Diffusive Load Balancing Policies for Dynamic Applications

Antonio Corradi
Dipartimento di Elettronica Informatica e Sistemistica - Università di Bologna
2, Viale Risorgimento - I-40136 Bologna
Ph.: +39-51-6443083 - Fax: +39-51-6443073
E-mail: acorradi@deis.unibo.it

Letizia Leonardi, Franco Zambonelli
Dipartimento di Scienze dell'Ingegneria - Università di Modena
213/b, Via Campi - I-41100 Modena
Ph.: +39-59-376735 - Fax: +39-59-376799
E-mail: {leonardi, zambonelli}@dsi.unimo.it

Abstract

Massively parallel architectures require decentralized and scalable system policies. The paper presents and evaluates a set of local dynamic load balancing strategies inspired by diffusion and characterized by different scopes of locality: the goal is to compare their performance depending on the application dynamicity. The paper shows that only slowly dynamic applications can take advantage of the diffusion policies with larger scope, because they are likely to base their decisions on non obsolete information only for limited load variations. Highly dynamic applications should preferably employ load balancing policies with a very limited scope, due to their promptness to react to load changes.

Keywords*: Load Balancing, Locality, Diffusion, Dynamicity, Massively Parallel Architectures*

Diffusive Load Balancing Policies for Dynamic Applications

1. Introduction

A parallel application defines a set of communicating components to be allocated onto the available physical resources of the target architecture. A static approach decides the allocation of the application components before execution and can take into account both execution and communication needs [NorT93]. A dynamic approach defers allocation decisions to execution time, to consider both the current system state and the dynamic needs peculiar to the application [EagLZ86]. While the static approach is effective when the application resource needs can be predicted easily, the efficient execution of applications with dynamic execution patterns requires run-time allocation decisions. In the dynamic case, decisions should be prompt for immediate use. For this reason, dynamic allocation policies usually restrict their attention to execution resources, neglecting the impact of the application communication patterns. Therefore, the goal of the allocation policy is **load balancing** (LB for short), i.e., the balance of the load of the system nodes.

In the case of massively parallel architectures, their distributed nature suggests decentralized control in the LB policies to avoid bottlenecks and to achieve scalability, but distributed protocols pay the price of the coordination towards the common LB goal. After ruling out a global coordination perspective, the **local approach** is the only possible choice [CorLZ92]: the scope of decisions is constrained to a locality to limit coordination efforts [LulMR91].

The algorithms presented in this paper achieve the LB goal only on the basis of load information belonging to a restricted space and by composing independent local actions. The **diffusion** phenomenon in the physical world follows the same criterion: it forces a system toward a homogeneous distribution, by working only on local states. Several LB algorithms can be modeled after diffusion with different neighborhoods and coordination degrees: from a

local algorithm in which coordination is limited to couple of nodes to less local algorithms that coordinate actions in larger domains.

This paper evaluates the behavior of these algorithms in a massively parallel architecture and analyses their performances in balancing different load situations with load patterns that exhibit different degrees of dynamism. Several papers in this area compare the efficiency of LB policies, either with static load situations or with a fixed load dynamicity [Xu95, Kao96]; the contribution of this paper is to show the impact of the application dynamicity on different diffusive LB policies.

2. Diffusive Load Balancing Policies

An effective implementation of an LB policy in a massively parallel architecture should follow precise design choices, among several possible ones [CasK88]. On the one hand, **distributed** implementations – in which decision components are present in all system nodes – avoid the bottleneck intrinsic in a centralized approach. On the other hand, the coordination degree among distributed components must be limited according to the **locality** principle, to keep overhead low and to make prompt decisions. Global coordination, instead, would introduce overhead and would slow down decision activities.

2.1 Definition

The above design considerations suggest the possibility of modeling LB policies on the physical phenomenon of **diffusion**. Diffusion balances non-homogeneous scalar distributions only on the basis of local states, by locally moving elementary items in the direction suggested by an energy minimization goal. Similarly, a distributed LB policy can balance the load only on the basis of local load information and by migrating executing items (e.g., processes, threads or active objects) along the directions of decreasing load. In particular, we define an LB policy as diffusive when:

- it is based on **replicated decision components**, each with the same behavior and capable of autonomous and asynchronous activity;
- the **LB goal is locally pursued**: the scope of the actions for each decision component is bound to a local area of the system (**locality domain**); each decision component tries to

balance the load of its domain as if it were the whole system, only on the basis of the **load information** in its domain.

- the domain of each decision component partially overlaps with the domain controlled by at least another component; the union of these domains achieves a **full coverage** of the whole system.

It can be easily proved that a diffusive policy can achieve, at least in static load situations, a global balancing of the system [Cyb89]: in fact, the composition of independent LB actions monotonically tends to a globally balanced load situation, due to the full coverage of the domains.

The above definition of diffusive LB policies excludes those policies that require global synchronization of their activities [Xu95]. Similarly, it excludes those policies where diffusion is used to globally propagate load information and to locally acquire a global view of the system [LinR87, LulMR91]. Neither of these approaches is scalable and both are difficult to implement in massively parallel architectures.

Furthermore, the paper restricts the study to adaptive policies in which the load situation of a node (i,e., underloaded, balanced or overloaded) is identified by comparing its load with the ones of the locality domain nodes. Non-adaptive policies, that identify the load state of a node on the base of fixed load levels, are not generally adaptable to the LB goal [ShiKS92].

2.2 Implementation Issues

Several different policies can derive from the diffusion scheme. Their implementation should deal with several issues in the local decision phases that must precede LB actions, in other words, before the load migrations [EagLZ86, ShiKS92]. In particular, for any node:

- **triggering** phase: the decision component identifies the conditions that start the successive LB phases;
- **state identification** phase: the decision component ascertains whether the nodes of its domain are in a state that requires LB actions;
- **location** phase: the decision component identifies, in the domain, the underloaded nodes (receivers) and the overloaded ones (senders) needing LB actions;

- **selection** phase: the decision component chooses which items to move from sender to receiver nodes.

The LB decision activity completes with the needed LB actions, i.e., migration of load items.

With regard to the **triggering phase**, LB policies can be either periodic or event-driven [ShiKS92]. Periodic policies operate at predefined intervals of time: they produce predictable overhead, but the choice of the optimal interval tends to be application dependent. Event-driven LB policies are triggered both by changes in the local load and by the receipt of updated load information. In other words, event-driven policies intrinsically adapt the frequency of their activities to the application evolution. Because event-driven policies are likely to produce high overhead if there are too many triggering events, a limited load variation does not cause any transmission of new load information. Event-driven diffusive LB policies are viable because the amount of information exchange is intrinsically limited by the locality scope.

