domain *D* of nodes, by exchanging load only among the nodes of the domain *D*, it also contribute to lower the global variance of the system.

<u>Proof:</u>

For sake of simplicity, let us consider the expression (4.ii) to be multiplied by *N*. The variance at time *t* can then be expresses as:

$$\sigma^2(t) = \sum_{i=1}^{N} (L_i(t) - \mu)^2 = L_1^2(t) + L_2^2(t) + \ldots + L_N^2(t) - 2\mu \sum_{i=1}^{N} L_i(t) + N\mu^2 = \sum_{i=1}^{N} L_i^2(t) - N\mu^2$$

After a load balancing action started at time *t* within a domain *D* of nodes, say at time *t+1*, the variance can be expressed as:

$$\sigma^2(t+1) = L_1^2(t+1) + L_2^2(t+1) + \ldots + L_N^2(t+1) - 2\mu\sum_{i=1}^{N} L_i(t+1) + N\mu^2 = \sum_{i=1}^{N} L_i^2(t+1) - N\mu^2$$

The variation of the variance can be expressed as:

$$\Delta\sigma^2(t+1,t) = \sigma^2(t+1) - \sigma^2(t) = \sum_{i=1}^{N} L_i^2(t+1) - \sum_{i=1}^{N} L_i^2(t) \qquad \text{(4.iv)}$$

Now, let us restrict the attention to one domain *D* and let us assume a balancing action takes place only within it. Because the algorithm is load conservative and because load is exchanged only within the domain *D* (by indicating as $\overline{D}$ the set of nodes not belonging to the domain and by $N_{\overline{D}} = N - N_D$ the number of its elements):

$$\mu_D(t) = \frac{1}{N_D}\sum_{i\in D} L_i(t) = \frac{1}{N_D}\sum_{i\in D} L_i(t+1) = \mu_D(t+1)$$

$$\mu_{\overline{D}}(t) = \frac{1}{N-N_D}\sum_{i\notin D} L_i(t) = \frac{1}{N-N_D}\sum_{i\notin D} L_i(t+1) = \mu_{\overline{D}}(t+1)$$

Consequently, all the terms of (4.iv) are unchanged but the *Li* terms corresponding the nodes of *D*. By adding and subtracting from (4.iv) the terms

$$-N_D\mu_D^2 + N_{\overline{D}}\mu_{\overline{D}}^{\;2},$$

we obtain:

$$\Delta\sigma^2(t+1,t) = -\sum_{i\notin D} L_i^2(t) + N_{\overline{D}}\mu_{\overline{D}}^{\;2} + \sum_{i\notin D} L_i^2(t+1) - N_{\overline{D}}\mu_{\overline{D}}^{\;2} - \sum_{i\in D} L_i^2(t) + N_D\mu_D^2 + \sum_{i\in D} L_i^2(t) - N_D\mu_D^2$$

that allow to express the variation of the variance as two separated contributes:

$$\Delta\sigma^2(t+1,t) = \Delta\sigma_D^2(t+1,t) + \Delta\sigma_{\overline{D}}^2(t+1,t)$$

In other words, the variation of the global variance can be expressed as the variation of the variance in the subsystems identified by D and $\overline{D}$. The load of the nodes of $\overline{D}$ has not changed, i.e.,

$$\Delta\sigma_{\overline{D}}^2(t+1,t) = 0$$

Consequently, if the local action in *D* decreases the local variance

$$\Delta\sigma_D^2(t+1,t) < 0$$

a global benefit stems from the local load balancing action:

$$\sigma^2(t+1) < \sigma^2(t)$$

The result of Lemma 1 assumes that the load balancing action occurs only in one domain at a time; this assumption can be loosen to show that a diffusive policy can make several domains act in parallel and proceed together toward the global goal, as long as they do not interfere with one another.

<u>Corollary:</u>

If load balancing actions take place within domains that do not overlap each other, i.e., domains that do not have any node in common (see figure 1 a, b), then:

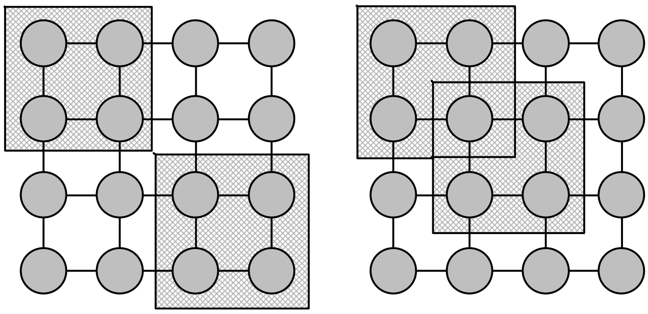
$$\Delta\sigma^2(t+1,t) = \sum_K \Delta\sigma_{D_K}^2(t+1,t)$$

where the D_K's are the not overlapping domain in which a balancing action has take place.

In other words, the variation of the variance of non overlapping domains separately contribute to diminish the global variance.

<u>Proof:</u>

It can be trivially derived from Lemma 1 by separating the contributes to the diminishing of the global variance among the D_K domains.



(a) (b)

Figure 2. Non overlapping (a) versus overlapping (b) domains

The result of the above lemma can be used to prove the theorem.

Let the subdivision of the system in domains guarantees a seamless coverage. In other words:

- each node of the system must belong, at least, to one domain;
- domains must overlap in a way that makes impossible to split the systems in two (or more) subsets of nodes without splitting at least one domain.

Let us suppose that the algorithm applies either to one domain at a time or, by exploiting the result of the corollary to the lemma, to several non overlapping domains at the same time (next section will describe how the implementation can grant the serialisation of the activities of overlapping domains). In addition, let us suppose any replicated component is capable of recognising an unbalance within its domain and of balancing it.

<u>Proof of Theorem 1</u>

Under the above assumptions, the balancing actions in a system stop only when all replicated components have found a balance for their controlled domain. At a given time *T-1*, any residual variance ε still unbalance the system, the seamless coverage assumption grants that there exist at least one domain *D* whose load is not balanced. By exploiting the result of lemma 1, the balancing action that takes place within the domain D provides a reduction of the global variance beyond ε.

We emphasise that, in the above proof, we have made no assumptions about the properties of the nodes that belong to one domain. Then, its result can be applied at any domain scale and at any topological relation within its nodes: the extreme case is the one of a global algorithm to reach load balancing, where a single domain includes the whole system. Obviously, the only interesting case is the one where the domains are composed of nodes that are neighbour in a physical relationship induced by a specific target topology, either regular or irregular. In this case, logical locality of domains reflects physical locality: that produces scalability and makes algorithms capable of issuing in parallel balancing actions for non overlapping domains.

4.2 Concrete Implementation Issues

In section 2.2 we have stated that the design of a dynamic load balancing policy can neglect mechanism issues but should account for their availability. For sake of generality, in the following we assume that migration mechanisms are available.

The key issues in the effective implementation of a diffusive migration policy for a massively parallel architecture can be sketched in phases [EagLZ86a, ShiKS92]:

- **triggering** phase: the policy decides to issue the decision process; in other words, it identifies the events that triggers itself;
- **state identification** phase: the policy decides if it is beneficial to get allocation decisions. More precisely, the policy recognises whether there are load unbalances that can be effectively diminished or no operation is opportune;
- **location** phase: an unbalance needs to be overcome by the identification of both some underloaded nodes (receivers) and some overloaded ones (senders) in the system: entities have to be moved from sender to receiver nodes to come to load balancing;
- **selection** phase: among the application entities, the policy chooses which ones to move from sender to receiver nodes.

Anyone of these phases induces precise design choices, even if some requirements can derive from several reasons; among the others, the overlapping and interfering of the duties of different phases can constrain their implementation. In addition, the implementation must grant that the action among overlapping domains are serialised, i.e., that no actions can take concurrently place in overlapping domains.

4.2.1 Triggering Phase

The actions of a migration policy can be either periodic or event-driven [ShiKS92].

Periodic migration policies evaluate the load situation and exchange load information between nodes at predefined intervals of time. These policies are very simple to implement but have the drawback of requiring a tuning of the time interval: too large intervals could let the load unbalances uncontrolled for too long; too short intervals could make the policy waste execution resources in evaluating load situations substantially unchanged since previous intervals. In general, the selection of the optimal interval is application dependent.

Event-driven migration policies are activated by precise system events - new data collected from the local monitor or new load information coming from other nodes of the system - and exchange load information when the local load situation has changed, making previous data obsolete. Event-driven policies make it possible to react to any unbalance in the system as soon as it appears. In addition, neither application dependent parameters are evaluated nor resources are wasted for unchanged load situation. A drawback of event-driven policies is that they require the presence of a monitor capable of analysing the data it collects and filtering to the policy only significant ones: this avoids the policy to be triggered too often and to cause high overhead. In any case, for large sized systems, letting load information be exchanged asynchronously between node may introduce a too high traffic of data on the communication network.

Diffusive policies can be implemented as event-driven without being affected by the above problems: their overhead is intrinsically limited by the local scope and so it is the amount of data exchanged - locally - between the nodes of the system.

4.2.2 State Identification phase

In the state identification phase, the policy - and the basis of the available load information - recognises the unbalances in the system and assess their magnitude.

As a general rule, it is useful not to start working for any little unbalance: when the load is closed to a balanced situation, a load balancing policy could waste resources only to find no effective actions to be done. This can happens because:

- in real systems, the entities to be allocated are not indefinitely divisible, but have a finite **granularity**. Entities can be too large with respect to the unbalance to permit their movement to reduce it; furthermore, the inability to move but a large amount of load can even make the balancing actions worsen the unbalance, leading to instability.

- the **costs of the allocation mechanisms**, either remote creation or migration ones, are not always negligible. The benefits (in terms of response time or resource exploitation) obtained by a better load balancing could sometimes not outweigh the amount of system resources used to achieve it.

Threshold policies can be introduced to avoid unnecessary decisions: whenever the recognised unbalance is below a given threshold, no balancing actions take place. On the one hand, migration is attempted only when the amount of load to be moved is greater than the average granularity of the entities. On the other hand, a threshold avoid actions that do not produce any real benefit [BarS85]. In addition, since in a real system the load of the nodes is likely to dynamically change in time, the load information available for the allocation decisions can be obsolete. Introducing a threshold diminishes the probability that the inconsistency between the available load information and the effective load leads the policy to an ineffective decision.

