How to Improve Local Load Balancing Policies by Distorting Load Information

Franco Zambonelli

Dipartimento di Scienze dell'Ingegneria - Università di Modena

213/b, Via Campi - 41100 Modena - Italy

Ph.: +39-59-376735 - Fax: +39-39-376799

E-mail: franco.zambonelli@unimo.it

Abstract

The paper focuses on local load balancing policies for massively parallel architectures and introduces a new scheme for load information exchange between neighbor nodes. The idea is to distort the exchanged load information to let the policy keep into account a more global view of the system and overcome the limits of the local scope. The presented scheme has been integrated into two variants of a direct-neighbor policy and evaluated in dependence of the characteristics of the system load. Experimental results show that the transmission of distorted load information provides high efficiency unless the dynamicity of the load becomes too high, in which case it is preferable to exploit non-distorted load information.

Keywords*: Massively Parallel Architectures, Dynamic Load Balancing, Direct-Neighbor Policies, Load Information Exchange, Performance Evaluation*

1. Introduction

Dynamic load balancing is required for parallel applications characterized by non-predictable patterns in the accesses to the system resources. The main goal is to dynamically tune the allocation of the application components onto the target architecture to effectively exploit the

execution resources and achieve good application speed-ups [ShiKS92]. In this perspective, dynamic load balancing can be assimilated to a *system control problem* [CanP95, CorLZ96]: the execution of the application must be monitored to detect its evolution and identify load imbalances; actions to lead the system to a balanced configuration must be decided and, then, performed.

The load balancing policy represents the decisional component of the dynamic load balancing control tool and, as that, has to guarantee:

- *stability*: the capacity of achieving the load balancing goal without wasting resources in actions not effective toward the solution;
- *low intrusion*: the capacity of limiting its overhead on the controlled system;
- *generality*: the capacity of acting independently of the specific properties of the controlled system, i.e., of the behavior of the applications;

In addition, any effective implementation in massively parallel architectures has to face the issue of *scalability*: the policy must show a limited dependence on the system size to work effectively even on large systems.

The above requirements constraint the possible choices in the design of the general structure of a load balancing policy with the target of a massively parallel architecture. First of all, a *distributed* approach is needed: autonomous and asynchronous decisional components are replicated and distributed over all the system nodes. Each component is in charge of the allocation decisions for its node, eventually by cooperating and coordinating itself with other decisional components. In addition, the need for scalability suggests limiting the coordination degree among the decisional components of distributed load balancing policy [LulMR91, Xu95]. In particular, both the amount of information exploited and the scope of the actions must be limited to a sub-set of the system nodes, according to a locality principle (either logical or physical, depending on the characteristics of the target architecture). Global policies, that consider information about the load of all nodes of the system and can extend their actions over the whole system, requires high coordination, and make the intrusion of the policy unacceptable in large systems [CorLZ92].

The distributed load balancing policies presented in this paper base their decisions only on load information coming from the neighborhood and achieve the global load balancing goal by composing local load balancing actions. The limited amount of load information available to the policies can sometime lead them to inefficient load balancing actions or even to stop working because of the recognition of a local load balance, without the capability of recognizing global imbalances. To overcome this problem, the paper introduces a new scheme for the *exchange of load information*. On the one hand, each node transmits a load information that is a weighted sum of its effective load and of the load of all its neighbors; on the other hand, the distorted load information incoming to a node is considered as representative of the sender load.

The above scheme has been integrated into both a simple *direct-neighbor policy* and an extended direct-neighbor policy that adopts a *non-local load distribution* scheme. The impact of the load information exchange scheme on the presented policies has been evaluated on a transputer-based architecture depending on the characteristics of the system load, in particular its dynamicity. The experiments show that the scheme achieves effective load balancing when the dynamicity of the application is low. In these cases, the adoption of the extended load distribution scheme achieves further benefits. When the dynamicity of the applications increases, instead, it is better to exploit non-distorted load information and adopt the basic direct-neighbor policy with the local load distribution scheme.

The paper is organized as follows. Section 2 describes the basic direct-neighbor policy. Section 3 presents the distorted load information exchange scheme and the extended load distribution scheme. Section 4 evaluates the effectiveness of these policies on a transputer-based architecture and compares their performances. Related works are discussed in section 5.

2. The Basic Direct-Neighbor Policy

The direct-neighbor policy achieves load balancing with asynchronous local actions limited in scope. Every node communicates only with its direct-neighbors and exchanges load information only with them. Load balancing actions are limited to two direct-neighbor nodes [LulMR91, XuL95].

A load balancing action within two neighbor nodes *i* and *j* strives to equalize their loads (see figure 1). Let $L_i(t)$ be the load of node *i* at time *t* and $L_j(t)$ the load of node *j*. At the end of the load balancing action, say at time *t+1*, the loads of these nodes – in the abstract case of load indefinitely divisible load – are:

$$L_i(t+1) = L_j(t+1) = \frac{1}{2}(L_i(t) + L_j(t))$$

The evolution of the global system load is the consequence of several asynchronous applications of the above.

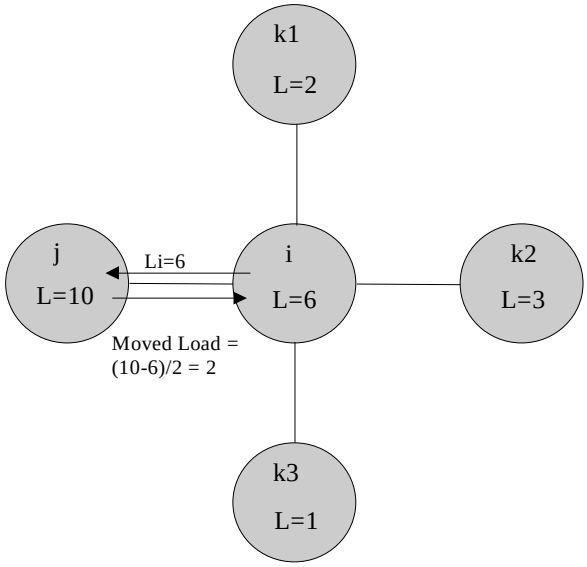

Figure 1. Load information exchange in the direct-neighbor policy

The triggering event to start the decisional activity on one node is the asynchronous arrival of updated load information, either from one of the neighbor nodes or caused by a change in the local state. Whenever one node receives updated load information, it compares it with the other available load information. If the local current load exceeds the load of its less loaded neighbor by more than a threshold (i.e., a percentage), a migration action is started toward it. Though the algorithm has been implemented as sender-initiated [EagLZ86], a receiver-initiated implementation is also possible and would not change the basic algorithm behavior. After a sender-receiver couple is established, the sender decides the entities to migrate: more than one entity can move in one migration step, though the granularity of the load – not indefinitely divisible – makes it sometimes impossible either to perfectly balance the load of the two nodes (or even to issue any migration between them).