With regard to the **state identification phase**, it is not useful to start working for any (even small) imbalance: when the load is quite close to a balanced situation, an LB policy could waste resources without achieving any significant progress [ShiKS92]. This is due to the non-negligible cost of re-allocation mechanisms and to item granularity: items can be too large to permit, by their movement, a reduction in the imbalance. LB policies should inhibit their actions whenever the recognized imbalance is likely to make them unsuccessful, i.e., it is below a given **threshold**. Also diffusive policies should adopt thresholds: each decision component must start balancing actions in its domain only if the amount of imbalance in the domain justifies it. Let us note that, in case of adaptive policies, the threshold values are dynamically calculated as a function of the current load.

Depending on the role of the nodes in the **location phase**, the LB policy is defined as either sender-initiated, receiver-initiated or symmetrically-initiated – when both sender and receiver nodes can start the location phase [EagLZ86]. Diffusive policies are not so strict with regard to the initiative: once the load states of the nodes in a domain have been identified, the partners of a migration relate to each other whichever node has initiated the action. The need for speeding up the policy activities suggests avoiding long negotiation phases [EaglZ86].

The important issue of the **selection phase** is load granularity: given that the load is not indefinitely divisible, the choice of which items to migrate must take into account the load of each item. In the case of too coarse load granularity (and with the influence of the threshold adopted), the LB policy stops its activity. This condition is essential to achieve effective LB actions: when load migrates from a sender to a receiver node, the LB policy must avoid moving an excess of load that could reverse the roles of the involved nodes, transforming the receiver into a sender and vice versa. Other requirements can restrict even further the choice of the items to migrate: a possible constraint that rules out the re-allocation of items is when they are tied to specific resources on the sender node. However, the paper neglects these issues strictly related to the peculiar characteristics of the target system and application.

Because diffusive policies proceed with actions that may overlap both in time and space on connected domains, the implementation of a diffusive LB policy must guarantee a **serialization** principle to grant consistency: a node cannot participate in concurrent LB actions in different (overlapping) domains. In other words, an LB action can proceed as long as it does not involve nodes already involved in other LB actions in different domains. Otherwise, the conflicting LB action – rather than being suspended – does take place at all, thus avoiding deadlock situations. For instance, a sender node cannot command migrations towards a receiver node currently receiving (or waiting to receive) load from a sender in another domain. The sender will engage the receiver node in other LB actions only when new load information triggers a new activity.

3. Basic Diffusive Algorithms

We have considered several diffusive LB policies with different definitions of locality domain. All of them achieve the full coverage in a topology that connects to each other nodes belonging to different domains.

We have implemented the local decision component of the LB policy as one entity called Allocation Manager (AM for short), replicated in each node of the system. Each AM is in charge of the local implementation of the LB policy and of granting the serialization condition by coordinating itself with the other AMs.

3.1 The Direct Neighbor Policy

The **Direct Neighbor** policy (**DN** for short) chooses the minimal domain size: each locality domain consists of only two nodes directly connected by a physical link (figure 1).

Given one node *N1*, the arrival of updated load information from one of its neighbor (say *N2*) triggers the LB activity in the domain composed of the nodes *N1-N2*. The AM of *N1* compares the load of *N2* with its current load. If the *N1* load exceeds the load of *N2* by more than a threshold (calculated as a percentage of the current local load), the AM of *N1* tries to balance the loads of *N1* and *N2*.

Note that the state identification and the location phases overlap: the state identification phase compares the load of two nodes and immediately establishes the sender-receiver couple. The policy does not include any bidding to determine the new location [ShiKS92]: only the serialization of the activities in overlapping domains has to be guaranteed. For instance, the node *N3* in the *N3-N1* domain cannot command any migration toward *N1* if *N1* is already involved in a LB action within the *N1-N2* domain. On completion, *N1* can accept a migration.

The selection phase ascertains that migration of items does not reverse the sender/receiver roles of the involved nodes: this condition guarantees that balancing actions always go in the direction of balancing the load in the domain.

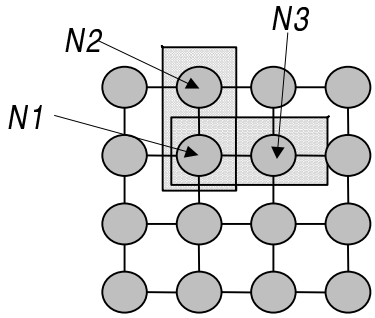


Figure 1. Two overlapping domains in the Direct Neighbor policy on a mesh topology.

3.2 The Average Neighborhood Policy

The **Average Neighborhood** policy (**AN**) enlarges the dimension of locality domains: one domain is constituted by one node and all its direct neighbors (figure 2). If *K* is the number of the direct neighbors of a given node *N*, *N* belongs to at most *K+1* different

domains: it is the center in one domain (we call it the *Master* of this domain), being a direct neighbor of all the other nodes of the domain; it is a peripheral node (*Slave* node) in the other domains to which it belongs.

The AM of one node *N* is in charge of the LB decisions for the domain *D* of which it is the master. This solution is the most natural, because *N* is the only node in *D* that can directly access and control all the *D* nodes. The AM of the node *N* keeps load information about every node of *D* and dynamically computes both the average domain load and two thresholds, T_{sender} and $T_{receiver}$, centered around it. A node is classified as a sender in *D* if its load exceeds T_{sender}, a receiver if its load is under $T_{receiver}$, and a neutral node in any other case.

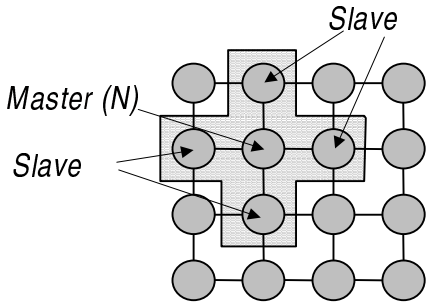


Figure 2. One domain *D* in the Average Neighborhood policy on a mesh topology.

Whenever the master node *N* receives an updated load information, this trigger forces the AM to recompute the average domain load and the thresholds T_{sender} and $T_{receiver}$, and to evaluate the load situation in *D*. The AM of the node *N* tries to push/pull the load of all the nodes in the domain *D* (and not only the load of the master node itself) as close as possible to the average domain load and without issuing migrations that could reverse the role of the nodes in the domain. Also in this policy, any participating node should not be concurrently involved in more LB actions, to follow the serialization principle.

3.3 The Average Extended Neighborhood Policies

The AN strategy defines the locality domains on the basis of the direct neighborhood relationship with the master node; a natural extension is to enlarge the size of domains. A domain can be defined as the set of nodes whose distance from the master node (in terms of hops) is either less than or equal to *d*. With *d*=1, the strategy is the previously described AN, with *d*=2 the strategy defines an average neighborhood of second level, **AN-2** for short, and so on (AN-d strategy).

Though still diffusive, the implementation of an AN-d strategy, with *d>1*, introduces more problems than the simple AN one: it requires communications between non-neighbor nodes and the master node may have problems in promptly evaluating the current average load of the enlarged domain. For these reasons, we limit our study to the AN-2 policy.

