JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING 23, 246–255 (1994)

A Uniform Framework for Dynamic Load Balancing Strategies in Distributed Processing Systems[1]

TAIEB F. ZNATI[2] AND RAMI G. MELHEM

Computer Science Department, University of Pittsburgh, Pittsburgh, Pennsylvania 15260

Load balancing plays a central role in processor utilizations in distributed systems. Several strategies have been proposed in the literature to achieve load balancing. Usually, these strategies attempt to achieve a tradeoff between reducing the execution time of an application and minimizing the synchronization and the communication overhead. In this paper, we present a general model in which load balancing decisions are reached by enforcing performance metrics which may be adapted to reflect the specific requirements of different environments. Many of the load balancing schemes that have been suggested in the literature can be viewed as specific instances of the general framework presented in this paper. The basic scheme in this framework uses a load contention number that accounts for the load of the processors, the communication cost and the distance among processors. It is meant to be adaptable to the overall load on the system, the load on the communication devices, the run time characteristics of the tasks, and the configuration of the system. Furthermore, its implementation is not computationally complex. Thus, the gains made by load balancing are not overshadowed by the load balancing cost. © 1994 Academic Press, Inc.

1. INTRODUCTION

The full potential of any distributed processing system can be realized by equally sharing the computational and communication load among all processors. The uniform distribution of the load among the various processing nodes maximizes resource utilization and enhances the total throughput of the system. This problem is referred to as *Load Balancing*.

Load balancing strategies may be either *static* or *dynamic*. In a static approach, the distributed process is viewed as a collection of tasks with a priori known dependencies. Consequently, load balancing can be realized by statically assigning the tasks to different processors in order to achieve system wide objectives such as minimum average response time, minimum communication traffic and maximum throughput. Once assigned to a particular processor, the task is bound to run on that processor until completion [4, 11, 17]. The problem of finding an optimal task assignment is known to be NP-complete [5] and several heuristics, based on graph theory, queueing theory, and network flow theory, have been proposed to solve it [12, 13].

Dynamic load may be modeled by dynamically created task precedence graphs. In this approach, tasks are spawned dynamically and spawned tasks may, in turn, spawn more tasks [14, 15]. The performance of a dynamic load balancing strategy depends upon the process migration mechanism and the size of information domain analyzed for load distribution. In essence, dynamic load balancing may be viewed as an ongoing decision process in which individual processors attempt to utilize a local view of the global state of the system to make independent decisions aimed at meeting global objectives. The performance associated with the dynamic load balancing strategy is a function of the collection of decisions made at the physically distributed nodes. The efficiency of the strategy depends on the cost of interprocess communication, the logical complexity introduced, and the need to maintain global state information at each processor involved in the process of balancing the load. Different strategies that strike a balance between performance and overhead have been suggested in the literature [1–3, 6, 8–10, 16, 18]. Most of the existing schemes employ a *centralized model*, a *fully distributed model*, or a *semi-distributed model*.

In a centralized model, a dedicated node is responsible for maintaining a global state of the system. Based on the gathered information, the central node schedules tasks to individual nodes. For large systems, the computational and storage overheads needed to maintain the global state become prohibitive. In a fully distributed model, nodes of the system build their own views of the global state, and make autonomous decisions based on information exchanged with other nodes. Several topology-dependent and hierarchical schemes have been proposed to improve the performance of the model. In general, however, decisions based on partial views of the global state lead to suboptimal decisions. Finally, a semidistributed model aims at exploiting the advantages of both centralized and distributed models. This is usually achieved by dividing the system into different regions. Within each region, a centralized scheduling strategy aims at balancing the load within the region. A higher level scheduling mechanism is used to migrate tasks among regions. The

[1] This research was partially supported by the National Science Foundation under Grant CDA-8920839.

[2] The author also holds a joint appointment in IS/telecommunication.

0743-7315/94 $6.00

performance of this strategy strongly depends on the criteria used to partition the nodes into separate regions.