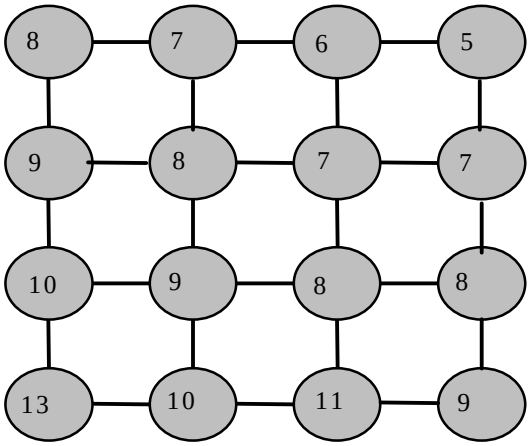

Figure 2. A global imbalance that cannot be locally recognized (the numbers in the nodes represent their load)

3. Local Policies with Distorted Load Information

Though local load balancing policies represent a forced design choice in massively parallel system, they present several drawbacks.

The local perspective in which load balancing decisions have to be taken can sometime make decisions not consistent with the global system state. It is proved that distributed load balancing policies based on local information provide to monotonically diminish the global imbalance [Cyb89, CorLZ96], but there is not any guarantee that their actions stop only after having reached a global balance. In fact, the necessary presence of a threshold of no-action can sometime lead a policy to inactivity, because of the recognition of a local balance even in presence of a high residual global imbalance (as in figure 2). In addition, the limited information available and the limited scope of the actions can force the distributed components to issue a high number of separated actions to achieve a given load balance. With reference to figure 1, the amount of load moved from *j* to *i* by the direct-neighbor policy is under-evaluated w.r.t. the more global situation of the system. In fact, the nodes *k1 k2* and *k3* are even more underloaded. Further step of the policy will cause load to be moved from *i* to *k1*, *k2* and *k3* and, then, to move further load from *j* to *i*.

This section presents the distorted load information exchange scheme integrated in the direct-neighbor policy to overcome the above-identified limits. In addition, it presents a further extension to the direct-neighbor policy that permits to better exploit the introduced information exchange scheme.

3.1 The Distorted Load Information Exchange Scheme

The basic idea of the proposed information exchange scheme is to distort the load information exchanged between neighbor nodes in order to make the information delineate a more global view of the system. Again with reference with figure 1, let one suppose the node *i* does not communicate to *j* its effective load (L_i) but, instead, a somehow inferior value DL_i (in consideration of the fact that all the neighbors of *i* but *j* are underloaded). If node *j*, by its side, interprets this information as representative of the effective load of the sender, the following load balancing action in *j* will move to *i* an amount of load greater than the one effectively needed to balance the two nodes. This excess of load received from *j* could be used by *i* to balance itself with its other neighbors.

Another situation where a similar scheme can be useful is the one in which the policy activities stop because each replicated component detects a local balance, though the system is globally imbalanced. In figure 2, for example, the North sector of the system is significantly less loaded than the South sector. Because the threshold prevents actions unless the imbalance in a neighborhood is of a significant amount, it is possible that none of the distributed components, having all of them neighbors with only a slightly different load, starts a load balancing action. The exchange of distorted load information, in this case, could force movements and overcome the problem.

Formally, in the proposed scheme, the effective load (L_i) of one node *i* is weighted with the average load of its neighborhood. The distorted load information (DL_i) sent by a node *i* to its neighbor *j* is:

$$DL_i = wL_i + \frac{(1-w)}{M-1} \sum_{k \in N, k \neq j} DL_k$$

Where L_i represents the effective load of the node *i*, *M* is the number of neighbors of the node *i* and DL_k represents the load of the node *k* to the knowledge of node *i* (that is, at its time, a distorted load information). *w* is a real between 0 and 1 and represents the weight given to the effective load w.r.t. the load of the neighbors. When *w=1* the scheme is not distorted.

The load balancing action that follows the receipt of a distorted load information strives to equalize the effective load of the node that receives the information and the distorted load of the neighbor, i.e.,

$$L_j(t+1) = \frac{1}{2}(L_j(t) + DL_i(t))$$

With reference to figure 3, the node *i* transmit its load information to node *j*. Though the effective load of node *i* is 6, the load information transmitted (with a weight *w=0,5*) is:

$$DL_i = 4 = 0{,}5 * 6 + \frac{1-0{,}5}{3}(2+3+1)$$

At the receipt of this load information, node *j* supposes that node *i* has a load of 4 and, then, sends it an amount of load of 3, in order to equalize their loads. Though this amount is excessive w.r.t. the real load of *i*, its movement is likely to be effective. In fact, following load balancing actions in *i*, will provide to move the load in excess to its underloaded neighbor, i.e., *k3*.

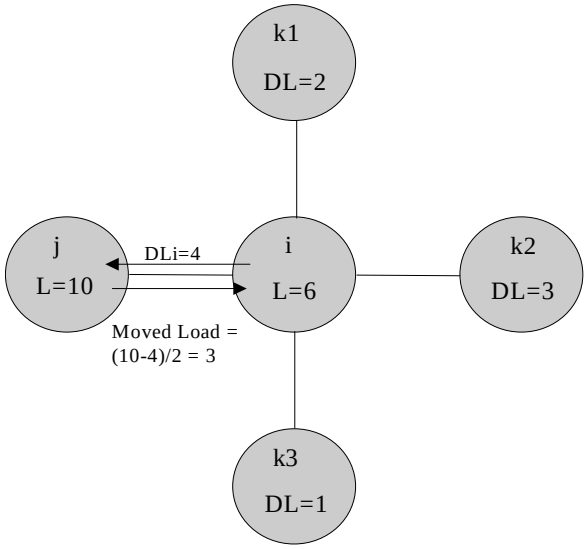

Figure 3. The node *j* receives distorted load information about the load of its neighbor *i* and moves load in excess toward it (L= effective load, DL= distorted load information)

The opposite situation is represented in figure 4. Though the effective load of node *i* is 6, the load information it transmit to *j* is 8, because of *i*'s overloaded neighbors *i*. In this case, the

distorted load information prevents node *j* in sending an excess of load to *i*: *j*, supposing to equalize the load the two nodes, send to *i* an amount of load of 1. Sending more load, in this case, were not worth because that would make it more difficult for *k1, k2* and *k3* to discharge, at their turn, their excess of load.