The introduction of threshold is necessary in diffusive strategies too: each replicated decisional components must start balancing actions in its domain of interest only if the unbalance in the domain overcomes a given threshold. However, diffusive policies tend to be influenced by those factor more than global ones: the unbalance within a domain can be under the threshold - and no balancing actions will take place - though a significant global unbalance exists in the system. As shown in section 6, diffusive policies are likely to stop their action when a global unbalance cannot be locally detected.

4.2.3 Location Phase

Once a node has identified an unbalance, it can detect itself either as a sender node, i.e., a node likely to get rid of some of its load by migration, as a receiver node, i.e., a node likely to receive entity, or as a neuter node. During the location phase the policy decides **where** one sender node must migrate entities to or, conversely, from where one receiver node should receive them.

Depending on the role of the node in charge of the location phase - looking for a partner node in the system with which to exchange load - the policy is defined as either **sender-initiated** or **receiver-initiated** [EagLZ86a, EagLZ86bb]. If both sender and receiver nodes are authorised to issue the location phase, the policy can be defined as **symmetrically-initiated** [Mir90]. In any case, the location phase has the main goal of establishing sender-receiver couples.

In diffusive policy, the location phase is not clearly separated from the state identification one. In fact, once an unbalance has been identified in the domain, a partner node to establish the couple is searched in the domain itself, whose load situations is already known. Further exchange of information and complex negotiation protocols could overcome the benefits produced by the local state identification phase [LulMR93]. As a consequence, diffusive policies are not strict with regard to the sender/receiver-initiated distinction: once the state of a node is identified and the partner of a migration is automatically identified, it is less important which node initiates the migration process.

4.2.4 Selection Phase

The selection phase decides which entities, among the ones of the sender node(s), have to be migrated onto the receiver one(s).

An important requirement to be evaluated during the selection phase is related to load granularity: since load is not indefinitely divisible, the choice of the entities to migrate must take into account - as far as available information allows - the amount of load each entities carry on, so to move a load amount close to the needed one. In case of too coarse load granularity, the policy must also be able to decide not to move any entity.

Several further requirements can restrict the choice in deciding which entities to migrate [Smi88, ArtF91, JuXY95]. For example:

- entities must be physically executable onto the receiver node, i.e., all needed resources must be available there;
- their expected time to live should be long enough to make their migration worth. To migrate entities that have almost finished their execution is a mistake because migration only delays their termination and does not provides any benefit;
- the link of the selected entities with the sender node must be as loose as possible, i.e., the migration should neither cause a great increase in the entities communication effort nor incur in high cost for deleting all dependencies on the sender nodes.

However, the above requirements tends to be application-dependent: even if some general considerations are possible, a precise discussion must assume a given execution environment and applications. Nevertheless, we emphasise that the locality principle upon which diffusive strategies are based shows its advantages also during the selection phase: when choosing which entities to migrate from a sender to a receiver node, locality can allow the collection of more detailed information about the entities and the characteristics of the nodes.

4.2.5 Serialisation of the Activities

As seen in section 4.1, the convergence of a diffusive algorithm is granted only in the case activities on overlapping domains are serialised. This means that some degree of coordination between the replicated components in charge of realising the policy is needed to avoid situations in which balancing actions take place concurrently in overlapping domains.

From a more concrete point of view, the above conditions can be somehow loosen so to allow a partial degree of concurrency even on overlapping domains: since a load balancing action in a domain does not necessarily involves all the nodes of the domain, all those actions that involves disjoint set of nodes, even if taking place in overlapping domains, need not to be serialised. More in particular, once the decisional activity on a node has been triggered and an unbalance identified, the policy can proceed to complete its balancing action if the nodes of the domain involved in the action are not involved - as part of an overlapping domain - in other balancing actions. As a consequence, the serialisation condition on a domain can be simply realised by granting the serialisation of the load balancing actions within each node of the system.

Whenever the activity of the policy is stopped because competing actions are taking place in overlapping domains, the policy cannot simply wait for their completion: the situation of the nodes of the domain is likely to have been changed, and the information upon which a decision were based could have become obsolete. The load balancing process must abort whenever it is delayed because of competing actions.

5. Implemented Algorithms

Several diffusion-based algorithms are presented in the paper, with the goal of evaluating and comparing possible implementations of diffusion based algorithms. We do not claim the presented algorithms are original: in the literature, many algorithms have been proposed that can be regarded as diffusive ones: even if we have not the ambition to be exhaustive, most algorithms presented in the literature can be easily classified close to the presented schemes.

All the presented algorithms maintain a local perspective and are characterised by a different definition of domain. Distribution can be implemented by one policy handler, replicated in each node of the system, in charge of realising the allocation policy and of coordinating itself with the other policy handlers and of granting the serialisation conditions in each node. We call Allocation Managers (AMs for short) these policy handlers.

We neglect non adaptive policies even if they are based on a strictly local scheme for the location phase [Kum94]: we recall that, by following the definition of section 4.1, an algorithm cannot be considered as diffusive since it implements an a priori threshold-based state identification phase. In addition, we neglect the study of those algorithms in which a diffusion scheme is used to build a global view of the system and to globally coordinate distributed activities. Non adaptive algorithms such as the gradient [LinK87] and the drafting one [NiXG85] fall into this category.

Finally, we neglect the study of random policies too [LulMR93]: some degree of randomness can be useful in peculiar conditions, in particular whenever the dynamicity of the system is so high to make it impossible to consider local load information as a consistent view of the system state. Many algorithms take this choice [Can93], but their analysis goes beyond the scope of this paper.

5.1 The Direct Neighbour (DN) Strategy

The direct neighbour algorithm minimises the domain dimension: each domain consists of only two nodes directly connected by a physical link (figure 3). The presence of nodes interconnected to several other ones, thus belonging to more than one domain, achieves seamless coverage (as defined in section 4).

A balancing action within the domain composed of two neighbour nodes *i* and *j* strives to balance the load of these nodes. Let *Li* be the load of the *i-th* node at time *t* and *Lj* the load of the *j-th* node. At the end of the balancing action, at time *t+1*, the load of the nodes *i* and *j* - in the abstract case of load indefinitely divisible - is:

$$L_i(t+1) = L_j(t+1) = \frac{1}{2}(L_i(t) + L_j(t)) \qquad (5.i)$$

The evolution of the global system load is the consequence of several asynchronous applications of the (5.i) for any *i* and *j*: each local action contributes to diminish the global variance in static load situations. In the ideal case of infinite granularity, after one application of a load balancing action involving the *i-th* and the *j-th*, the global variance diminishes of

$$\Delta\sigma^2(t, t+1) = \frac{1}{2N}(L_i(t) - L_j(t))^2 \qquad (5.ii)$$

because the loads of nodes *i* and *j* reach a perfect balance.

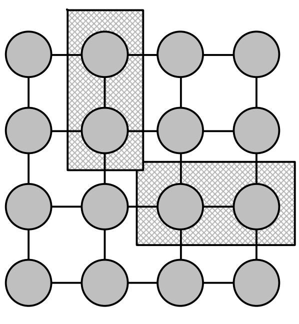


Figure 3. Two domains in the Direct Neighbour policy

The above behaviour of the algorithm does not take into account concrete issues, such as the need of triggering events, the introduction of a threshold, the granularity of the load. Now, let us analyse the DN algorithm behaviour from a more implementation point of view.

The triggering event that makes the load balancing decision activity starts in one node is the arrival of load information - either locally produced by the monitor or coming from one of the neighbour nodes - to the AM. The information exchange occurs any time the monitor detects a significant change of load, as described in 4.2.1.

Whenever the AM node receives updated load information, it compares it with the other available load information. If the local current load exceeds the load of one of its neighbours more than a threshold (i.e., a percentage), the AM starts a migration action toward it. Though the algorithm has been implemented as sender-initiated, a receiver-initiated implementation is also possible and (as asserted in section 4.2.3) would not change the algorithm behaviour.

State identification and location phases overlap. In fact, the state identification compares the load of two nodes and the sender-receiver couple is immediately established. The policy does not include any bidding to determine the new location: a migration can take place whenever two neighbour nodes have great difference in their load.

In the selection phase, the sender of the couple decides the entities to migrate: more than one entity can move in one migration step.

The migration of one or more entities is allowed only if it does not reverse the roles of the involved nodes. In other words, when two neighbour nodes exchange load, the overloaded node cannot become underloaded in its turn and viceversa. This condition guarantees that balancing actions go always in the direction of diminishing the variance of the domain and avoid instability. An immediate instability can stem from the decision of migrating more load from the sender i to the receiver j to make j a potential sender and i its receiver.

The granularity of the load - not indefinitely divisible - makes it sometimes impossible to perfectly equalise the load of a couple of sender and receiver nodes or even to issue any migration between them. That can make the convergence rate toward a global balance poorer than the ideal case: the diminishing of the variance within a domain is usually less than in (5.ii). A similar result derives because of the threshold adoption.

Many diffusive algorithms that can be modelled as a direct neighbour ones have been studied on the past. In [HonTC88] the effectiveness of a direct neighbour strategy is analysed on a hypercube architectures, by considering the impact of the granularity on the achievable load balancing. In [LulMR93] a direct neighbour strategy is presented and compared with many, mainly non adaptive, policies. In [Hos90] the problem of the interference of the activities among overlapping domain in a direct neighbour algorithm is analysed; however, the proposed solution in the field of graph coloring requires global synchronisation of the activites and, then, is not viable for real massively parallel systems.

Several other works discuss and analyse the effectiveness direct-neighbour schemes based on a global synchronisation among the nodes of the system[Cyb89, XuL92, XuL95]: mathematical analysis shows that equalising the load of the sender-receiver couple is not always the optimal solution to grant fast convergence to load balancing; instead, a parameter must be introduced to tune, depending on the topology of the target architecture, the percentage of load to be exchanged among a couple of nodes. Nevertheless, these optimisations produce no effects on asynchronous implementation, as described in [Xu95] and confirmed by our experience.

An interesting extension of the direct neighbour algorithm for taking into account the communication cost is presented in [UehT91]: the advantages of moving one entity to a neighbour node must outweigh the disadvantages derived for the increase in the communication costs.