3.4 The Direct Neighbor Repeated Policy

The **Direct Neighbor Repeated** (**DNR**) policy exploits more information than the DN one does during the location phase, while it maintains the same triggering, state-identification and selection phases. In the DN policy, once a sender-receiver couple *N1-N2* is established, the migrating load can move, from the sender node *N1*, only to the receiver node *N2*. In its turn, *N2* can have an even less loaded neighbor, say *N3*, belonging to a different DN locality domain. The DNR strategy identifies these situations and allows receiver nodes to directly forward the migrating load to more and more underloaded nodes (arrows in figure 3 show the forwarding actions). Load migration stops only when there are no longer useful movements, i.e., the load arrives at a node *Nx* whose load is minimal in its neighborhood (*N6* in figure 3).

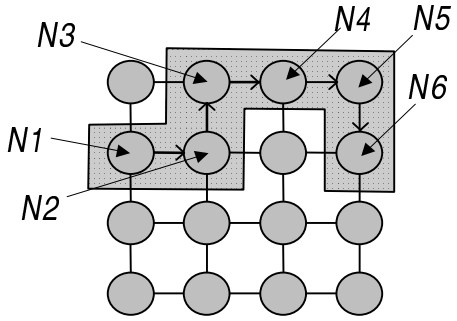


Figure 3. One domain in the DNR Policy on a mesh topology. The simple domain *N1-N2* **of the DN policy dynamically extends to the underloaded nodes** *N3, N4, N5* **and** *N6*.

The DNR policy enlarges locality of LB actions by providing the possibility to move items in successive hops as a single migration. From an implementation point of view, the load forwarding of the DNR policy is easy to implement: when a direct-neighbor couple is established and a node *Ny* is waiting for migrating load, *Ny* knows whether one less loaded node exists in its neighborhood. In this case, *Ny* acts as a forwarder toward the underloaded neighbor. Any other node, down to a node with a local minimum of load, does the same.

The DNR policy is still a diffusive one. The main difference from the previously described ones is the dynamic definition of the locality domains: the nodes involved in a

migration can dynamically determine one domain composed of a chain of neighbor nodes (from *N1* to *N6* in figure 3).

4. Performance Comparison of the Implemented Diffusive Strategies

To evaluate the diffusion policies, we have used a transputer-based architecture (a 100-nodes Meiko CS-1). The possibility of configuring different network topologies makes this architecture a suitable evaluation target.

We have allocated several application processes, dynamically created and destroyed, onto the system nodes to provide execution load and emulate in place the behavior of dynamic parallel applications. This choice permits the evaluation of a wide range of load situations without being committed to a specific parallel application.

4.1 Load Metrics

We have identified a few significant global load indicators to analyze our diffusive load balancing policies:

- *granularity*, i.e., the coarseness of the items allocated on the nodes of the systems;
- *imbalance*, i.e., the deviation of the load situation from the ideal balanced one;
- *dispersion*, i.e., the patterns of the load distribution of the system nodes;
- *dynamicity*, i.e., the frequency of the load changes.

The load *granularity* identifies the average number of items allocated to each node and the amount of load per item imposed on a node. We have considered a range from very coarse-grained loads (less than 10 processes per node) to very fine-grained ones (about 100 processes per node) with different item loads. This reflects the characteristics of different real-world applications, characterized by different excess of parallelism. The tests have shown that while granularity influences the achieved LB results, its impact is independent on the adopted LB policy: when granularity is too coarse, it prevents any movement and does not permit the achievement of good balancing. Similarly, the presence of items with different load worsens the achieved balance, because of longer selection phases.

The granularity of the load determines the optimal threshold for the diffusive LB policies. The best tuning choice is to make threshold comparable with granularity: because granularity

can limit load exchanges, a too fine-grained threshold w.r.t. granularity does not permit further increases in LB quality over a more reasonable choice. The influence of these parameters does not change the relative performance of the policies. For this reason, we report only the results for a granularity of 20 – with all the items of the same load – and a threshold of 20% of the domain load.

The load *imbalance* is measured by the global standard deviation (σ) of the load of all the *N* system nodes, normalized to the average system load (μ):

$$\sigma = \frac{1}{\mu}\sqrt{\frac{\sum_{i=1}^{N}(Li-\mu)^2}{N}} \qquad \text{where } Li \text{ is the load of the } i\text{-}th \text{ system node and } \mu = \frac{1}{N}\sum_{i=1}^{N} L_i$$

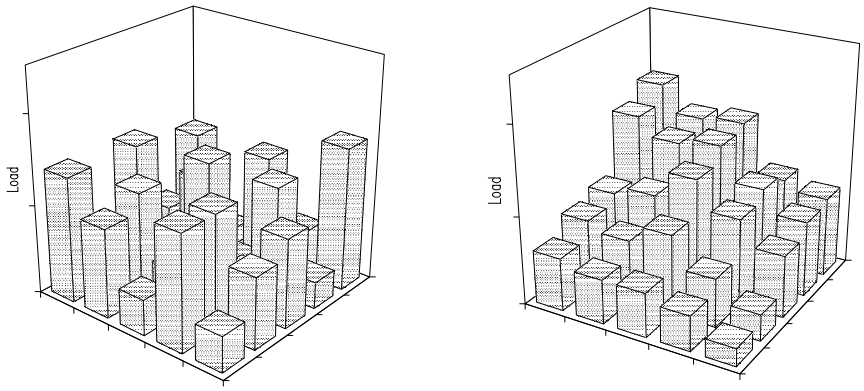
The capacity to keep σ low during the execution is the main quality parameter for a load balancing policy, because it can dramatically influence the application response time.

In addition, the imbalance in the system can be more or less evenly distributed: either overloaded and underloaded nodes can be present in every part of the system or they can be concentrated in regions (see figure 4). The load *dispersion* indicator (δ) measures this factor: it represents the global load standard deviation (again normalized to the average system load) computed as if each node had a load equal to the average of its load and of its neighbor nodes:

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

where μ_i is the average load of a neighborhood *Di*, composed of N_i nodes:

$$\mu_i = \sum_{h\in Di}\frac{L_h}{N_i}$$



(a) **(b)**

Figure 4. Different load dispersion with fixed standard deviation (σ=50%) on a 5x5 mesh topology. (a: δ=30% of σ; b: δ=80% of σ)

At one extreme, if δ is small compared with σ, each neighborhood reflects a local imbalance comparable to the global system imbalance, as in figure 4a. In other words, the imbalance is evenly distributed in the system. At the other extreme, δ close to σ means that the global imbalance cannot be easily identified in its real magnitude with a local view of the system (as it is the view of diffusive LB policies), as in figure 4b. In other words, the load varies in a continuous way and the local imbalance is very small in any neighborhood, because even a significant global imbalance can consist of only minimal contributions in any neighborhood.

To characterize load *dynamicity*, we have extended the σ and δ indicators to take into account load variability. In particular, we measure load dynamicity as the normalized standard deviation of the load changes between time *t* and *t+Δt*, computed as if they were the only load to account for:

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

where $\Delta L_i(t, t+\Delta t)$ is the load change of the node *i* in the Δt interval and $\mu(t)$ is the average system load at time *t*. Let us stress that the Δt interval must be small enough to capture all significant variations of load and to avoid situations where load is generated and then destroyed within the same interval.