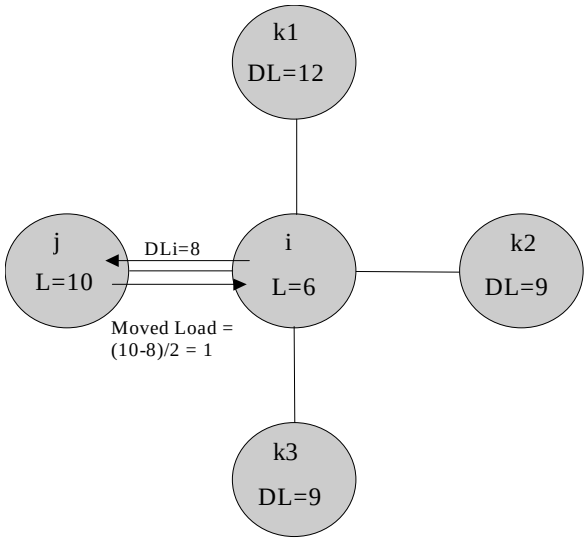

Figure 4. The node *j* receives distorted load information about the load of its neighbor *i* and limits the load moved toward it

3.2 Extended Load Distribution Scheme

A simple extension to the direct-neighbor policy permits to exploit additional information to locate the destination of migrating entities [CorLZ97, Wu95]. In the direct-neighbor policy, once a sender-receiver couple is established, the load to be migrated is allocated on the receiver node. However, the receiver node holds load information about different nodes than the receiver. Then, it has the possibility of detecting the presence of even less loaded nodes. The extended load distribution scheme takes advantage of this situation by allowing the receiver to forward the migrating load to more and more underloaded nodes. Load migration stops only when no useful movements are detected, i.e., a node is reached whose load is minimal in its neighborhood (as in figure 5).

The above load distribution scheme can be particularly effective in presence of the distorted load information exchange scheme, as figure 6 shows. Because of the distorted load information, node *j* moves an excess of load to *i*; this load is immediately forwarded by *i* to *k3*, thus achieving

in a single migration action what would have needed, in the basic direct-neighbor policy, several actions.

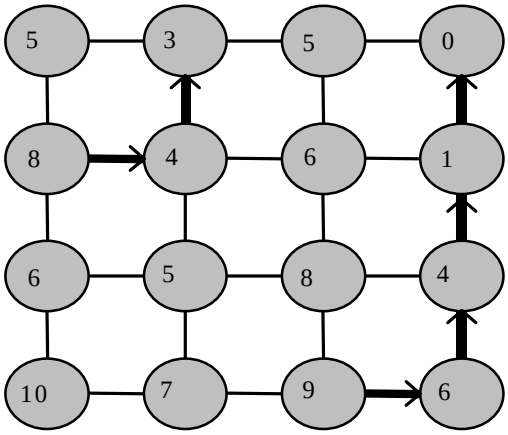

Figure 5. The extended load forwarding scheme
(the number in the nodes represents the effective load of the nodes)

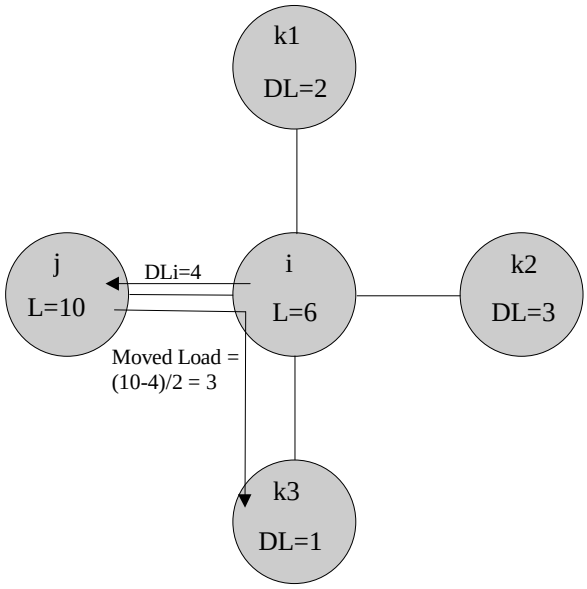

Figure 6. Exploiting the extended load distribution scheme to forward load in excess

From an implementation point of view, the scheme can be realized with a simple protocol: when a direct-neighbor couple is established and a node *i* is waiting to receive load, it looks in its neighborhood for one less loaded node. If this node exists, the node *i* acts as a forwarder at the receipt of the migrating load. Any node up to a local minimum of load repeats the same protocol.

4. Performance Evaluation

To evaluate the effectiveness of the presented policies, I have used as target a 100-nodes transputer-based architecture (a Meiko CS-1 [Mei89]), shaped after a 10x10 mesh. Though obsolete, this target architecture is still a good evaluation testbed and does not invalidate our results. In fact, the locality concept upon which our policies are based – and from which the results derive – is independent from the characteristics of the communication hardware.

The policies has been implemented by means of one multithreaded policy handler, called Allocation Manager (AM for short), replicated in each node of the system. Each AM is in charge of implementing the allocation policy and of coordinating itself with the AMs of the neighbor nodes.

To evaluate a wide range of load situations without being committed to a specific application, I have allocated "dummy" processes onto the execution load, with the aim of generating execution load. No inter-process dependencies are modeled. The process migration time, i.e., the time for a process on a node to be frozen, transferred and resumed on a different node, is about 25ms in the target architecture.