5.2 The Average Neighbourhood (AN) Strategy

The domain size of diffusive strategies can enlarge still keeping a local perspective: we can identify domains as partitions of the system composed by one node and all its direct neighbours (figure 4). If N is the number of the direct neighbours of a given node, that node belongs at most to $N+1$ different domains: in one domain it is a centre (we call it the *master* of this domain), being in a direct neighbourhood relation with all other nodes of the domain, in the others it is a peripheral node (*slave* nodes of the domain). When the number of domains equals to the total number of system nodes the seamless coverage condition is guaranteed.

By considering a domain D composed of N_D nodes, the goal of the policy, i.e., of the AM on the master node of D, is to balance the load of the nodes of D:

$$L_{i \in D}(t+1) = \frac{1}{N_D} \sum_{i \in D} L_i(t) = \mu_D(t) \qquad (5.iii)$$

The variance of the domain D is:

$$\sigma_D(t) = \sum_{i \in D} \frac{(L_i(t) - \mu_D(t))^2}{N_D} \qquad (5.iv)$$

and it is easy to see that the decrement of the total variance achieved by one balancing action within D is:

$$\Delta\sigma^2(t, t+1) = \sum_{i \in D} \frac{(L_i^2(t) - \mu_D^2(t))}{N} \qquad (5.v)$$

The goal of the AN policy is to balancing the load of all the nodes within a domain. This goal is wider than simply trying to balance the load of the master node of the domain only:

$$L_{master}(t+1) = \frac{1}{N_D} L_{master}(t) + \sum_{j \in D, j \neq master} \frac{L_j(t)}{N_D} \qquad (5.vi)$$

Such a restricted choice can limit the effectiveness of the neighbourhood domain strategy, as shown in [CorLZ92].

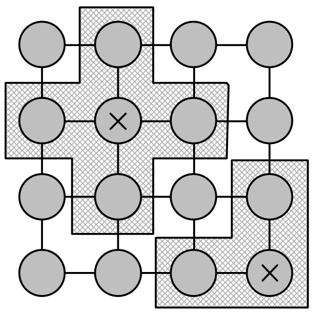


Figure 4. Two domains in the Average Neighbourhood policy (X = master of each domain)

In the concrete case of implementation of the average neighbourhood strategy, the AM in one node is in charge of the balancing decision for the domain it is the master, on the basis of the average domain load. This solution is the most natural, since the master is the only node in the domain able to directly control each node of that domain.

The AM of the master node holds load information about every node of D: it computes the average domain load and two thresholds, T_{sender} and $T_{receiver}$, centred around it. A node is classified as one sender in D if its load exceeds T_{sender}, receiver if its load is under $T_{receiver}$; in any other case it is neuter. Any time a master node receives load information from one of its neighbours (the exchange of load information between nodes is asynchronous, like in the pure local policy, and represents the triggering event), the AM calculates the new average domain load and updates T_{sender} and $T_{receiver}$, to evaluate the needs of load balancing in the domain.

In a single balancing step, depending on the domain load situation, one action can be decided:

1. if the master is sender and one peripheral nodes is a receiver, the master migrates load toward it;
2. if the master is receiver and there is at least one sender in the domain, the master requests load from one of them, in particular the most loaded one;
3. if the master is neuter but there are senders in the domain, it provides to request load from the most loaded sender and to give it to the most underloaded node of the domain.

In other words, the master of a domain tries to push the load of all the nodes in the domain as close as possible to the average domain load, as in (5.iii). However, the concrete algorithm gives priority to the balance of the master node (point 1 and 2). Only if the master is balanced the algorithm tries to balance peripheral nodes (point 3). This choice is an implementation-dependent one: it is not worthwhile to allow the master to move load from many different senders of the domain in one single balancing action because it could become too complex and expensive to complete. The described implementation achieves the goal of moving the load of every domain node within the range defined by T_{sender} and $T_{receiver}$, with more balancing actions but at a lower intrusion cost.

As in the DN policy, a migration must not reverse the role of nodes in the domain to grant stability; in other word, the entity reallocation must be limited to avoid a sender node to become a receiver in the domain and viceversa. Again, granularity and threshold can make it impossible to achieve a perfect balance within the domain and reduces the convergence factor.

Many algorithm based on the AN concept have been presented in literature. However, most of them have the goal of mathematical analysis without any attention to concrete considerations [Cyb89, Boi90, QIaY91, Con93]. Matrix analysis is usually exploited to prove the convergence of diffusive load balancing algorithms and to find an upper bound for the convergence rate. Similarly to the case of the DN policy, the equalisation of the load among the nodes of the domain is proved not to be the optimal scheme for all architecture topologies [XuL94]: tuning the load exchange scheme can improve the convergence rate but, again, the effects of this optimisation are not significant in concrete asynchronous implementations [Xu95].

Moreover, there exist several papers that describe implementations of AN-like load balancing policies [FonMD94, XuH93, DutM94, XuTM95]. However, most of them evaluate the policies on a few concrete applications, and strive not to analyse their general properties.

5.3 Extended Average Neighbourhood Strategies

The AN strategy defines domains that are in a direct neighbourhood relation with a central (master) node. Extensions of this strategy can enlarge the dimension of domains: a domain can be defined as the set of nodes whose distance from a centre node is less than n. When n is 1, the strategy is the above described AN one, when n is 2 the strategy can be defined as average neighbourhood of second level, AN-2 for short, and so on.

Though still based on a diffusion scheme, the implementation of a AN-n strategy with $n>1$ introduces different problems:

- non-direct neighbours of the master cannot directly communicate to it and routing protocols need to be introduced to allow this communication to take place;
- the load information of non-direct neighbour nodes take longer to be available to the master node than the one coming from direct neighbour ones, and it is likely to be obsolete;

- the enlarged dimension of domains makes higher the number of different overlapping domains, and this is likely to limit inter-domain parallelism (we recall that the correctness of a diffusion algorithm is based on avoiding nodes to be concurrently involved in activities belonging to different domains).

In summary, the AN-n strategy introduces higher intrusion on the system than the DN and the AN strategies. Moreover, the enlarged dimension of the domains makes it difficult to achieve the goal of load balancing the whole domain (as expressed in 5.v); instead, the policy succeeds, in most concrete cases, to balance the load of the master node only.

An algorithm based on enlarged system domains is presented in [Can93]. However, the presented algorithm can be no longer considered as a diffusive one: it focuses on the load balancing of the central node of the domain only and it introduces random parameters to deal with the information obsolescence. Similarly, in [DutM94], the enlargement of the domain is applied to the state identification phase only, while load exchange is limited to a domain of direct neighbourhood.

5.4 The Extended Direct Neighbour (EDN) Strategy

In the previous algorithms, entities can migrate between neighbour nodes or within a statically defined domain of nodes. However, when one entity moves, it reaches a different part of the system and enlarges its perspective of the system status. The extended direct neighbour (EDN) algorithm exploits this additional information to locate the destination node of migrating entities.

With reference to the DN policy, once a sender-receiver couple is established, the load to be moved can be allocated only on the receiver node. However, a node less loaded that the original receiver can become available. In the EDN strategy, the load can take advantage of this by moving to more and more underloaded nodes and overcoming the neighbourhood limit. Load reallocation stops when there are no more useful movements, i.e., a node is reached whose load is minimal in its neighbourhood.

The EDN policy is still a diffusive one: it respects all the requirement defined in section 4. The main difference is that the EDN policy dynamically defines the domain decomposition: as soon as entities migrate, the crossed nodes define a domain composed of a chain of neighbour nodes (see figure 5).

From the point of view of the global variance, a single balancing action of the EDN policy always provides a stronger diminishing of the global variance than the DN policy provides. Let us consider the movement as a succession of steps of the direct neighbour kind. The lowering of the global variance is either grater or equal with respect to the direct-neighbour algorithm:

$$\Delta\sigma^2(t,t+1) = \frac{1}{2N}(L_i(t) - L_j(t))^2 + C, \quad C>0 \quad (5.vii)$$

In fact, the first step of the algorithm is the same of the direct-neighbour algorithm, and the same is the contribute of this step to the diminishing of the global variance. The node j that has received load from the node i can forward it to a less loaded neighbour, say h. It is easy to see that also this action contributes to lower the global variance by diminishing the difference of load between the *j-th* and the *h-th* node: the greater the difference of load between the nodes j and h, the greater the decreasing. A more convincing proof is reported in appendix.

A similar extension could be applied to the AN policy (extended average neighbourhood strategy, EAN). However, we have found that the effect of forwarding load to non neighbour nodes overcomes the peculiar advantages of the AN policy. In other words, once the load is allowed to move to farther nodes, the overall performances of the EDN and EAN policies are comparable. Then, in the following, we only detail the EDN policy, by keeping in mind that most considerations and evaluation results apply also to the EAN one.

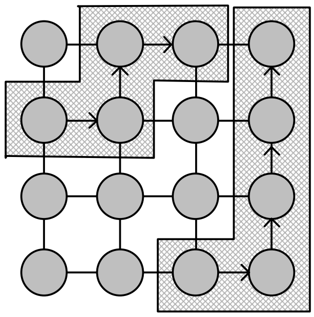


Figure 5. Two domains in the EDN Policy

From an implementation point of view, the load forwarding in the EDN policy is realised by a simple protocol: when a direct-neighbour couple is established and a node *i* is expecting to receive load, it looks in its neighbourhood for one less loaded node, on the basis of its load information. If this node exists, the node *i* acts as a forwarder to it. The same protocol is repeated by any node up to the final receiver, that is a node with a local minimum of load, i.e., that has no neighbours less loaded than it.

A load balancing policy that implements a scheme similar to the above is presented in [Kal88]. It adds two additional parameters within the policy: a maximum number of moves to forward and a minimum distance to move the load from the sender node. The first parameter tries to avoid thrashing, i.e., the situation of moving load indefinitely in the system (such condition could happens in high dynamic conditions, as we will see in the following). The second parameter is introduced to overcome the limits of local forwarding, that can make slow or impossible the reaching of a good load balancing in large system. The possibility of forwarding load to less and less loaded is also considered in [Can93].