A more integral indicator of the load dynamicity takes into account the load variations in contiguous intervals of time over a time unit T:

$$\Delta\sigma / T = \frac{1}{T}(\Delta\sigma\ (t, t+\mathrm{T})) = \frac{1}{T}(\sum_{i=1}^{T/\Delta t}\Delta\sigma(t+(i-1)\Delta t, t+i\Delta t))$$

The dispersion of the generated load (Δδ/T, defined analogously to Δσ/T) can further characterize the dynamic load changes.

In our experiments, we have considered a wide range of dynamic situations, from static and quasi-static ones (Δσ/T≈0) to highly dynamic ones (Δσ/T=500/sec) and with different values of dispersion of the generated load (Δδ/T=30%-70% of Δσ/T). This range of dynamicity covers practical application cases that could benefit from dynamic load balancing: over this limit, the load changes so quickly that no LB policy can produce significant benefits (as discussed in sub-section 4.3).

In the following, we discuss the experimental results obtained on a 100 nodes (10x10) mesh, unless otherwise stated. Of course, several runs of each testbed have been performed to obtain average data.

4.2 The Influence of Dynamicity

Figures 5 and 6 plot the average LB quality (expressed as the average standard deviation of the system during execution) achieved by the implemented diffusive policies depending on the dynamicity of the system load, i.e., of $\Delta\sigma/T$, for different values of $\Delta\delta/T$ (30% and 70% of $\Delta\sigma/T$). These two figures also report the average standard deviation in the absence of any LB policy (No Load Balancing case, NLB for short). Figures 7 and 8 report the LB effort (expressed as the number of migrations per second).

4.2.1 Slowly Dynamic Load Situations

In quasi-static load situations, i.e., with a very low degree of dynamicity ($\Delta\sigma/T \leq 10$/sec, left sides of figures 5–8), all the diffusion policies achieve the goal of balancing the system load with an acceptable number of migrations, but with different LB quality.

When $\Delta\delta/T$ is low (figures 5 and 7), the imbalance tends to be evenly distributed in the system and all policies achieve quite good a LB quality with a comparable number of migrations. When $\Delta\delta/T$ increases (figures 6 and 8), policies with too limited domains make it impossible to detect a global imbalance. In this case, the DN policy cannot overcome a high residual σ, while AN and AN-2 policies are less and less sensitive to the $\Delta\delta/T$. By dynamically enlarging locality, the DNR policy achieves the best LB quality with the lowest number of migrations, almost independently of $\Delta\delta/T$.

It is interesting to note that a low degree of dynamicity not only preserves the effectiveness of the LB policies, but it can even improve the achieved LB quality over the static case. In fact, a continuous evolution of the load can avoid situations where the LB policies stop in a still unbalanced situation: this can happen when all the domains are almost locally balanced but the whole system is not (in case of δ comparable to σ).

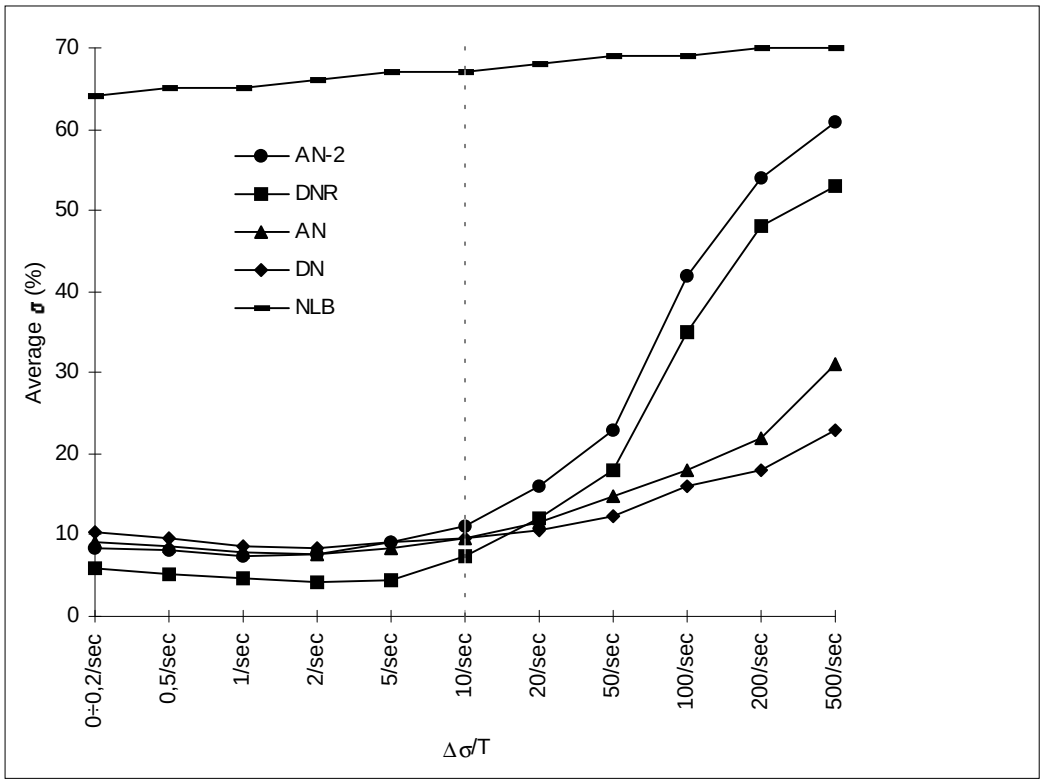


Figure 5. Influence of dynamicity on the LB quality (Δδ/T= 30% of Δσ/T).