4.1 Load Metrics

The paper identifies a few load factor relevant towards the study of local load balancing policies: the *imbalance*, i.e., the deviation of the load situation from the ideal balanced situation, the *dispersion*, i.e., the presence in the system of a heterogeneous distribution of overloaded and underloaded nodes, and the *dynamicity*, i.e., the frequency of the dynamic changes of the load.

The load *imbalance* is the global standard deviation of the load of all the system nodes (let N be their number), normalized to the average system load:

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

The achievement of a good load balancing quality can be measured by the capacity of a policy of keeping σ low during the execution.

The imbalance in the system can be more or less distributed: overloaded and underloaded nodes can be homogeneously dispersed in every part of the system or they can be concentrated in regions. The *dispersion* indicator (δ) measures this property: it represents the load standard deviation (again normalized to the average system load) calculated as if each node had a load equal to the average of its load and of the ones of its neighbors, i.e.,

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

where μ_i is the average load of the domain of nodes D_i, composed by N_i nodes, i.e., i and its N_i-1 direct neighbors:

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

If δ is small compared with σ, each neighborhood reflects a local imbalance comparable to the global system imbalance: that indicates a homogeneous distribution of the imbalance in the system. If δ is close to σ, the imbalance is concentrated and it is not easily recognizable in its real magnitude with only a local view of the system. Figure 2 reports an example of one imbalance with a high δ.

To characterize the load *dynamicity*, the previously described indicators (σ and δ) are extended to take into account dynamic variations. In particular, the fluctuations of the load between time t and $t+\Delta t$ are measured by the normalized standard deviation of the load changes in the time span, as if it were the only load to take into account. Formally:

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

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 in one Δt. A more integral indicator of the dynamic load keeps into account over a time unit T the load variation in contiguous intervals of time:

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

Similarly, one can further characterize the dynamic changes of load by measuring the dispersion of the load generated in a given interval (Δδ/T). The experiments have tested a wide range of

dynamic situations (by dynamically creating processes with different completion times), to covers most practical cases corresponding to real-world applications, from static or quasi-static ones to highly dynamic ones ($\Delta\sigma$/sec=200) and with different values of the dispersion of the generated load ($\Delta\delta$/sec=30%-70% of $\Delta\sigma$).

Another important load factor toward the study of all kinds of load balancing policies is the *granularity* of the load, i.e., the average number of entities allocated onto each node and their variance. As a general rule, when the granularity is too coarse it prevents any load movement and does not permit to achieve good load balancing. Similarly, the variance of the load imposed by each entity slightly influences the load balancing quality: its growth makes the achieved load balancing quality worsen, because the AMs have less chances to find entities with the appropriate granularity to be selected for migration. Because granularity and variance of the load have a similar impact for all algorithms and do not change their relative performances, the following of this section assumes a granularity of 20 with 0 standard deviation and a threshold of 20%. The interested reader can refer to [CorLZ96] for a detailed analysis of this issue.

4.2 Quasi-Static Load Situations

Figures 7 and 8 report the load balancing quality (i.e., the average σ) achieved depending on the dynamicity of the system load, i.e., of $\Delta\sigma$/sec, for two different values of $\Delta\delta$/sec; figure 9 and 10 report the corresponding number of migrations issued in a for each $\Delta\sigma$ generated.

When the system load exhibits a very low degree of dynamicity, i.e., in quasi-static load situations, all policies achieve good load balancing with a limited number of migrations.

The basic direct-neighbor policy without distorted information (i.e., *w=1*) exhibits the worst behavior in quasi-static and slowly dynamic low situation (see the left part of the figures): it is not able to reach the same balancing quality of the other policies and it employs the higher number of migrations.

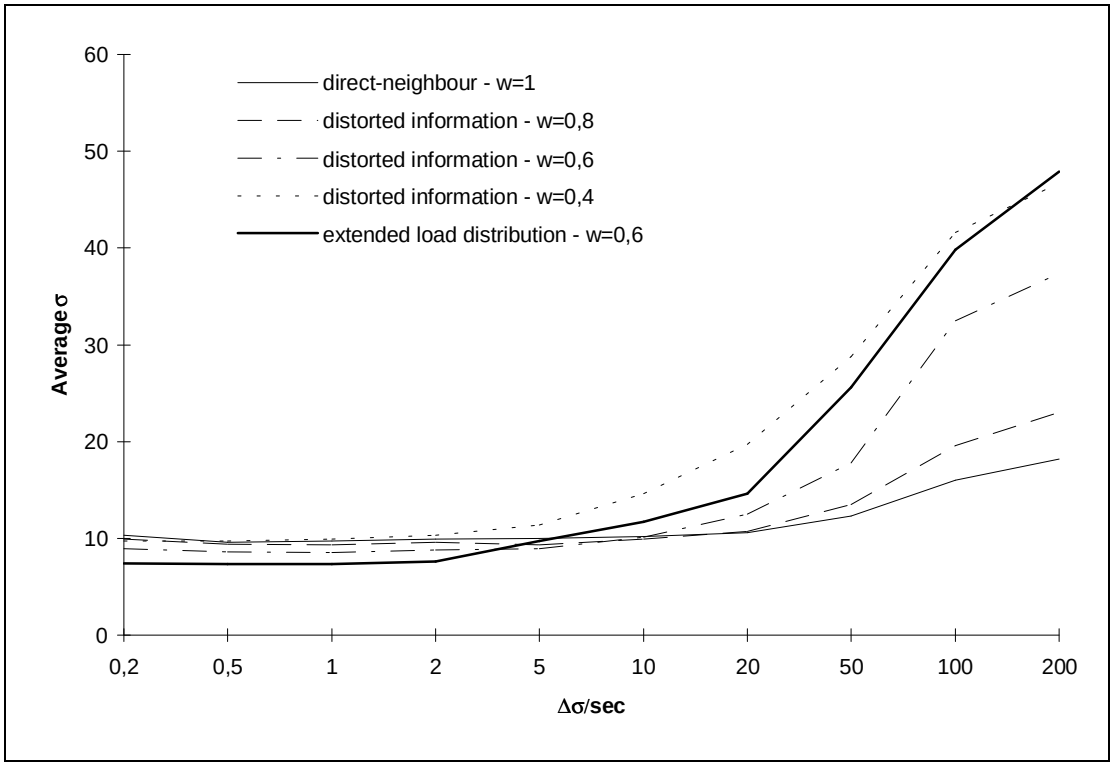

Figure 7. Load balancing quality depending on the dynamicity of the load; low dispersion (Δδ/sec = 30% of Δσ/sec)