The purpose of this paper is to provide a parametric model for load balancing strategies. For specific parameter values, the model expresses many of the load balancing strategies suggested in the literature [1, 13–15]. The parameter space is continuous, thus allowing adaptation to the system's characteristics and environment. The basic idea of the scheme is to use a load contention number that accounts for the load of the processors, the communication cost, and the distance among processors. When a new task is created, it is assigned to the processor with the lowest contention number.

In the remainder of this paper, the basic characteristics of the environment are described, and a *profitability model* to assess the performance of dynamic load balancing algorithms is provided. A new parametric scheme to achieve dynamic load balancing is shown to be *profitable* according to our model. It is also shown that this scheme can be tuned to control the relative emphasis of communication and computation costs. Furthermore, it is shown that, for specific values of the parameters, the scheme reduces to some well known load balancing strategies. Finally, the results of a simulation study to assess the performance of our scheme under different operating conditions are presented.

2. ENVIRONMENT CHARACTERISTICS AND ASSESSMENT MODEL

The environment considered is a loosely coupled multiprocessing system with a number of processing elements interconnected through a point-to-point interconnection network. The topology of the system is represented by an undirected graph, $G = \langle N, E \rangle$, where N is a set of nodes (representing processors), and E is a set of edges (representing communication links). The parameters n and Δ are used to denote the cardinality of N and the diameter of G, respectively. The shortest path between nodes s and d in G is denoted by $\delta(s, d)$. In addition to the data interconnection network, we assume that the elements are connected by a broadcast based control subnet, such as a bus, a tree, two-dimensional spanning buses, or a hypercube.

Tasks are spawned, terminated, or newly generated at arbitrary rates in different nodes. However, a task may be processed at any node of the system. A task constitutes the unit of computational activity in the system. The model does not make any assumption about the particular operation of any given task. Consequently, a task may be a job, a program, or a process. Let $U_s(t)$ be the unfinished work at processor P_s at time t. This represents the computation and communication *load* required by all the tasks assigned to P_s at time t.

For a given task, τ, forked at P_s, $0 \le s < n$, we define the following:

- $\pi_s(\tau)$ as the cost of processing τ at P_s,
- $\nu(\tau)$ as the number of messages caused by the migration of τ from P_s to another processor, P_d. These messages are required to handle the initiation of the task τ on P_d and the communication between P_s and P_d during the execution of τ.

Since messages involve overhead in both time and space, they become a primary component of dynamic load balancing performance. The main objective of any dynamic scheme is to strike a balance between the overhead of synchronization and the degree of global state knowledge. In addition, local communication dependencies must be taken into consideration. In other words, an efficient solution must ensure that applications composed of tasks with a high degree of local communication dependency are allocated to closely connected processors.

Two criteria may be used to evaluate the performance of a load balancing strategy. The first is the overhead of task migration and the second is the ability to achieve load balancing. We define the overhead cost, $C_{s,d}(\tau)$, caused by migrating τ from P_s to P_d as

$$C_{s,d}(\tau) = M_{s,d}(\tau) + \omega,$$

where $M_{s,d}(\tau)$ is the communication cost and ω is the overhead required to determine the destination P_d to which τ is to be migrated. The communication cost can be expressed as $M_{s,d}(\tau) = \nu(\tau) f(\delta(s, d))$ for some function f which may depend on the communication load of the system as well as the topology and the bandwidth of the communication network. In general, f may be very complex and may depend on the dynamics of the system. A simple approximation is to assume that f is a linear function thus resulting in $M_{s,d}(\tau) = \nu(\tau)\mu_r\delta(s, d)$, where μ_r is the cost of routing and transmitting a message between two adjacent nodes in the network.