An interesting EDN-like algorithm is presented in [Wu95]: the possibility of forwarding migrating load to less loaded nodes is extended with the possibility for underloaded nodes of requesting load in a receiver-initiated way and of letting these requests be forwarded to more loaded nodes.

6. Evaluation

To evaluate the effectiveness of the presented policies, we have used as target a 100 nodes transputer-based architecture (a Meiko CS-1 [Mei89]).

Load balancing policies do not interact directly with applications (as shown in section 2), and the effectiveness of the policies can be evaluated by running them in a simulated execution environment, by simulating the monitor and the migration mechanisms only. Figure 6 shows how the load balancing simplifies, in the target environment, with respect to the one presented in figure 1: the application box has disappeared, the monitor and the mechanism box are replaced by simulators. The monitor simulates the presence of (even dynamic) load on each node and the mechanisms tool simulates the effect of the migration mechanism on the application itself.

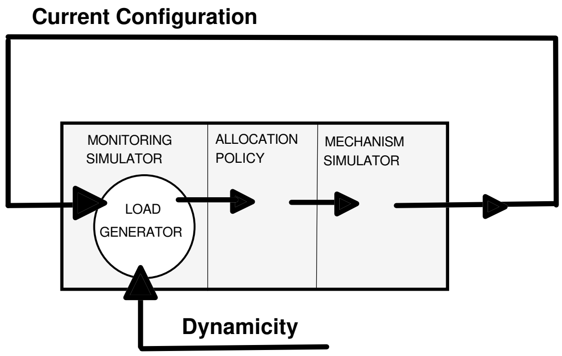


Figure 6. The evaluation environment

The target evaluation environment presents several advantages both over a fully simulated (where the policy is simulated too) and over a fully implemented one (where a real application is running under a load balancing tool). The proposed environment permits to evaluate the behaviour of the policy for real architectures in a more flexible manner than a fully implemented one. It permits:

- to evaluate the behaviour of the policies for different load situations without implementing all possible applications;
- to test the behaviour of applications for a given load balancing policy, just by emulating the application behaviour relevant to load balancing;
- to separately analyse the characteristics of the load balancing policy, the mechanisms and of the monitoring: evaluating the execution behaviour of a real application under a complete load balancing tool can make it difficult to identify where possible inefficiencies stem from, either in the policy, in the monitor or in the mechanisms.

One can criticise this approach because it does not give any measure of the quantitative advantages of one policy over another one in terms of response time. However, the effectiveness of diffusive load balancing policies for dynamic applications in massively parallel systems has been already verified [CorLZ94, LulMR93, DutM94]. On the basis of past results, we lack not another validation but a systematic analysis of the behaviour of different algorithms. This may provide a framework to help application developers in choosing and designing the most appropriate load balancing policy for their application.

6.1 Characterisation of the Load

The implemented policies have been evaluated on different kinds of load situations. The main load factors that can influence the behaviour of the policies are:

- *granularity*, i.e., the coarseness of the entities allocated on the nodes of the systems;
- *unbalancing*, i.e., the deviation of the load situation from the ideal balanced situation;
- *dispersion*, i.e., the presence in the system of a heterogeneous distribution of overloaded and underloaded nodes;
- *dynamicity*, i.e., the frequency of the dynamic changes of the load.

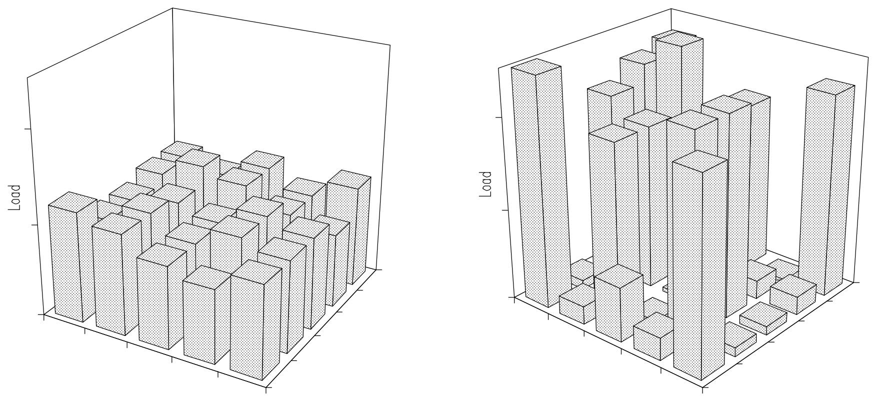
These factors can be quantified with an indicator each.

In our experiments, the load is normalised to a fixed unit, homogeneously defined, that designate one process. In this case, to measure the load granularity is trivial: it can be defined as the average number of entities allocated onto each node. The tests have considered load from very fine grain (up to 100) to very coarse grain (less than 10).

The load unbalance is the global standard deviation of the load of all the system nodes, normalised to the average system load:

$$\sigma = \frac{1}{\mu}\sqrt{\frac{\sum_{i=1}^{N}(Li-\mu)^2}{N}}$$

The experiments have simulated load situations from *lightly unbalanced* ones (σ=10%) to *very unbalanced* ones (σ=100%). Figure 7 reports two examples of different load unbalancing for a 25 nodes mesh topology.



(a) **(b)**

Figure 7. Different unbalancing on a 25 mesh topology
(a: σ=10%; b: σ=100%)

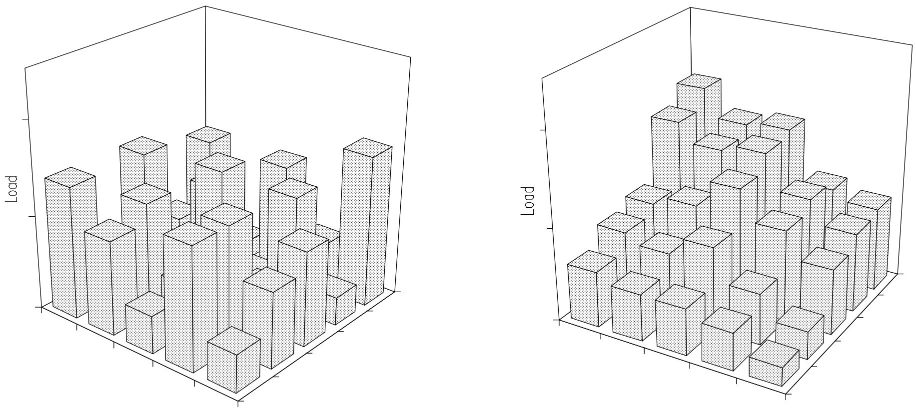
The unbalance in the system can be more or less distributed: overloaded and underloaded nodes can be homogeneously present in every part of the system or they can be concentrated, when the system is divided in regions of overloaded nodes and in regions of underloaded ones. The dispersion (δ) indicator can measure this property: it represents the load standard deviation σ (again normalised to the average system load) as if each node had a load equal to the average of its load and of its neighbour nodes, i.e.:

$$\delta = \frac{1}{\mu}\sqrt{\sum_{i=1}^{N}\frac{(\mu_{Di}-\mu)^2}{N}}$$

where μ_{Di} is the average load of a neighbourhood domain i, composed of N_{Di} nodes:

$$\mu_{Di} = \sum_{h \in Di}\frac{L_h}{N_{Di}}$$

On the one hand, if δ is small w.r.t. σ, each neighbourhood reflects a local unbalance comparable to the global system unbalance: that indicates a homogeneous distribution on the unbalance in the system (figure 8a). On the other hand, if δ is close to σ, the unbalance is concentrated and it is not easily recognisable in its real magnitude with only a local view of the system (figure 8b). Our experiments have simulated value of δ varying from 30% to 80% of σ.



(a) **(b)**

Figure 8. Different load dispersion at fixed standard deviation (σ=50%) on a 25 mesh topology. (a: δ=30% of σ; b: δ=80% of σ)

To characterise the load dynamicity, the previously described load unbalancing indicators (σ and δ) can be extended to take into account the dynamic variations. In particular, the fluctuations of the system load between time *t* and *t+Δt* can be measured by the standard deviation of the load changes in the time span, as if it were the only load to account for. Formally:

$$\Delta\sigma(t, t+\Delta t) = \frac{1}{\mu(t)} \sqrt{\sum_{i=1}^{N} \frac{(\Delta L_i(t, t+\Delta t))^2}{N}}$$

Obviously, the *Δt* interval must be small enough to capture all significant variations of load and to avoid situations where load has been generated and then destroyed in a *Δt*.

A more macroscopic behavioural indicator of the system load is the σ generated by load variations in successive contiguous intervals of time averaged over a given unit period, for example a second:

$$\sigma_{generated}(t, t+N\Delta t) = \sum_{n=1}^{N} \Delta\sigma(t+(n-1)\Delta t, t+n\Delta t)$$

We have simulated a wide range of dynamic situations: from static or quasi static ones, to high dynamic ones (σ /sec=500%).

Similarly, one can further characterise the dynamic changes of load by measuring the dispersion of the generated load.

In the following, we separate the static experiments from the dynamic ones. Static situations more precisely characterise the behaviour of different algorithms; dynamic situations show which strategy better suits highly dynamic applications.

To verify the influence of system size and topology on performance, the algorithms have been tested for different number of nodes (from 4 up to 100 transputers) and for different topologies. For sake of simplicity, in the following we refer to a 100 node (10x10) mesh, apart from when differently specified.

Several runs of each testbed have been performed to obtain averaged and significant data.

6.2 Static Load Situations

All implemented diffusion policies are stable. In each static simulation, they always achieve a stable load situation: no cases signalled the presence of entities blocked in endless migration. This result confirms the model developed in section 4.1.

Another general consideration that overcomes the behavioural differences of the presented algorithms relates to the effects of the threshold and of the granularity.

When the threshold is too high with respect to the load granularity, it prevents any movement and the worsening of the achieved load balancing, i.e., the final σ, is proportional to the threshold value, as in figure 9. Smaller threshold values can improve the balancing quality. The same results apply to granularity: when it is too coarse, it can inhibit any movement; finer granularity approximates the ideal behaviour and policies work considerably better.