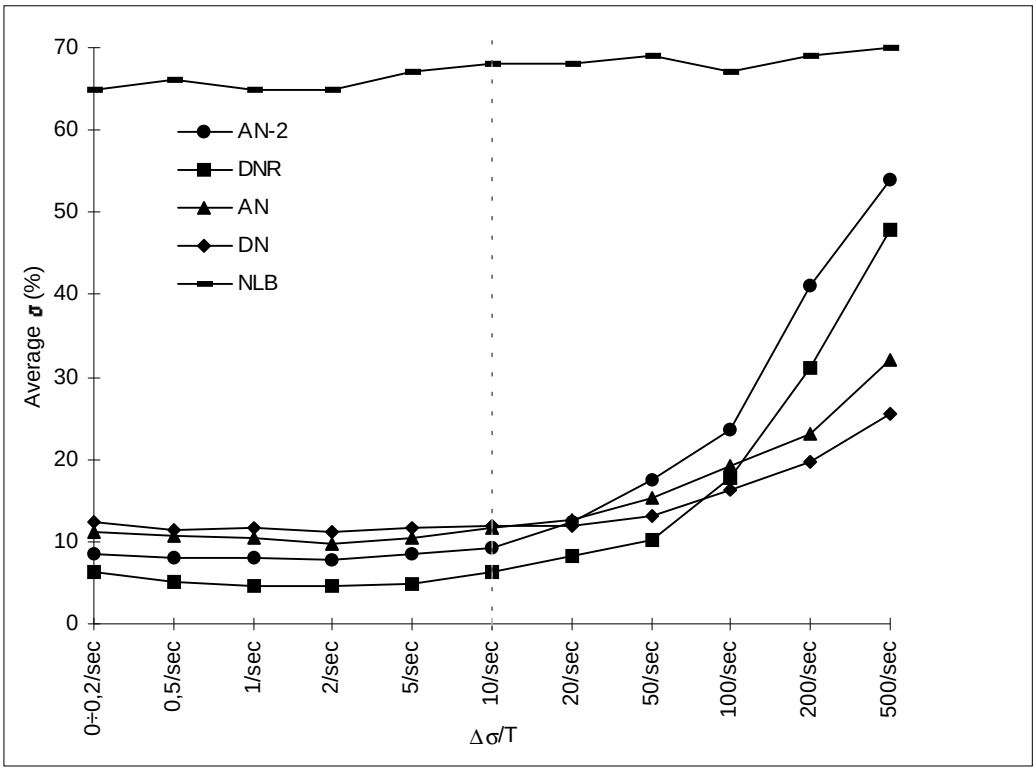


Figure 6. Influence of dynamicity on the LB quality (Δδ/T = 70% of Δσ/T).

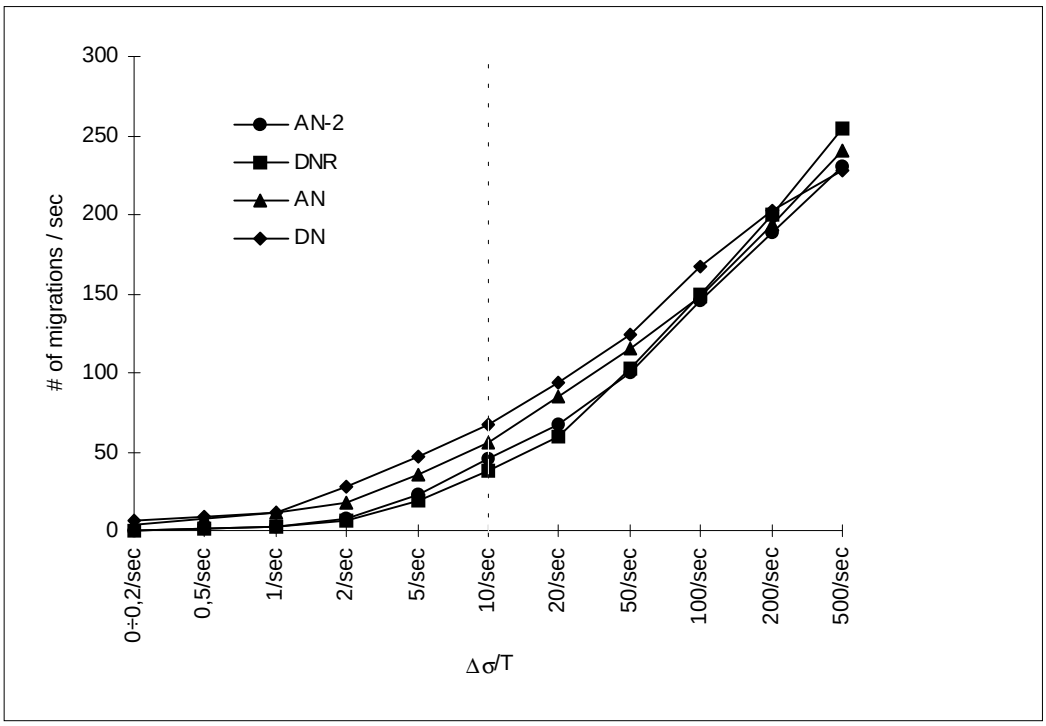


Figure 7. Influence of dynamicity on the LB effort (Δδ/T = 30% of Δσ/T).

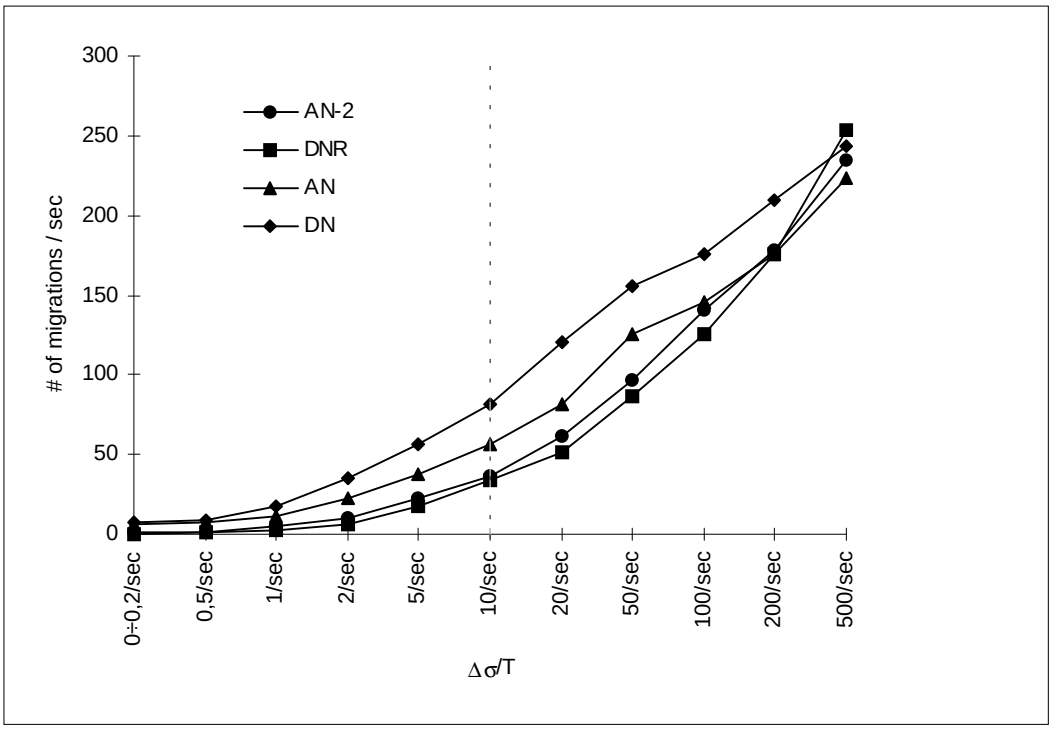


Figure 8. Influence of dynamicity on the LB effort (Δδ/T= 70% of Δσ/T).

4.2.2 Highly Dynamic Load Situations

In highly dynamic load situations ($\Delta\sigma/T>10$/sec, the right sides of figure 5–8), all the LB policies implemented tend to decrease the LB quality and to increase the LB effort. However, the enlarged domain policies, DNR and AN-2, behave worse than the AN and DN ones. The latter, in particular, exhibits the more robust performances.