A *Profitability Assessment Function* (*PAF*), is usually used to assess the profitability of migrating a given task, τ, from the source node to another node in the system. This factor is related to the degree of *load imbalance*, Φ, at time t. Several metrics have been used in the literature to define Φ. For example, some schemes attempt to reduce the difference among unfinished works at different nodes of the system by defining Φ as the root mean square difference of unfinished work of all processors [7]. Other metrics aim at reducing the difference between the most loaded processor and the least loaded processor. In our scheme, we adopt the latter metric, for which

$$\Phi(U_0(t), U_1(t), \ldots, U_{n-1}(t)) = \{\max_{0 \le i < n} U_i(t) - \min_{0 \le j < n} U_j(t)\}.$$

Based on the above criteria, a dynamic load balancing algorithm can be conceptually defined as follows:

For each task, τ, forked on P_s at time t **do**
begin
d = PAF(t, τ, s);
if $d \neq s$ migrate τ to P_d.
end

where the Profitability Assessment Function, $PAF(t, \tau, s)$, determines the destination P_d for the task, τ forked at P_s. This function is described in Fig. 1, where $U_i'(t)$ is the load at P_i if the task is migrated to P_d. Nonerroneous minimization of Φ is defined next:

DEFINITION. The processor loads $\{U_0, \ldots, U_{n-1}\}$ are said to be *processor-wise heavier* (pw-heavier) than the processor loads $\{U_0', \ldots, U_{n-1}'\}$ if, for some permutation, σ, $U_{\sigma(i)} \geq U_i'$, for $i = 0, \ldots, n-1$.

DEFINITION. If $\Phi(U_0', \ldots, U_{n-1}') \leq \Phi(U_0, \ldots, U_{n-1})$ and $\{U_0', \ldots, U_{n-1}'\}$ are pw-heavier than $\{U_0, \ldots, U_{n-1}\}$, then $\Phi(U_0', \ldots, U_{n-1}')$ is said to be *erroneously smaller* than $\Phi(U_0, \ldots, U_{n-1})$.

The reason behind the above definition is that reducing Φ by merely increasing the load on the system should not be profitable. To illustrate this, consider a three-processor system in which $U_0 = 5$, $U_1 = 15$, and $U_2 = 10$. For this system $\Phi(5, 15, 10) = 10$. Clearly, increasing the load U_0 to 8 will decrease the load imbalance to $\Phi(8, 15, 10) = 7$, but will certainly not be profitable to the system. Similarly, decreasing U_2 to 8 while increasing U_0 to 10 will reduce the load imbalance to $\Phi(10, 15, 8) = 7$, but will not be profitable to the system since $\{10, 15, 8\}$ is pw-heavier than $\{5, 15, 10\}$. These two examples show that erroneous reductions of Φ are not profitable to a system.

Few remarks regarding Fig. 1 are in effect. First, the cost $M_{s,d}$ is equally charged to the source and the destination processors. A different PAF may be easily written for nonsymmetric systems in which that cost is charged unevenly to the source and the destination processors. Second, the overhead, ω, of determining the destination is not included in Fig. 1. This cost is fixed for a given scheme, and thus will not affect the choice of the destination, P_d. Finally, it is obvious that some of the parameters involved in the computation of $PAF(t, \tau, s)$, such as $\nu(\tau)$ and $\pi_s(\tau)$, may not be available when τ is forked. A practical load balancing algorithm should not use such parameters.

In the next section, we introduce a general load balancing scheme which allows highly interacting tasks to be

Begin
Find d that minimizes, non-erroneously, $\Phi(U_0', U_1', \ldots, U_{n-1}')$, where
$U_d' = U_d(t) + M_{s,d}(\tau) + \pi_d(\tau)$
$U_s' = U_s(t) + M_{s,d}(\tau)$, if $s \neq d$
$U_i' = U_i(t)$, for $0 \leq i < P$, $i \neq s$, $i \neq d$,
return (d)
end

FIG. 1. Profitability assessment function.

determine processor P_d that **minimizes**

$$LCN_d(s) = U_d(t) + M_{s,d}(\tau)$$

If $d \neq s$ and $M_{s,d}(\tau) < \pi_s(\tau)$, **then** migrate τ to processor P_d

FIG. 2. LCN-based load balancing strategy.

assigned to neighboring processors, thus preserving the local communication dependencies of the application.