Therefore, we claim that the values of granularity and threshold determine the balancing quality that can be obtained. In particular (figure 10), it is useless to choose a threshold too small with respect to granularity: balancing quality stops improving, independently on the threshold, as soon as granularity blocks any load exchange. Conversely, it is not convenient to lower the balancing quality by setting thresholds too high with respect to granularity. The best tuning choice is to make threshold comparable with granularity; in this way, threshold does not limit the balance quality and avoid efforts that cannot achieve any better situation.

In the following (figures 11 to 14) we refer to a value 20 of granularity and a threshold of 20%. However, all the consideration that will be done so far with regard to the relative performance of the policies qualitatively apply to a wider range of these parameters, i.e., to any granularity of load, from very few entities up to hundreds entities for each node of the system, and to any threshold value. Only the absolute values scales as granularity or threshold change.

Let us now analyse the capability of achieving a good load balancing, i.e., a small σ, and the effort needed toward this goal.

While all policies are able to greatly diminish the initial unbalance, they do not achieve the same load balancing quality. The DN policy exhibits the worst behaviour in most load situations: it is not able to reach the same balancing quality of the other policies and, moreover, it employs the higher number of migrations. The AN policy does not significantly improve the quality w.r.t. the DN one but substantially reduces the number of migrations. This makes the AN policy more efficient against the DN one. The AN-2 policy achieves a further improvement, both in terms of final balancing quality and number of migrations, thanks to the enlargement of the dimension of the neighbourhood domains. The EDN policy behaves even better, by obtaining the best balancing quality with the lowest migration effort.

Let us have a more detailed look to the peculiar behaviour of the different policies.

Figure 11 shows that the dependence on the initial σ is low for all policies, i.e., the initial unbalance entity does not influence the final result. As figure 12 shows, the significant parameter is δ. When δ is low, i.e., the unbalance is homogeneously distributed in the system, all policies achieve good results. When δ increases, i.e., it become comparable to σ, the local scope can make it impossible to detect the entity of the global unbalance. In this case, the DN policy often stop its balancing actions with a high residual σ. The reason is the limited scope of the DN policy, whose decisions take into account the load of only two adjacent nodes at a time. The AN policy and, then, the AN-2 one, enlarge their view of the system from domains of two nodes only to larger ones and show themselves less and less sensitive to the dispersion of the unbalance. However, a dependence on the load dispersion is present in the AN policy and, less significantly, even in the AN-2 policy too. The EDN policy shows a very little dependence on δ. Thanks to the capability of moving load farther in the system, the EDN policy mostly overcomes the limits of locality. However, the event that triggers the load to move is the same of the DN policy and this does not allow to completely eliminate the dependence of the EDN policy on δ: high δ can still make it impossible to locally recognise any unbalance and to diminish it. Only global strategies can fully overcome this problem.

With reference to the number of migrations needed to achieve load balancing, figure 13 shows its dependence from the initial σ. Obviously, the more the load is initially unbalanced and the more load has to be moved in order to achieve the balance. However, this growth is higher for the DN policy than for the AN and AN-2 ones: not only a larger scope permits to achieve a better load balancing quality, but it also permits better migration decisions and diminishes their number. The AN-2 policy reaches a better load balancing than the AN one with almost the same number of migrations. The EDN policy achieves the balance with an even more limited number of migration by moving entities, in successive "hops", between non neighbour nodes (a movement composed of several hops is considered as a single migration in our evaluation).

As previously analysed, when δ increased, the capability of balancing of the DN policy substantially diminish. However, as in figure 14, the number of migrations does not proportionally diminish. This is due to the fact that, in case of high δ, local movements of entities can produce only limited benefits towards the global perspective. This situation is less evident in the case of the AN policy and, then, of the AN-2 one; however, the number of migrations does not still diminish proportionally to the achieved balancing. The EDN policy, instead, maintains this proportionality: since an entity hops from node to node in the system before reaching a stable location, movements are no more strictly local and, then, they are more effective even in case of high δ. However, δ affects the number of hops: the more it is high and the more hops entities have to travel on the average.

6.3 Dynamic Load Situations

The analysis of the policies in static load situation evaluates the characteristics of the system load that mainly influences the behaviour of the policies, i.e., their capability of achieving load balancing goal and the needed cost.

When dealing with real applications, however, the load imposed on the nodes of the system is subject to dynamic changes, for several reason. In particular:

- some entities can change their load factor: for examples, one entity can pass through CPU-bound execution phases and I/O-bound ones;
- the number of entities that charge the node of the system can change because of termination or dynamic creation of entities.

Then, it is necessary to evaluate the policies w.r.t. dynamicity.

Figure 15 reports the load balancing quality the policies achieve depending on the dynamicity of the system load, i.e., of $\Delta\sigma$.

From a qualitative point of view, when the load varies slowly, all the policies are able to effectively balance: the quality of the obtained balancing is even improved with respect to the static case. The reason is that a small degree of dynamicity, without mining the effectiveness of the policy, can avoid stock situations where the policies stop their action in a non well-balanced situation (that happens when the situation is locally balanced but not globally balanced). The EDN policy is the one that most takes advantage of this characteristic by improving in a relevant way the load balancing quality obtained with respect to a static load situation.

When the frequency of the load variations becomes too high, all the policies decrease their capacity of balancing. It is interesting to note that, for highly dynamic load situations, the AN-2 and EDN policies behave worst than the AN and DN ones: the DN is the most robust one.

This results can be explained by assimilating the dynamic load balancing tool to a control system. Dynamic load balancing could cause an unstable behaviour in dynamic situations: actions, instead of producing benefits from the balancing point of view, could be either unuseful or, in same cases, dangerous: expensive migration are issued with no progress toward a balanced situation. Figure 16 shows that both the AN-2 and the EDN policies exhibit a high number of migrations even if they could do not balance the load of the system.

Instability can stems from two factors: **slowness** and **over-reactivity**.

If a policy is slow in reacting to an unbalance, the actions are issued when the load information they are based on could be obsolete. Thus, instead of producing a better balance, they could even produce a worse unbalance. The DN policy is the quickest to react, because it it triggered by any new load information and bases its decisions only on this load information. The AN policy too can react as soon as one load information arrives; however, the decision involves the load information about all the neighbourhood, with an increased probability of obsolescence. This is the reason why the DN policy behaves better the the AN one in very dynamic load situations. In the AN-2 policy, whose decisions are based on information coming from non-neighbour nodes, the risk of evaluating obsolete load information is even higher, making this policy inefficient for highly dynamic applications.

With regard to the EDN policy, the load information and the degree of promptness is the same of the DN policy. Then, its limited capability of load balancing highly dynamic load situations is not due to slowness in reacting but, instead, to its over-reactivity: the actions of the EDN policy are exaggerated with respect to the unbalance they aim to correct and are likely to produce an opposite result. In section 5, to avoid this over-reactivity, we have introduced both a threshold in the state identification phase and a constraint on the amount of load to be moved during the selection phase. However, in the EDN policy, the migrating load can move farther to more and more underloaded nodes with no consideration of the threshold, if any less loaded node exists in the neighbourhood. This property of forwarding without any threshold inhibition makes the EDN policy more sensitive to dynamic changes: since the probability of obsolescence of available load information is never null, avoiding the control of any threshold in the forwarding of the load increases the probability of producing dangerous effect instead of benefits (as described in 4.2.2). Typically, the absence of threshold could cause load to be forwarded to more loaded nodes. To avoid this problem, we have experienced the introduction of a further threshold in the EDN policy: this solution increases the robustness of the EDN policy in dynamic situations, but makes it similar to the DN policy in all other cases, obscuring its peculiar advantages.

NOTE AGGIUNTE CHE FORSE NON CI STANNO A DIRE NIENTE

A more quantitative analysis to characterise in absolute terms the robustness of the policy to dynamic load situation cannot be done without precise assumptions on the whole execution environment: in a real system, the instability of the policies in highly dynamic load situations is caused by the delay introduced by the whole load balancing chain, including the monitoring and the mechanisms, and not by the delay introduced by the policy only:

$T_D = T_{Monitor} + T_{Policy} + T_{Mechanism}$

Nevertheless, our experiments permit some quantitative considerations.

By recalling the definition of load variation given in 6.1, and by recalling that there exist maximum time Δt that allows to capture all of the variation of the load in any interval $(t, t+\Delta t)$, we have found that since the Δt needed to captures all load variation is larger than the average delay introduced by the whole load balancing tool, i.e.,

$\Delta t >> T_D$

all the policies behave well and do not suffer from the presence of dynamic load.

As soon as Δt and T_D become of the same magnitude, i.e.,

$\Delta t < 10T_D$

and depending on the value of δ, the AN-2 and the EDN policies begins to degrade their performance.

When the absolute values of Δt and T_D are comparable, even the DN and the AN policy begins to degrade their performance.

This results accord with the ones reported in [CanP95].

6.4 Scalability of the Policies

To evaluate the scalability of the policies, i.e. their ability of working well independently on the system size, we have run them in systems of different size, in particular mesh shaped configurations with a number of nodes varying from 16 to 100. Two important characteristics have to be evaluated, in function of the system size, to grant the practical scalability of the policies: their overhead and their capability of achieving good load balancing.

All the policies grants low intrusion, whichever the system size: as in figure 17, referring to the execution of the policies in a dynamic load situation with $\Delta\sigma/s=100\%$, the introduced overhead is under 3% for all the policies and it is show only a light dependence on the system size.

With regard to the obtained load balancing quality, it is not considerably different for the four policies since the system size is small (figure 18). While the size grows, however, the balancing quality worsens: the more the scope of the policies is restricted the more the worse is. The EDN policy exhibits a lower degradation and seems to be less influenced by the growth of the system size. This result can be explained by the enlarged scope of control. The DN, AN and AN-2 policies maintain a constant local scope, whichever the system size. When the system size increases, the relative area they control becomes smaller compared with it. The EDN policy, instead, can extend its actions to larger areas of the system, making its scope grow on need by forwarding load farther.

To infer the behaviour of the policies in larger systems, we have studied their behaviour in configuration with a larger diameter (the maximum distance between any two nodes in the system), i.e., a ring one. These tests confirmed the above considerations about policies scalability, by making more evident the effectiveness of the EDN policy over the others for large sized systems.