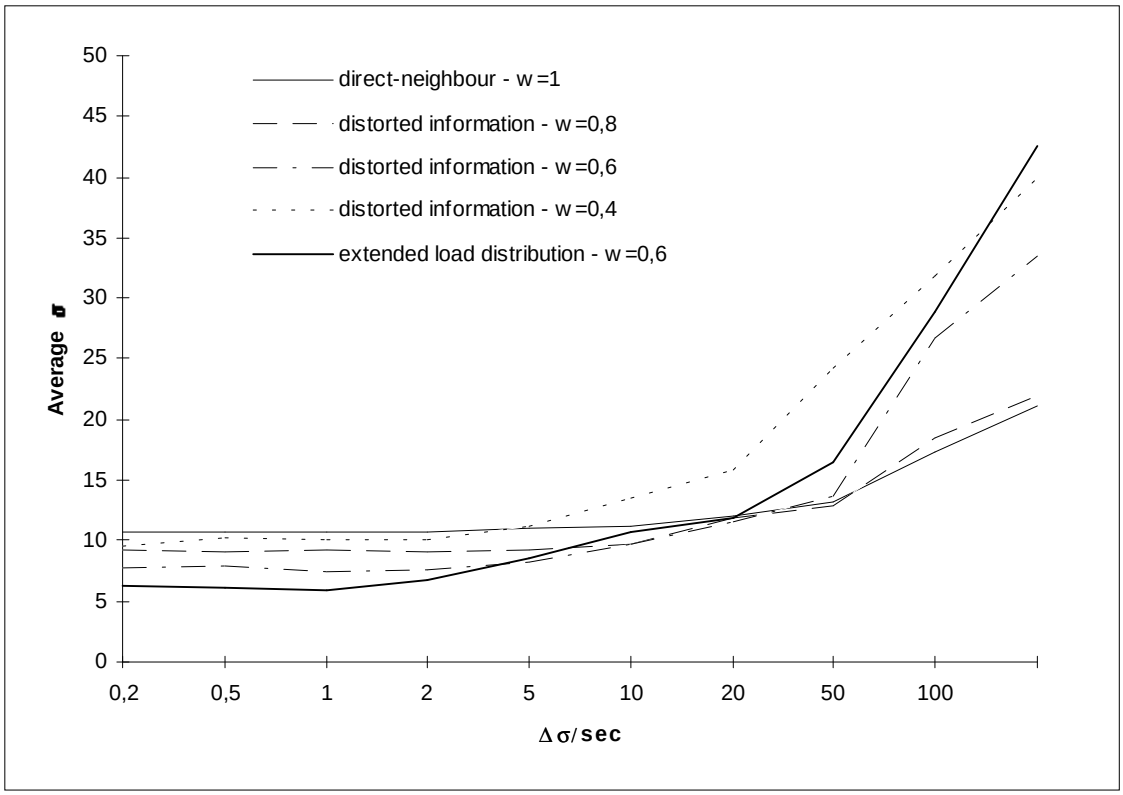

Figure 8. Load balancing quality depending on the dynamicity of the load; high dispersion (Δδ/sec = 70% of Δσ/sec)

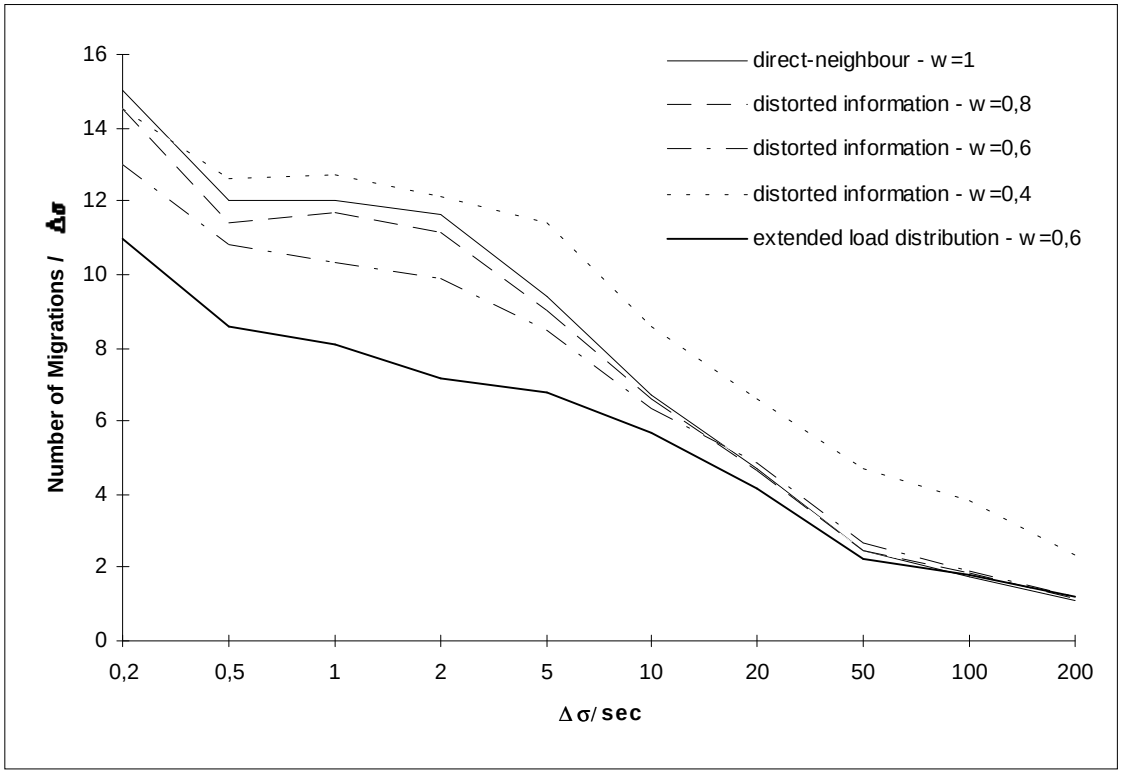

Figure 9. Number of migrations per Δσ depending on the dynamicity of the load; low dispersion (Δδ/sec = 30% of Δσ/sec)