Even in this case, $\Delta\delta/T$ influences the behavior of the policies: the higher the $\Delta\delta/T$ of the load generated, the higher the value of $\Delta\sigma/T$ at which the DNR and the AN-2 policies begin to operate worse than the AN and DN ones (see figures 5 and 6). In any case, for any $\Delta\delta/T$, there is always a point where policies with enlarged locality domains become less effective than more local ones.

These results can be understood by assimilating the dynamic LB tool to a control system [CanP95]. Dynamic LB policies could become ineffective in dynamic load situations: expensive LB actions could be useless and even damage applications, making worthwhile switching off load balancing. This behavior stems from two factors: **slowness** and **over-reactivity**.

If a policy reacts too slowly, it decides its actions possibly based on obsolete load information. Instead of producing a better balance, it could even worsen the imbalance. The AN-2 policy decisions, by taking into account the load information of also non-neighbor nodes, is likely to react with a significant delay w.r.t. the “age” of the available load information. The AN policy is less sensitive to this problem, because its decisions involve only load information coming from neighbor nodes. The DN policy reacts quickly, being triggered by any load change and based on very local load information.

The DNR policy is similar to the DN one for locality and promptness. Its limited capacity of balancing highly dynamic load situations stems from the fact that it is likely to over-react to the imbalances it tries to correct and is likely to worsen results. In fact, the DNR policy forwards the load without any threshold inhibition, becoming more sensitive to dynamic changes and increasing the probability of dangerous effects. To overcome this problem, we have tested the introduction of a threshold in the DNR policy: the load is forwarded only to those nodes that are underloaded “enough”, i.e., below the threshold introduced. This solution

increases the robustness of the DNR policy in highly dynamic situations, but assimilates its results to the DN policy in static and slowly dynamic situations, obscuring its peculiar advantages.

4.3 Discussion

The activities of all the policies induce a limited overhead, defined as the percentage of resources subtracted by the policy from those available to applications. In slowly dynamic applications ($\Delta\sigma/T<10$/sec), all the policies causes an overhead under 5%; in highly dynamic situations, the policies cause a higher, but still acceptable, overhead (under 10% when $\Delta\sigma/T<100$/sec).

A global indicator can be introduced to thoroughly evaluate the effects of different LB policies on parallel applications: it is the *normalized response time* [Keo96]. For a given LB policy, the normalized response time (*NRT* for short) is defined as follows:

$$NRT = \frac{RT_{NLB} - RT_{LB}}{RT_{NLB} - RT_{IDEAL}}$$

Where RT_{NLB} represents the response time in the absence of the LB policy, RT_{LB} is the response time obtained by applying the LB policy (by taking into account all the costs of its appliance), RT_{IDEAL} is the expected response time in the presence of an ideal LB tool that achieves a perfect load balancing at zero cost. The higher the efficiency of an LB policy, the closer to 1 is its *NRT*. The policy does not produce significant benefits on applications when *NRT* goes down to 0. A negative *NRT*, instead, reflects a performance degradation due to the LB policy.

Figures 9 and 10 report the *NRT* achieved by the diffusive LB policies depending on the application dynamicity and for two different values of $\Delta\delta/T$ (30% of $\Delta\sigma/T$ and 70% of $\Delta\sigma/T$). The process migration time, i.e., the time for a process to be frozen, transferred and resumed on the received node, is about 30ms in our target architecture.

All the policies are quite close to the ideal LB case for slowly dynamic situations. However, the DN and AN policies exhibit a worse *NRT*, because they need a higher number of migrations to reach a lower LB quality than the other policies. The AN-2 improves the *NRT*, due to the enlargement of its locality domains. The DNR policy achieves the best *NRT*,

by obtaining the best balancing quality with the lowest migration effort. As expected from the analysis of previous sub-sections, these results are even more evident in the case of high $\Delta\delta/T$ (as in figure 10).

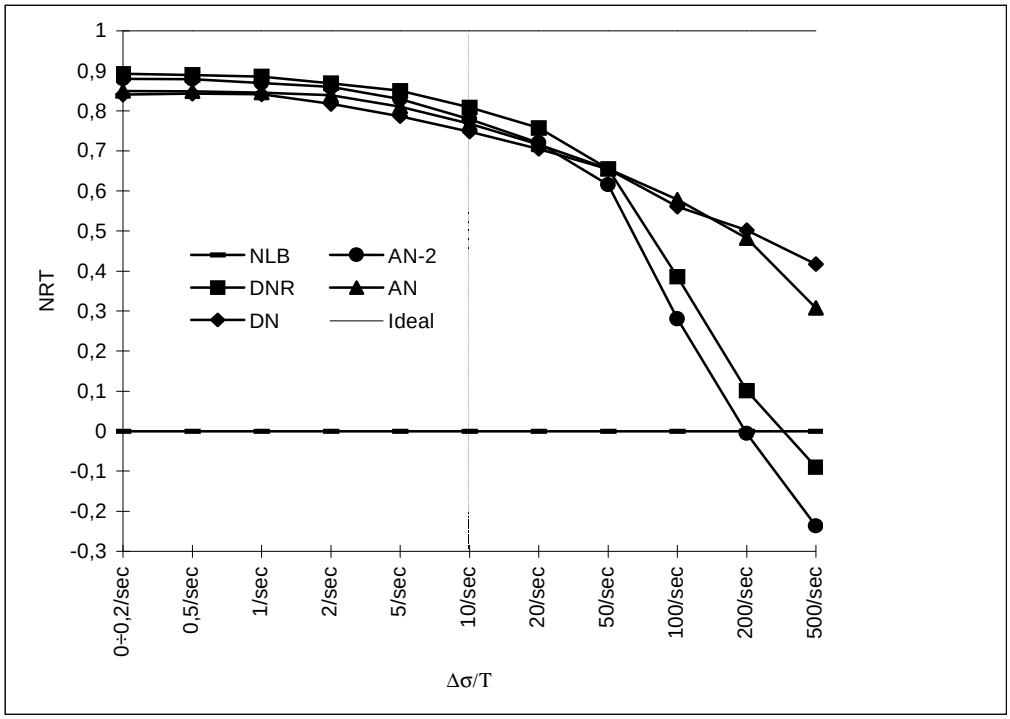


Figure 9. Influence of dynamicity on the normalized response times ($\Delta\delta/T$ = 30% of $\Delta\sigma/T$).