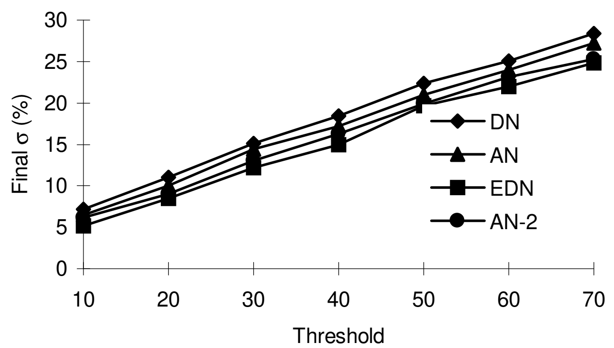


Figure 9. Influence of Threshold on quality precision
(σ = 40-60%, δ = 40-60%, Granularity = 100)

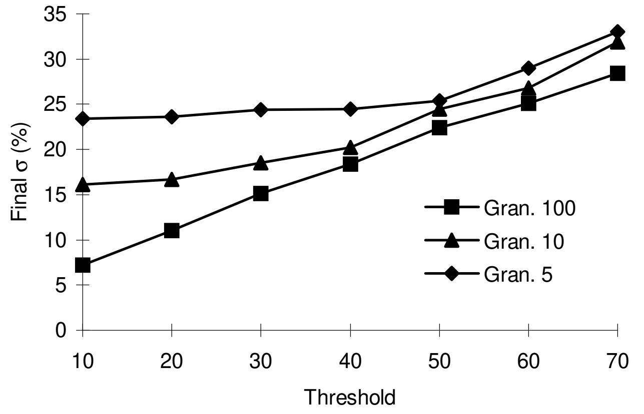


Figure 10. Influence of Threshold on quality precision for different load granularity (Policy = DN, σ = 40-60%, δ = 40-60%)

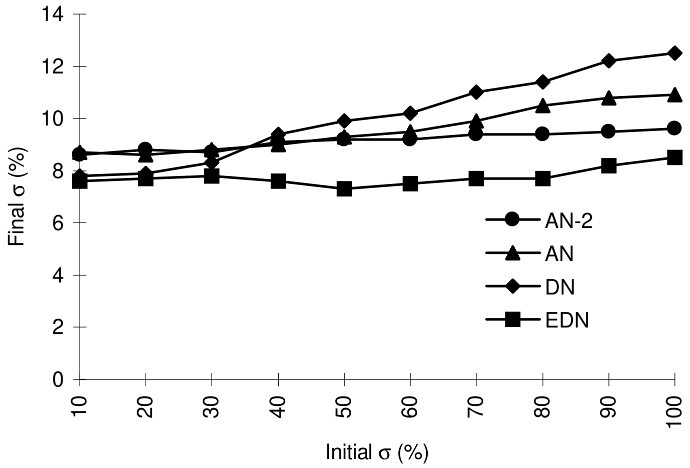


Figure 11. Influence of initial unbalancing on quality precision (δ = 50%, Granularity = 20, Threshold = 20%)

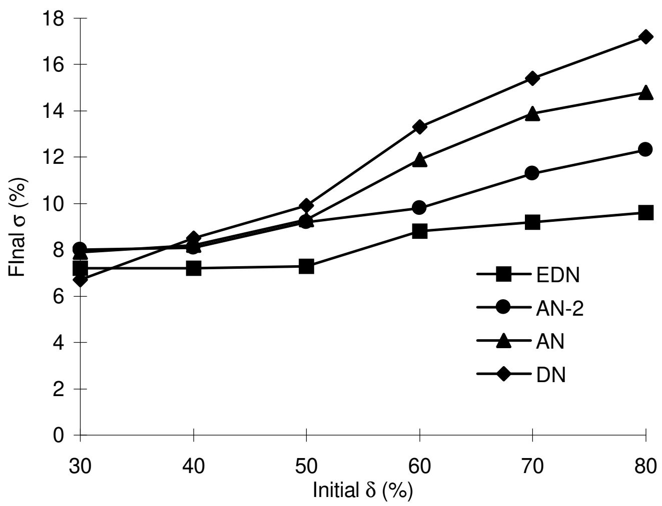


Figure 12. Influence of load dispersion on quality precision (σ = 50%, Granularity = 20, Threshold = 20%)

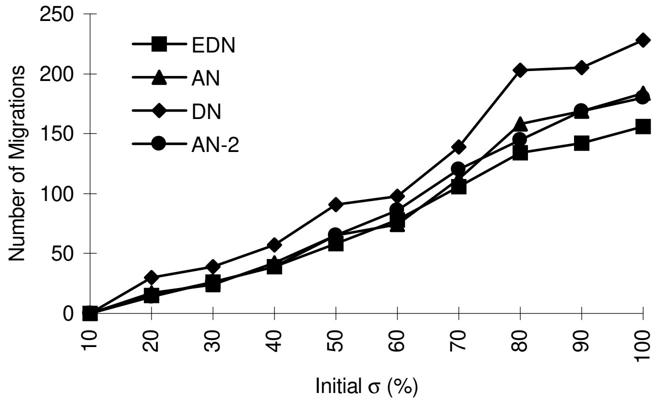


Figure 13. Influence of initial unbalancing on migration effort (δ = 50%, Granularity = 20, Threshold = 20%)

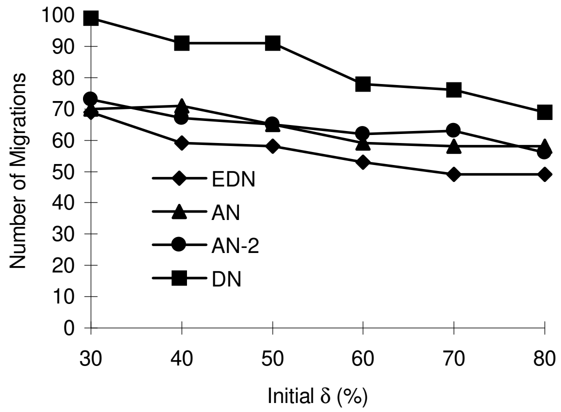


Figure 14. Influence of dispersion on migration effort (σ = 50%, Granularity = 20, Threshold = 20%)

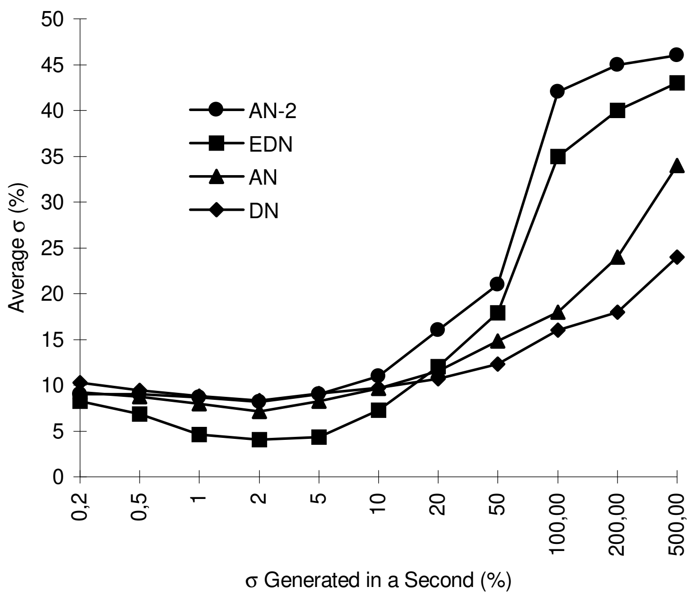


Figure 15. Influence of Dynamicity on quality precision (δ = 40%:60%, Granularity = 20:30, Th. = 20%)

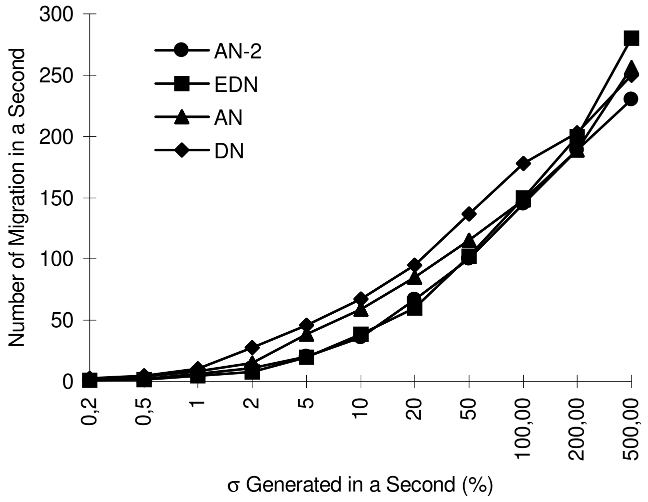


Figure 16. Influence of Dynamicity on migration effort
(δ = 40%:60%, Granularity = 20:30, Threshold = 20%)

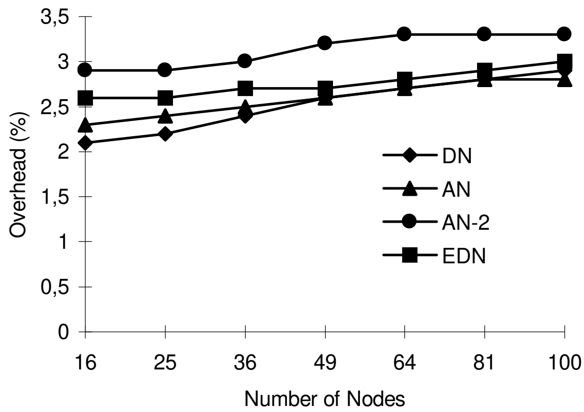


Figure 17. Overhead of the policies in function of the system size
(Δσ = 100%, δ = 40%:60%, Granularity = 20:30, Threshold = 20%)

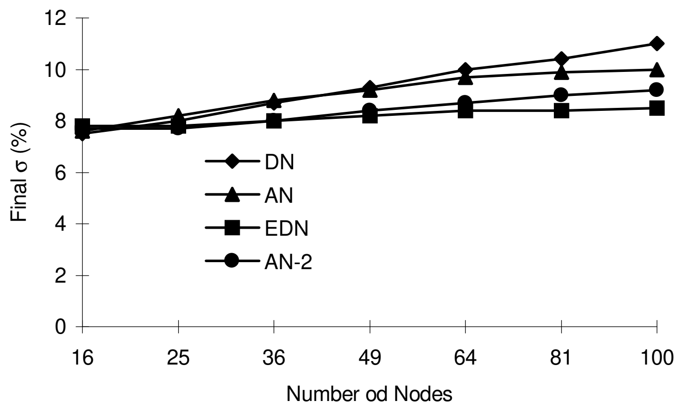


Figure 18. Balancing quality in function of the system size

7. Conclusions

The paper focuses on load balancing policies for massively parallel architectures. The fully distributed nature of massively parallel architectures rules out the possibility of a centralised load balancing control. Distributed implementations are needed but pay the cost of coordinating distributed decision centres toward a common goal: a local approach restricts each decision centre to a local view of the system and limits coordination among them, is the only one viable in massively parallel architectures.