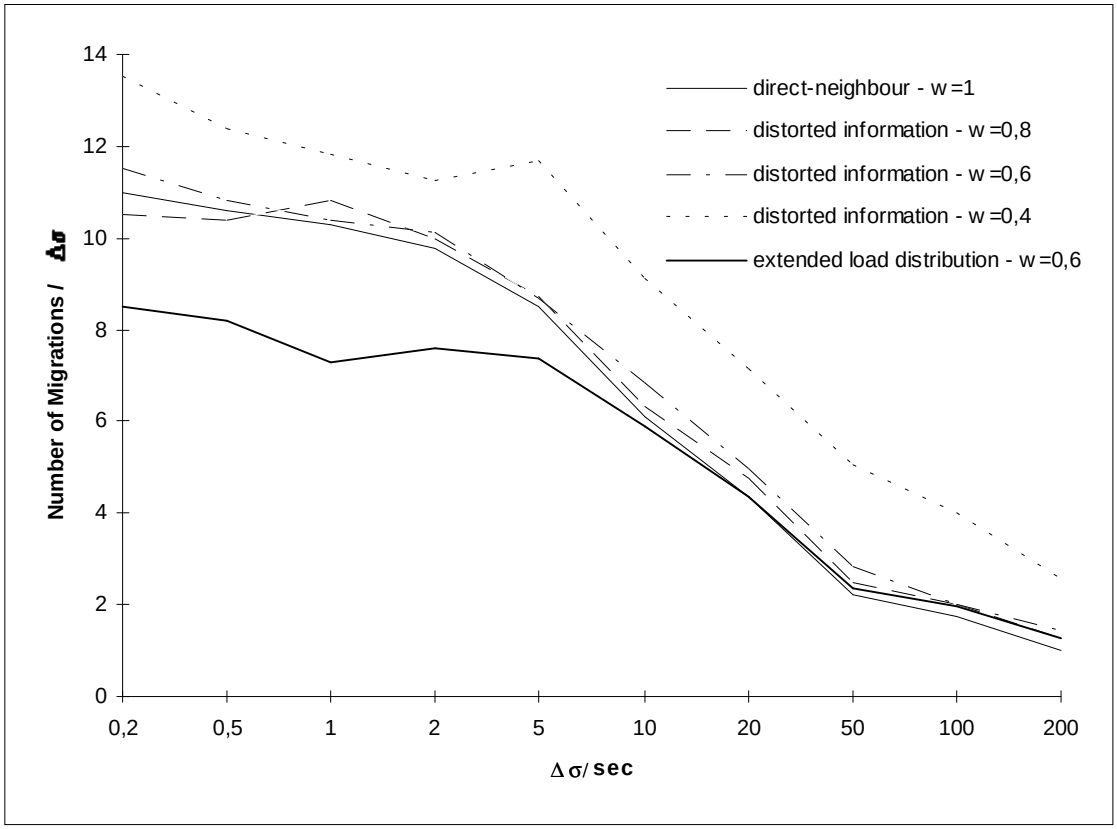

Figure 10. Number of migrations per Δσ depending on the dynamicity of the load; high dispersion (Δδ/sec = 70% of Δσ/sec)

The distorted load information exchange scheme improves the load balancing quality and decreases the number of migrations, because it achieves more focused migration decisions. In case of too high distortion of the load information (i.e., of low *w*), the scheme deteriorates its behavior: the residual σ is high and the number of migrations is large. As expected, highly-distorted load information make the load balancing actions no longer reflect the real needs of the system nodes and do not permit the policy to exploit them effectively. The experiments have revealed that the values of *w* that lead to the best load balancing quality with the lowest migration effort are comprised, in slowly dynamic situations, between 0,5 and 0,6.

The extended load distribution scheme, applied with a moderated distortion of load information exchange, behaves the best, by obtaining the best load balancing quality with the lowest migration effort. The capability of immediately forwarding the load to less and less underloaded nodes permits to efficiently exploit the excess of load one node is likely to receive from its neighbors to limit the number of migrations.

By comparing figures 7 and 8, one can see that the distorted scheme achieves the greatest improvement to the achieved load balancing in case of a high $\Delta\delta$. When $\Delta\delta$ is low even the simple direct-neighbor policy achieves quite good load balancing. When $\Delta\delta$ increases, the local scope can make it impossible for the direct-neighbor policy to detect the magnitude of the global imbalance. The distorted load information exchange scheme and the extended load distribution one enlarge their local view of the system and makes the policy less and less sensitive to the dispersion of the imbalance.

4.3 Dynamic Load Situations

The situation described in the previous sub-section dramatically changes when the dynamicity of the load becomes higher. While all the policies decrease their capacity of keeping the system well balanced, the basic direct-neighbor policy shows itself more robust (see the right side of figures 7 and 8). The distorted load information scheme makes the policy less and less robust as the degree of distortion increases. The extended load distribution scheme makes the policy even less robust w.r.t. load dynamicity. Also in this case, $\Delta\delta$ influences the behavior of the policies:

because the distorted load information scheme and the extended distribution scheme are more robust w.r.t. $\Delta\delta$, the higher the $\Delta\delta$ of the generated load, the higher the $\Delta\sigma$/sec at which these two schemes start behaving worse than the basic direct-neighbor policy. Nevertheless, whatever the $\Delta\delta$, there exists a point in which the load dynamicity makes the distorted load information scheme and, then, the extended load distribution one, less effective in load balancing.

These results derives from two main factors: slowness in reaction and over-reactivity.

When the actions of a policy are delayed w.r.t. the age of the information upon which they are based, the load information could have become obsolete. Thus, instead of producing a better balance, the actions could even produce a worse imbalance. The simple direct-neighbor policy, triggered by any new load information, bases its decisions only on the load information that come from its neighbors, not weighted with any other load information coming from farther nodes. Then, the probability of obsolescence of this load information is very low. The distorted load information exchange scheme, instead, makes the policy base its decisions on the load information coming also from non-direct-neighbor nodes. The risk of evaluating obsolete load information is higher, making the scheme less efficient for highly dynamic applications.