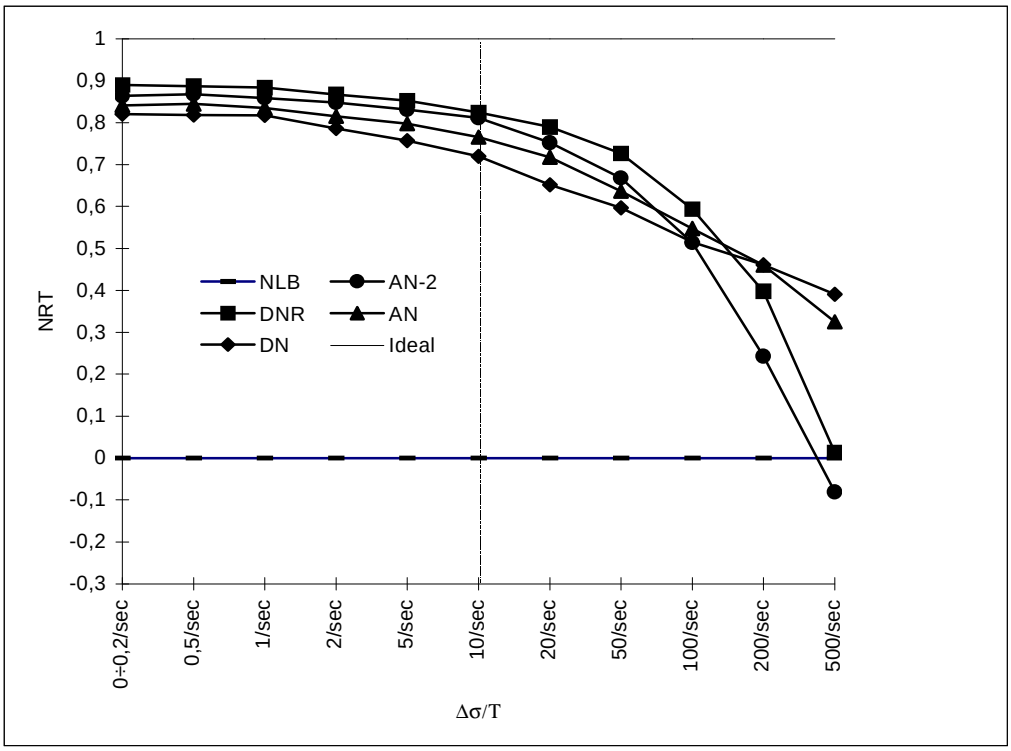


Figure 10. Influence of dynamicity on the normalized response times ($\Delta\delta/T$ = 70% of $\Delta\sigma/T$).

In highly dynamic situations, *NRT* decreases for all the policies, due to the worse balancing quality and to the increased LB effort. For the AN-2 and DNR policies, in particular, *NRTs* become negative. This signals that the application of these LB policies is not at all worthwhile. In this interval of dynamicity, only the AN and DN policies are effective. A further increase of dynamicity over the shown range leads to negative *NRTs* for all policies, suggesting the avoidance of any diffusive policy.

4.4 Scalability of the Policies

To evaluate the scalability of the LB policies, i.e., their ability to work well independently of the system size, we have run the algorithms in target systems of different sizes: in particular, we have used 2-D mesh configurations with a number of nodes varying from 16 to 100. Two important factors influence the scalability of the policies: their overhead and their capacity of achieving good LB quality.

All the diffusive LB policies show only a light dependence on the system size w.r.t. their overhead. Figure 11 shows that, in the case of $\Delta\sigma/T=10$/sec. the overhead introduced is only slightly dependent on the system size. More dynamic situations exhibit the same behavior.

With regard to the final LB quality, the four policies behave independently of the system size. Only in very small-sized systems (i.e., less than 25 nodes) are the differences in behavior of the policies less perceivable (see figure 12): in these cases, even policies with a very limited scope acquire an adequate “global” view of the system.

To analyze the behavior of LB policies in larger systems, we have considered, for our target, different topologies with larger diameters (the maximum distance between any two nodes in the system) [Keo96]. Not only do these tests confirm both policies’ scalability and relative behavior depending on dynamism, but they also suggest that the reported data can apply to many parallel architectures with different interconnection networks. In fact, the locality concept intrinsic to diffusive policies maintains its validity independently of the peculiar architecture properties, such as the network topology and the presence of specialized communication hardware.

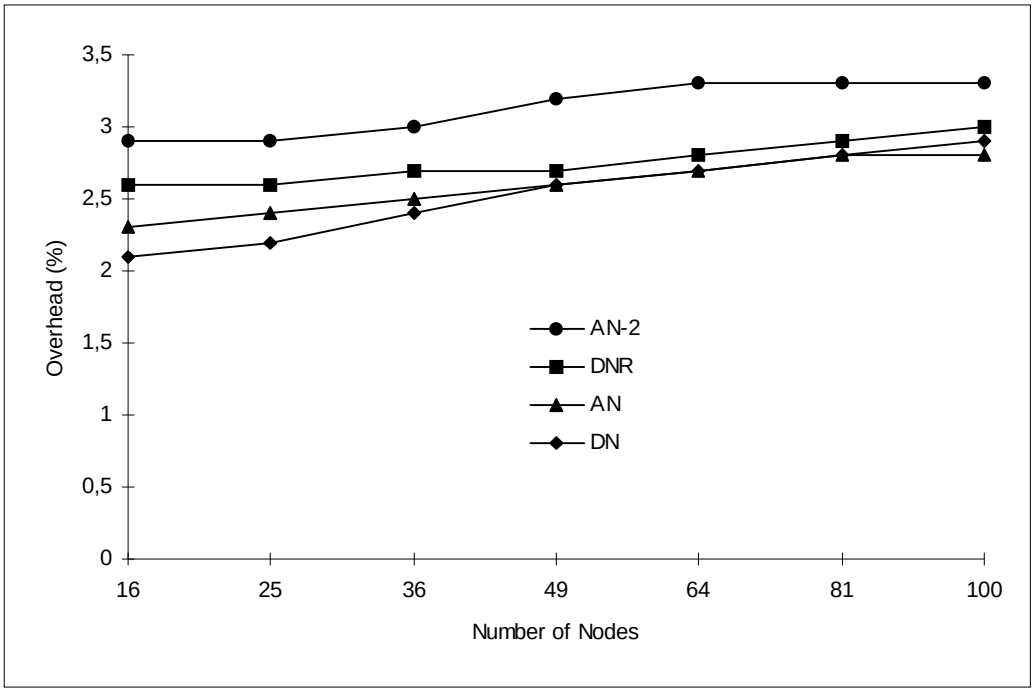


Figure 11. Overhead of the policies depending on the system size (Δσ/T= 10/sec, Δδ/T= 50% of Δσ/T).