The algorithms presented in this paper are based on the idea of load balancing the system only on the basis of space limited load information and by achieving a global goal by composing local actions. A similar approach is typical phenomenon in the physical word: **diffusion** forces a system toward an homogeneous distribution only on the basis of the local state. Diffusion has inspired the implementation of the algorithms presented in the paper, with different coordination degree among the neighbourhood. The paper evaluates these algorithms and analyses their performance in balancing different load situation with different kind of unbalances: several parameters are introduced toward a metrics of the load.

All the implemented algorithms show themselves well-scalable, achieving good load balancing even for large sized systems. Moreover, all algorithms are robust, i.e. their dependence on the particular kind of unbalance is minimised. However, the paper shows that by enlarging the locality scope of the algorithms best load balancing performance can be obtained, unless the dynamicity of the applications increases. In the latter case, algorithms with a very limited locality scope perform better.

These results make further work in the area of adaptive algorithms promising. In particular, we are currently investigating the viability of implementing diffusive based-load balancing capable of dynamically expanding or shrinking their domain of control so to adapt their behaviour to the dynamicity of the application. However, several problems are involved in this work, in particular there is a need of a larger degree of coordination among the distribution component, since the enlargements of the domain in one part of the system does not necessarily reflect in all part of the system. More intelligence has to be put on the dynamic load balancing policy to deal with this enlarged need of coordination, but the overhead this will impose on the system has still to be proved little enough with respect to the performance benefits achieved such intelligent load balancer.

8. Acknowledgements

The authors like to thanks the Research Institute for Parallel Computing Systems (IRSIP) of Naples (I) for having made them available its computing resources.

9. References

[AhmG91] I. Ahmad, A. Ghafoor, "Semi-Distributed Load Balancing for Massively Parallel Multicomputer Systems", IEEE Transactions on Software Engineering, Vol. 17, No. 10, Oct. 1991, 987-1004.

[AlmG94] G. S. Almasi, A. Gottlieb, "Highly Parallel Computing", Benjamin/Cummings Publishing Company, 1994.

[ArtF89] Y. Artsy, R: Finkel, "Designing a Process Migration Facility: the Charlotte Experience", IEEE Computer, Vol. 22, No. 9, Sept. 1989, 47-56.

[BarGW93] A. Barak, S. Guday, R. G. Wheeler, "The MOSIX Distributed Operating System: Load Balancing for UNIX", Lecture Notes in Computer Science, No. 672, Springer-Verlag, 1993.

[BarS85] A. Barak and A. Shiloh, "A distributed Load-Balancing Policy for a MultiComputer", Software - Practice and Experience, Vol. 15, No. 9, Sept. 1985, 901-913.

[BenD94] K. Benmohammed-Mahieddine, P. M. Dew, "A Periodic Symmetrically-Initiated Load Balancing Algorithm for Distributed Systems", ACM Operating Systems Review, Vol. 28, No. 1, Jan. 1994, 66-79.

[Boa95] M. Boari, A, Corradi, L. Leonardi, C. Stefanelli, "Adaptive Routing for Dynamic Applications in Massively Parallel Architectures", IEEE Parallel and Distributed Technology, Vol. 3, No. 1, Spring 1995, 61-74.

[Boi90] J. E. Boillat, "Load Balancing and Poisson Equation in a Graph", Concurrency: Practice and Experience, Vol. 2, No. 4, Dec. 1990, 289-313.

[BonK90] F. Bonomi, A. Kumar, "Adaptive Optimal Load Balancing in a Non-Homogeneous Multiserver System with a Central Job Scheduler", IEEE Transactions on Computers, Vol. 39, No. 10, Oct. 1990, 1232-1250.

[BurM93] C. Burdorf, J. Marti, "Load Balancing Strategies for Time Warp on Multi-User Workstations", The Computer Journal, Vol. 36, No. 2, 1993, 168-176.

[Can93] M. Cannataro, Y. D, Sergeyev, G. Spezzano, D. Talia, "A Dynamic Load Balancing Strategy for Massively Parallel Computing", Lecture Notes in Computer Science, Vol. 694, Springer-Verlag, 1993.

[CasK88] T. L. Casavant, J. G. Kuhl, "A Taxonomy of Scheduling in General-Purpose Distributed Computing System", IEEE Transactions on Software Engineering, Vol. 8, No. 4, Feb. 1988, 141-154.

[Chow90] S. Chowdhury, "The Greedy Load Sharing Algorithm", Journal of Parallel and Distributed Computing, Vol. 9, No. 1, Nov. 1990, 93-99.

[Cla94] T. W. Clark et. al., "Parallelizing Molecular Dynamics using Spatial Decomposition", Scalable High-Performance Computing Conference, Knoxville (TN), May 1994.

[Con93] A. J. Conley, "Using a Transfer Function to Describe the Load Balancing Problem", Argonne National Laboratory, Technical Report No. ANL-93/40, Nov. 1993.

[Con95a] "Resource Management in Parallel and Distributed Systems: Static Scheduling", Special Issue of Concurrency Practice and Experience, Vol. 7, No. 5, Aug. 1995.

[Con95b] "Resource Management in Parallel and Distributed Systems: Dynamic Scheduling", Special Issue of Concurrency Practice and Experience, Vol. 7, No. 7, Oct. 1995.

[CorLZ92] A. Corradi, L. Leonardi, F. Zambonelli, "Load Balancing Strategies for Massively Parallel Architectures", Parallel Processing Letters, Vol. 2, No. 2&3, Sept. 1992, 139-148.

[CorLZ94] A. Corradi, L. Leonardi, F. Zambonelli, "The Benefits of Migration in a Parallel Objects Environment", Proceesings of the 2nd EUROMICRO Workshop on Parallel and Distributed Processing, IEEE CS Press, 1994, 291-298.

[Cyb89] G. Cybenko, "Dynamic Load Balancing for Distributed Memory Multiprocessors", Journal of Parallel and Distributed Processing, Vol. 7, No. 2, Feb. 1989, 279-301.

[DouO91] F. Douglis, J. Ousterhout, "Transparent Process Migration: Design and Alternatives and The Sprite Implementation", Software - Practice and Experience", Vol. 21, No. 8, Aug. 1991, 757-785.

[DutM94] S. Dutt, N. M. Mahapatra, "Scalable Load Balancing Strategies for Parallel A* Algorithms", The Journal of Parallel and Distributed Computing, Vol. 22, No. 3, Sept. 1994, 488-505.

[EagLZ86a] D. L. Eager, E. D. Lazowska, and J. Zahorjan, "A Comparison of Receiver-Initiated and Sender-Initiated Adaptive Load Sharing", ACM Performance Evaluation Review, Vol. 16, No. 1, May 1986.

[EagLZ86b] D. L. Eager, E. D. Lazowska, J. Zahorjan, "Adaptive Load Sharing in Homogeneous Distributed Systems", IEEE Transactions on Software Engineering, Vol. 12, No. 5, May 1986, 662-675.

[EagLZ88] D. L. Eager, E. D. Lazowska, J. Zahorjan, "The Limited Performance Benefits of Migrating Active Processes for Load Sharing", ACM SIGMETRICS Conference on Modelling of Computer System, ACM, 1988, 63-72.

[FonMD94] C. Fonlupt, P. Marquet, J. Dekeyser, "A Data-Parallel View of the Load Balancing: Experimantal Result on MasPar MP-1", Proceedings of the HPCN International Conference, Munich (D), April 1994, Lecture Notes in Computer Science, No. 796, Springer-Verlag, 338-343.

[FurTI90] M. Furuichi, K. Taki, N. Ichiyoshi, " A Multi-Level Load Balancing Scheme for OR-Parallel Exhaustive Search Programs", Proceedings of the 2nd Symposium on Principles and Practice of Parallel Programming, ACM, 1990, 50-59.

[GarJ79] M. R. Garey, D. S. Johnson, "Computers and Intractability: a Guide to the Theory of NP-Completeness", Freeman, San Francisco (CA), 1979.

[GosDI93] K. K. Goswami, M. Devarakonda, R. K. Iyer, "Prediction-Based Dynamic Load-Sharing Heuristic", IEEE Transactions on Parallel and Distributed Systems, Vol. 4, No. 6, June 1993, 638-648.

[HanS91] R. V. Hanxleden, L. R. Scott, "Load Balancing on Message Passing Architectures", Journal of Parallel and Distributed Computing, Vol. 13, No, 4, April 1991, 312-324.

[HonTC88] J. Hong, X. Tan, M. Chen, "From Local to Global: an Analysis of Nearest Neighbour Balancing on Hypercube", ACM Performance Evaluation Review, Vol. 16, No. 1, May 1988, 73-82.

[Hos90] S. H. Hosseini, B. Litow, M. Malkavi, J. McPherson, K.Vairavan, "Analysis of a Graph Coloring Based Distributed Load Balancing Algorithm", The Journal of Parallel and Distributed Computing, Vol. 10, No. 10, Oct. 1990, 160-166

[JuXY95] J. Ju, G. Xu, K. Yang, "An Intelligent Dynamic Load Balancer for Workstation Clusters", ACM Operating Systems Review, Vol. 29, No. 1, Jan. 1995.

[Kal88] L. V. Kalè, "Comparing the Performance of Two Dynamic Load Distribution Methods", Proceedings of the International Conference on Parallel Processing, IEEE CS Press, 1988, 8-12.