With regard to the extended load distribution scheme, its limited robustness in highly dynamic situations is not simply caused by slowness but from a form of over-reactivity caused by inaccuracy in load information. The scheme makes load migrate farther to more and more underloaded nodes, if any less loaded node is detected in the neighborhood, with no consideration of the threshold. Thus, because the available load information may be inaccurate – and this problem is exacerbated in highly dynamic situations – the absence of threshold could cause load to be forwarded to more loaded nodes, with an effect of over-reaction in load balancing that leads to instability.

4.4 Discussion

The *normalized process response time* [Keo96] can be introduced as a global indicator to comprehensively evaluate the impact of the presented policies on applications. For a given load balancing policy, the normalized response time (*NRT* for short) is defined as follows:

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

Where RT_{LB} is the response time obtained by applying the load balancing policy, RT_{NLB} represents the response time in absence of the load balancing policy, RT_{IDEAL} is the expected response time in presence of an ideal load balancing tool that achieves a perfect load balancing at zero cost. The higher the efficiency of a load balancing 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 load balancing policy.

Figures 11 and 12 report the *NRT* (referred to the average process response times) achieved by the implemented 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 reported data confirm our previous considerations about the behavior of the policies:

- a limited distortion of load information achieves benefits over the simpler direct neighbor scheme in slowly dynamic situations;
- the extended load distribution scheme can be effectively exploited by distorted load information exchange scheme to provide further benefits in slowly dynamic situations;
- neither the distorted load information exchange scheme nor the extended load distribution one are effective in highly dynamic situations, when the basic direct-neighbor scheme exhibits the better performances.

Note that the extended schemes not only do not produce any benefits in highly dynamic situations, but they also worse the response time (negative *NRT*) w.r.t. the no load balancing (NLB) case. A further increase of dynamicity over the shown range would let the basic direct-neighbor policy produce a negative *NRTs* too. However, the considered range of dynamicity is wide enough to cover most practical situations that can benefit from a load balancing tool.

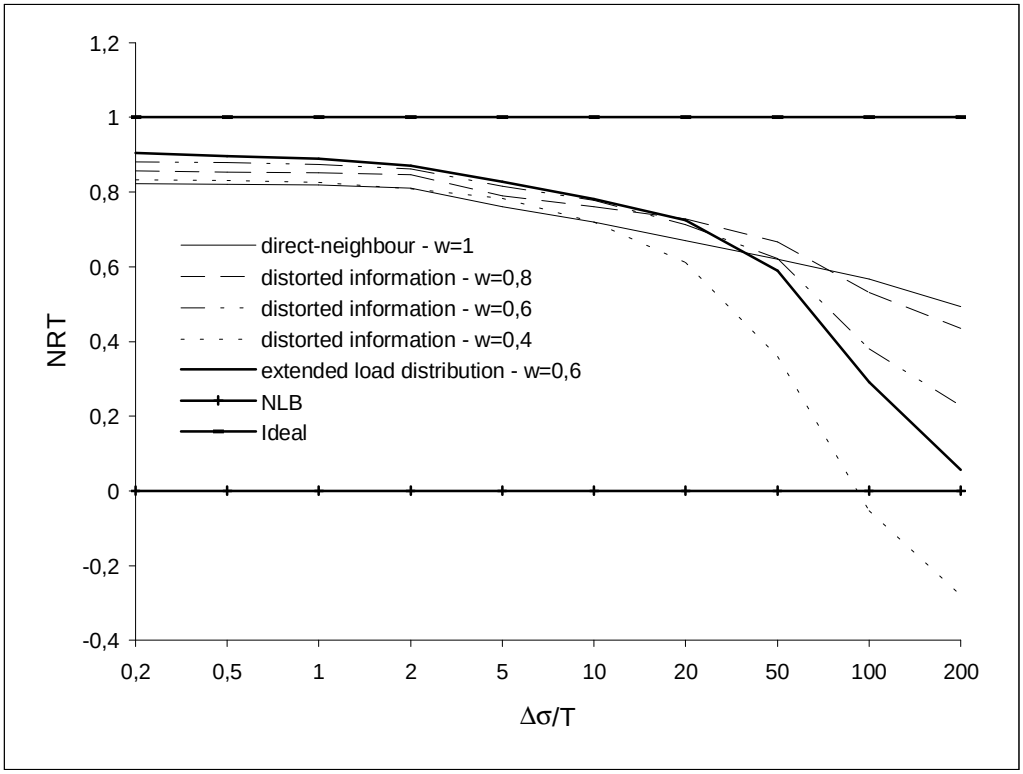

Figure 11. NRT depending on the dynamicity of the load; low dispersion (Δδ/sec = 30% of Δσ/sec)