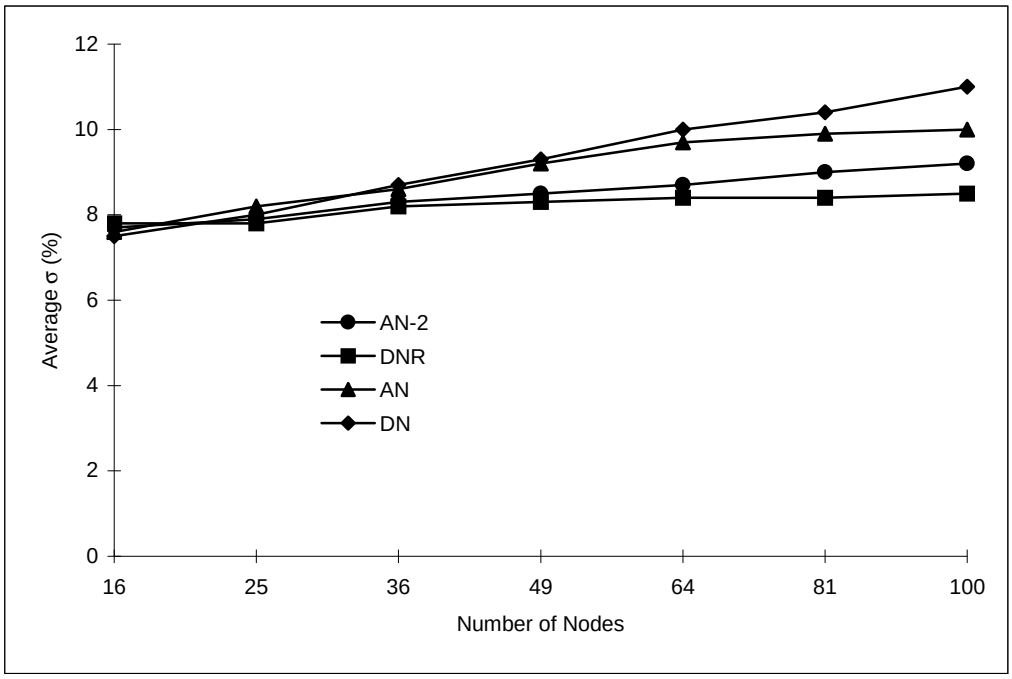


Figure 12. LB quality depending on the system size (Δσ/T= 1/sec, Δδ/T = 50% of Δσ/T).

As an additional consideration, we have ported all the presented diffusive policies to distributed architectures composed of workstations interconnected by high-speed LANs. In this case, even if communications and consequent process migrations are more expensive than in massively parallel architectures, the first results confirm the trend obtained for the Meiko CS-1.

5. Related Work

Local diffusive LB policies, based on either DN or AN concepts, have been widely studied in the literature. However, many papers deal with formal issues rather than concrete ones [Cyb89, XuL95]. On the one hand, they assume synchronous operations with all distributed decision components acting at the same time: this leads to difficult and non-scalable implementations in parallel architectures. On the other hand, these papers aim to compute the convergence rate of the LB algorithms for static load situations with no interest in dynamic applications. Although several of them show that, to achieve fast convergence, it is better not to completely balance the nodes in one domain, these considerations are significant in synchronous implementations, but do not apply to asynchronous implementations, as in our experience and in [XuL94, XuL95].

Other authors evaluate DN- and AN-like LB policies for a few concrete applications but do not consider their general properties [DutM94, XuTM95]. Papers describing practical implementations of diffusive LB policies, tested in environments similar to our one, provide useful information on the behavior of the policies, but they lack to evaluate the effects of different dynamicity degrees [WilR93]. A detailed analysis of load balancing in a dynamic environment can be found in [CanP95]. However, the goal in that case is not to compare the behavior of different LB policies but rather to build a general model for dynamic parallel applications under dynamic LB.

[Can93] presents one algorithm based on an enlarged domain strategy, similar to our AN-2 policy, but it can be no longer classified as diffusive because it introduces random parameters to deal with information obsolescence. An LB algorithm with a scheme similar to the DNR policy is presented in [Kal88] with two additional parameters: a maximum number of forwarding moves and a minimum moving distance from the sender node. These parameters tend to avoid an excess of load forwarding and to overcome the limits of locality in the case of high dispersion of the load; however, they require an application-dependent tuning which makes the policy less general. Another interesting DNR-like algorithm is in [Wu95], with an extended (non strictly local) state identification phase that aims to improve

the behavior of the simplest DNR scheme in the case of high dispersion, but makes the policy more sensitive to information obsolescence.

6. Conclusions

The efficient execution of dynamic parallel applications requires run-time allocation decisions to meet the load balancing goal. The design of an optimal and general-purpose allocation tool seems impossible: the tool strategy should adapt to the application dynamicity. The paper shows that, within the context of diffusive policies, no policy is suitable for all kinds of parallel applications. In fact, the enlargement of the locality domains in diffusive policies produces better results for slowly dynamic applications and poorer performances for highly dynamic ones.

We are interested in the design of a tool to automatically decide the most suitable locality domain by monitoring the application dynamicity: this should automatically adapt the internal parameters of the LB policy to the application behavior without any user involvement. In addition, we are facing the impact of other application-dependent parameters that can influence the performance of a diffusive load balancing policy, such as the communication and the synchronization patterns of applications.

Acknowledgements

This work has been carried out under the financial support of the Italian Ministero dell'Università e della Ricerca Scientifica e Tecnologica (MURST) in the framework of the Project "Design Methodologies and Tools of High Performance Systems for Distributed Applications". We also thank the anonymous referees for their suggestions that helped us to improve the quality of the paper.

References

[CanP95] R. Candlin, J. Phillips, “The Dynamic Behaviour of Parallel Programs under Process Migration”, Concurrency: Practice and Experience, Vol. 7, No. 7, pp. 591-514, Oct. 1995.

[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, pp. 141-154, Feb. 1988.

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

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

[EagLZ86] D. L. Eager, E. D. Lazowska, J. Zahorjan, “Adaptive Load Sharing in Homogeneous Distributed Systems”, IEEE Transactions on Software Engineering, Vol. 12, No. 5, pp. 662-675, May 1986.

[Keo96] P. K. Keong Loh, W. Jing Hsu, C. Wentong, N. Sriskantan, “How Network Topology Affects Load Balancing”, IEEE Parallel and Distributed Technology, Vol. 4, No. 3, pp. 25-35, Fall 1996.

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

[LulMR91] R. Lueling, B. Monien, F. Ramme, “Load Balancing in Large Networks: A Comparative Study”, Proceedings of the 3rd IEEE Symposium on Parallel and Distributed Processing, pp. 686-689, IEEE CS Press, 1991.

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

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

[Xu95] C. 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, pp. 707-736, Oct. 1995.

References of the Related Work Section

[Can93] M. Cannataro, Y. D, Sergeyev, G. Spezzano, D. Talia, “A Dynamic Load Balancing Strategy for Massively Parallel Computing”, Lecture Notes in Computer Science, No. 694, Springer-Verlag (D), 1993.

[CanP95] R. Candlin, J. Phillips, “The Dynamic Behaviour of Parallel Programs under Process Migration”, Concurrency: Practice and Experience, Vol. 7, No. 7, pp. 591-514, Oct. 1995.

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

[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, pp. 488-505, Sept. 1994.

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

[WilR93] M. H. Willebeck-Le Mair, A. P. Reeves, “Strategies for Dynamic Load Balancing on Highly Parallel Computers”, IEEE Transactions on Parallel and Distributed Systems, Vol. 4, No. 9, pp. 979-993, Sept. 1993.

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

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

[XuL95] C. 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, pp. 72-85, Jan. 1995.

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