[KreK92] O. Kremien, J. Kramer, "Methodical Analysis of Adaptive Load Sharing Algorithms", IEEE Transactions on Parallel and Distributed Systems, Vol. 3 No. 6, Nov. 1992, 747-760.

[Kum94] V. Kumar, G. Elsesser, N. R. Vempaty, "Scalable Load Balancing Techniques for Parallel Computers", The Journal of Parallel and Distributed Computing, Vol. 22, No. 1, July 1994, 60-79.

[Kun91] T. Kunz, "The Influence of Different Workload Descriptions on a Heuristic Load.Balancing Scheme", Procedings of the Symposium on Operating System Principles, ACM, July 1991.

[Mar90] D. C. Marinescu, J. E. Lumpp Jr., T. L. Casavant, H. J. Siegel, "Models for Monitoring and Debugging Tools for Parallel and Distributed Software", The Journal of Parallel and Distributed Computing, Vol. 9, No. 2, June 1990, 171-184

[LelO86] W. E. Leland, T. J. Ott, "Load Balancing Heuristic and Process Behavior", ACM Proceedings of Performance, 1986.

[LinR92] H. C. Lin, C. S. Raghavendra, " A Dynamic Load Balancing Policy with a Central Job Dispatcher", IEEE Transactions on Software Engineering, Vol. 18, No. 2, Feb. 1992, 148-158.

[LinK87] F. C. H. Lin, R. M. Keller, "The Gradient Model Load Balancing Method", IEEE Transactions on Software Engineering, Vol. 13, No. 1, Jan. 1987, 32-38.

[LulMR93] R. Luling, B. Monien, and F. Ramme, "Load Balancing in Large Networks: A Comparative Study", Proceedings of the 3rd IEEE Symposium on Parallel and Distributed Processing, 1993.

[Mir90] R. Mirchandaney, D. Towsley, J. A. Stankovic, "Adaptive Load Sharing in Heterogeneous Distributed Systems", Journal of Parallel and Distributed Computing, 9, 1990, 331-346.

[NitL91] B. Nitzberg, V. Lo, "Distributed Shared Memory: a Survey on Issues and Algorithms", IEEE Computers, Vol. 24, No. 8, Aug. 1991.

[NiXG85] M. Ni, C. Xu, T. B. Gendrau, "A Distributed Drafting Algorithm for Load Balancing", IEEE Transactions on Software Engineering, Vol. 11, No. 10, Oct. 1985, 1153-1161.

[NorT93] M. G. Norman, P. Thanisch, "Models of Machines and Computation for Mapping in Multicomputers", ACM Computing Surveys, Vol. 25, No. 3, Sept. 1993, 263-302.

[Nut94] M. Nuttal, "A Brief Survey of Systems Providing Process or Object Migration Facilities", ACM Operating Systems Review, Vol. 28, No. 4, Oct. 1994, 64-79.

[QiaY91] X. S. Qian, Q. Yang, "Load Balancing on Generalized Hypercube and Mesh Multiprocessors", Proceedings of the 11th International Conference on Distributed Computing Systems, IEEE CS Press, 1991, 402-409.

[ShiKS92] N. G. Shivaratri, P. Krueger, M. Singhal, "Load Distributing for Locally Distributed System", IEEE Computer, Vol. 25, No. 12, Dec. 1992, 33-44.

[Smi88] J. M. Smith, "A Survey of Process Migration Mechanisms", ACM Operating Systems Review, Vol. 22, No. 3, July 1988. 28-40.

[Sta85] J.A. Stankovic, "Stability and Distributed Scheduling Algorithms", IEEE Transactions on Software Engineering, Vol. 11, No. 10, Oct. 1985, 1141-1152.

[SueW92] T. T. Suen, J. S. K: Wong, "Efficient Task Migration Algorithms for Distributed Systems", IEEE Transactions on Software Engineering, Vol. 3, No. 4, July 1992, 488-499.

[UehT91] M. Uehara, M. Tokoro, "An Adaptive Load Balancing Method in the Computational Field Model", ACM OOPS Messenger, Vol. 2, No. 2, April 1991, 109-113.

[WanM85] Y. T. Wang, R. J. T. Morris, "Load Sharing in Distributed Systems", IEEE Transactions on Computers, Vol. 34, No. 3, Mar. 1985, 204-217.

[WalB95] C. Walshaw, M. Berzins, "Dynamic Load Balancing for PDE Solvers on Adaptive Unstructured Meshes", Concurrency Practice and Experience, Vol. 7, No. 1, Feb. 1995, 17-28.

[Wu95] M. Y. Wu, "Symmetrical Hopping: a Scalable Scheduling Algorithm for Irregular Problems", Concurrency: Practice and Experience, Vol. 7, No. 7, Oct. 1995, 689-706.

[XuL92] M. Xu, F. C. M. Lau, "Analysis of the Generalised Dimension Exchange Method for Dynamic Load Balancing", The Journal of Parallel and Distributed Computing, Vol. 16, No. 4, April 1992, 385-393.

[XuL94] M. Xu, F. C. M. Lau, "Optimal Parameters for Load Balancing with the Diffusion Method in Mesh Networks", Parallel Processing Letters, Vol. 4, No. 2, 1994, 139-147.

[XuL95] M. Xu, F. C. M. Lau, "The Generalised Dimension Exchange Method for Load Balancing in k-ary n-cubes and Variants", The Journal of Parallel and Distributed Computing, Vol. 24, No. 1, January 1995, 72-85.

[Xu95] M. Xu, B. Monien, R. Luling, F. C. M. Lau, "Nearest Neighbour Algorithms for Load Balancing in Parallel Computers", Concurrency: Practice and Experience, Vol. 7, No. 7, Oct. 1995, 707-736.

[XuTM95] C. Z. Xu, S. Tschoke, B. Monien, "Performance Evaluation of Load Distribution Strategies in Parallel Branch and Bound Computation", Proceedings of the 7th IEEE Symposium on Parallel and Distributed Processing, 1995.

[XuH93] J. Xu, K. Hwang, "Heuristic Methods for Dynamic Load Balancing in a Message-Passing Multicomputer", Journal of Parallel and Distributed Computing, Vol. 18, No. 1, May 1993, 1-13.

[Zay87] E. R. Zayas, "Attacking the Process Migration Bottleneck", Proceedings of the 11th ACM Symposium on Operating Systems Principles, ACM, 1987, 1-12.

[Zhou88] S. Zhou, "A Trace-Driven Simulation Study of Dynamic Load Balancing, IEEE Transactions on Software Engineering, Vol. 14, No. 9, Sept. 1988, 1327-1341.

[CanP95] R. Candlin, J. Phillips, "The Dynamic Behaviour of Parallel Programs under Process Migration", Concurrency: Practice and Experience, Vol. 7, No. 7, Oct. 1995, 591-614.

[ShiS95] N. G. Shivaratri, M. Singhal, " A Load Index and a Transfer Policy for Global Scheduling Tasks with DeadLines", Concurrency: Practice and Experience, Vol. 7, No. 7, Oct. 1995, 671-688.

[CorLZ96] A. Corradi, L. Leonardi, F. Zambonelli, "High-Level Management of Allocation in a Parallel Objects Environment", to appear in the Journal of System Architectures.

10. Appendix: The convergence factor of the DN policy against the EDN one

Theorem:

The contribution of the diminishing of the global variance in a single step is greater in the EDN strategy than in the DN one.

Proof:

Let us suppose to have three nodes, say X, Y and Z, defining a domain D. Let the load be:

$L_X(t) = x + \Delta,\ L_Y(t) = x - \Delta,\ L_Z(t) = x - \Delta - h,\ \Delta > 0,\ h > 0$

The average load of the domain D composed of these three nodes is:

$$\mu_D = \frac{x + \Delta + x - \Delta + x - \Delta - h}{3} = x - \frac{\Delta + h}{3}$$

At time t, the variance of the domain composed of these three nodes is:

$$\sigma_D^2(t) = \frac{1}{3}((x + \Delta - \mu_D)^2 + (x - \Delta - \mu_D)^2 + (x - \Delta - h - \mu_D)^2)$$

By substituting the μ_D term and with some simple transformation one can find that:

$$\sigma_D^2(t) = \frac{2}{9}(\Delta + h)^2 + \frac{2}{3}\Delta^2$$

By applying the ND strategy for the X and Y nodes and leaving the load of the node Z untouched, the loads at time t+1 are:

$L_X(t+1) = x,\ L_Y(t+1) = x,\ L_Z(t+1) = x - \Delta - h$

Being the load moved only within a domain, the average load has not changed. Then, the variance of the domain D at time t+1 becomes:

$$\sigma^2_{D,DN}(t+1) = \frac{1}{3}((x-\mu_D)^2 + (x-\mu_D)^2 + (x-\Delta-h-\mu_D)^2)$$

By simplifying this expression one can find that:

$$\sigma^2_{D,DN}(t+1) = \frac{2}{9}(\Delta+h)^2$$

i.e., the load variance diminishes at time t+1, as it is expected.

Now, let us suppose to apply the EDN strategy and that the load flows from X to Y and, finally, to Z. Then, the load at time t+1 of the nodes are:

$$L_X(t+1) = x,\ L_Y(t+1) = x-\Delta,\ L_Z(t+1) = x-h$$

Thus, the variance at time t+1 of the domain D becomes:

$$\sigma^2_{D,EDN}(t+1) = \frac{1}{3}((x-\mu_D)^2 + (x-\Delta-\mu_D)^2 + (x-h-\mu_D)^2)$$

Again, by substituting the μ_D term and by transforming the expression one can find that:

$$\sigma^2_{D,EDN}(t+1) = \frac{2}{9}(\Delta+h)^2 - \frac{2}{3}\Delta h$$

Because $\Delta>0$ and $h>0$, not only the variance is diminished with respect to time t, but also with respect to the appliance of the DN strategy:

$$\sigma^2_D(t) > \sigma^2_{D,DN}(t+1) > \sigma^2_{D,EDN}(t+1)$$

By recalling the results of lemma 1, that states that diminishing the local variance of a domain contributes proportionally to diminishing the global variance, the EDN strategy is faster in decreasing variance that the DN one. In particular, the greater the difference of load among the sender and the receiver node (i.e., the h value) the greater the advantage.

•