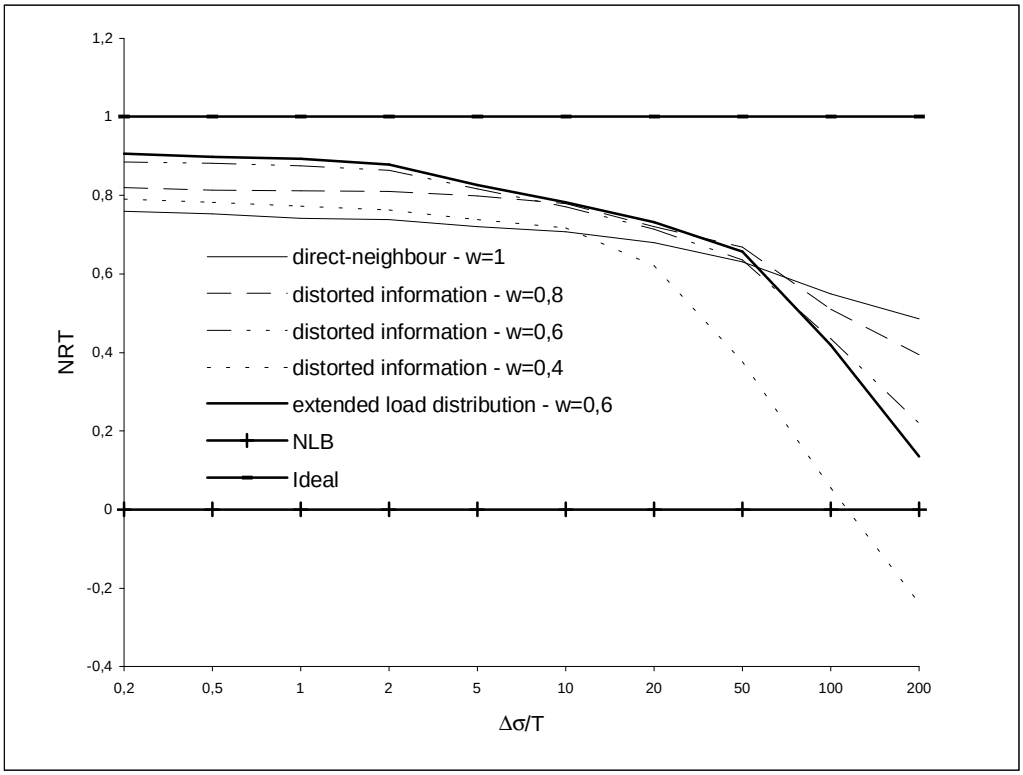

Figure 12. NRT depending on the dynamicity of the load; low dispersion (Δδ/sec = 70% of Δσ/sec)

5. Related Work

Several works deal with algorithms based on the direct-neighbor concept. However, most of them stress formal proofs and do not pay attention to concrete issues [Cyb89, QiaY91, XuL94, MurV97]. On the one hand, these algorithms assume a global synchronization of the distributed activities that hardly applies to real implementations on massively parallel architectures. On the other hand, they aim to compute the convergence rate of the algorithms and to determine the optimal schemes for load exchange, but the results apply to static load situations only and to synchronous implementations only [Xu95]. In general, they give little information about the behavior of the policies in real dynamic execution environments.

Different local load balancing policies have been described and evaluated in a real parallel architecture with artificially-generated load [WilR93] or in the context of peculiar application areas [WalB95, XuTM95]. Though these works provide useful information on the behavior of the policies, they lack in evaluating the effect of different degrees of dynamicity on the performances of the policies. A detailed analysis of the effect of dynamic load balancing in a dynamic environment can be found in [CanP95]. However, this work does not aim to compare the behavior of different load balancing policy, neither local ones, but rather to build a general model for dynamic parallel applications under dynamic load balancing.

The issue of achieving global view of the system while keeping the communication restricted to direct-neighbors is addressed in the well-known gradient model [LinK87] and its extensions [LulMR91]: each underloaded node signals its presence to its neighbors; each node communicates to its neighbors its distance from the nearest underloaded nodes. This aims to build a “proximity” map for underloaded nodes to be used to move load in the direction of the nearest underloaded node. Though conceptually interesting, the algorithm does not take into account the obsolescence of the load information and the intrinsic delay in which the construction of the proximity map incurs. Then, it can hardly fit dynamic load situations.

A local policy based on distorted load information is presented in [DutM94]: a node transmits to its neighbours, instead of its real load, the average load of its neighborhood. The approach,

though similar to the one presented in this paper, does not permit the parameterization of the degree of distortion and has not been evaluated with different degrees of dynamicity but only within a specific application.

Load balancing algorithms with extended load distribution schemes similar to the one presented in the paper are described in [Kal88] and [Wu95]. Apart from some difference that can improve their effectiveness in case of high δ – the first defines minimum and maximum limits for the number of forward actions, the seconds integrates receiver-initiated actions – both these extensions are likely to exhibit a weak behavior in highly dynamic load situations. I have already experienced the extended load distribution scheme in past works [CorLZ97] but never experienced it with distorted load information.

6. Conclusions

The paper focuses on distributed load balancing policies that achieve load balancing only by exchanging load information between neighbor nodes. A new scheme of information exchange is presented in which the load information transmitted are distorted to make them take into account a larger view of the system and overcome the limit of the local view. The paper evaluates the behavior of the policies in several load situations. The main result is that the enlargement of the locality scope produces better results for slowly dynamic applications and poorer performances in highly dynamic situations.

On the basis of the above result, I am currently working in the design of an adaptive load balancing policy that automatically tunes its internal parameters (i.e., the weight *w* of the distorted load information exchange scheme) depending on the dynamicity of the execution environment [Dec95, AvvRV97]. This will probably require a deeper analysis of the weight *w* and of its impact depending on the characteristics of the system load.

References

[AvvRV97] M. Avvenuti, L. Rizzo, L. Vicisano, “A Hybrid Approach to Adaptive Load Sharing and its Performance”, Journal of Systems Architecture, Vol. 42, No. 9&10, pp. 679-696, Feb. 1997.

[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.

[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.

[CorLZ96] A. Corradi, L. Leonardi, F. Zambonelli, “Diffusive Algorithm for Dynamic Load Balancing in Massively Parallel Architectures”, DEIS Technical Report No. DEIS-LIA-96-001, University of Bologna, April 1996. Available at http://www-lia.deis.unibo.it/Research/TechReport.html

[CorLZ97] A. Corradi, L. Leonardi, F. Zambonelli, “Performance Comparison of Diffusive Load Balancing Policies”, Proceedings of EUROPAR ’97, Lecture Notes in Computer Science, No. 1300, Springer-Verlag (D), pp. 882-886, Sept. 1997.

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

[Dec95] T. Decker, R. Diekmann, R. Lüling, B. Monien, “Towards Developing Universal Dynamic Mapping Algorithms”, Proceedings of the 7th IEEE Symposium on Parallel and Distributed Processing, pp. 456-459, 1995.

[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.

[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.

[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.

[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. Luling, B. Monien, F. Ramme, “Load Balancing in Large Networks: A Comparative Study”, Proc. of the 3rd IEEE Symposium on Parallel and Distributed Processing (SPDP'91), pp. 686-689, IEEE CS Press, 1991.

[Mei89] Meiko Ltd., Computing Surface Technical Manual, *Meiko Technical Report,* 1989

[MurV97] T. A Murphy, J. G. Vaughan, “On the Relative Performance of Diffusion and Dimension Exchange Load Balancing in Hypercubes”, 5th EUROMICRO Workshop on Parallel and Distributed Processing, London (UK), January 1997, pp. 29-34, IEEE CS Press.

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

[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.

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

[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.

[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, pp. 707-736, Oct. 1995.

[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, pp. 139-147, 1994.

[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, pp. 72-85, January 1995.

[XuTM95] C. Z. 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.