3. BASIC DYNAMIC LOAD BALANCING SCHEME

The main purpose of the algorithm is to migrate tasks from heavily loaded processors to lightly loaded processors in the system. The basic scheme takes the communication overhead into consideration, thus reflecting the importance of the communication architecture for the processing capabilities of the processors.

Let P_s, $0 \leq s < n$, be the processor where a newly created task, τ, originates. The proposed algorithm for choosing a processor to which τ is to be migrated is shown in Fig. 2. To implement this algorith, each processor, P_d, computes a *Load Contention Number*, LCN_d, and the task is assigned to the processor holding the lowest Load Contention Number. The LCN of P_d reflects the current load of P_d, the communication requirement of the task, $\nu(\tau)$, and the cost of transmitting a message between P_d and P_s.

The algorithm in Fig. 2 may be used to balance the load on any system. However, the following theorem shows that this algorithm exactly implements the PAF of Fig. 1 for homogeneous systems.

THEOREM 1. *In a homogeneous system, if a task τ is generated at processor P_s, then the algorithm of Fig. 2 will minimize Φ in a nonerroneous sense.*

Proof. Let $\pi_i(\tau) = \pi(\tau)$ for any i and let P_u and P_d be two processors different from P_s. To prove the theorem, we need to show that if $LCN_d < LCN_u \leq LCN_s$, then migrating τ to P_d rather than keeping it on P_s is profitable (reduces Φ) if $M_{s,d}(\tau) < \pi(\tau)$, and that migrating τ to P_d is more profitable than migrating it to P_u. To simplify the notation, we omit the argument τ from M and π.

We divide the proof into two parts. First, we prove that if $M_{s,d} < \pi$, and

$$U_d + M_{s,d} < U_s, \tag{1.a}$$

then

$$\phi_{s,d}(U_s + M_{s,d}, U_d + M_{s,d} + \pi) < \phi_{s,d}(U_s + \pi, U_d), \tag{1.b}$$

where $\phi_{i,j}(U_i, U_j)$ is a convenient notation for $\Phi(U_0, \ldots, U_{n-1})$ in which only the loads used as arguments may change, while the other loads are fixed. Using $U_d < U_s$

from (1.a), and $M_{s,d} < \pi$, we get

$$\min\{U_s + M_{s,d}, U_d + M_{s,d} + \pi\} > \min\{U_s + \pi, U_d\} = U_d$$

$$\max\{U_s + M_{s,d}, U_d + M_{s,d} + \pi\} < \max\{U_s + \pi, U_d\} = U_s + \pi$$

which implies (1.b).

For the second part of the proof, we show that if

$$U_d + M_{s,d} < U_u + M_{s,u} < U_s \tag{2.a}$$

then either $\{U_s + M_{s,d}, U_d, U_u + M_{s,u} + \pi\}$ is pw-heavier than $\{U_s + M_{s,d}, U_d + M_{s,d} + \pi, U_u\}$, or

$$\phi_{s,d,u}(U_s + M_{s,d}, U_d + M_{s,d} + \pi, U_u) < \phi_{s,d,u}(U_s + M_{s,u}, U_d, U_u + M_{s,u} + \pi) \tag{2.b}$$

If $U_d \geq U_u$, then from (2.a), $M_{s,d} < M_{s,u}$, and thus $\{U_s + M_{s,u}, U_d, U_u + M_{s,u} + \pi\}$ is pw-heavier than $\{U_s + M_{s,d}, U_d + M_{s,d} + \pi, U_u\}$.

If $U_d < U_u$, and $U_d + \pi > U_s$, then $U_u + \pi > U_s$ and from (2.a)

$$\min\{U_s + M_{s,d}, U_d + M_{s,d} + \pi, U_u\} > \min\{U_s + M_{s,u}, U_d, U_u + M_{s,u} + \pi\} = U_d \tag{3.a}$$

$$U_d + M_{s,d} + \pi = \max\{U_s + M_{s,d}, U_d + M_{s,d} + \pi, U_u\} < \max\{U_s + M_{s,u}, U_d, U_u + M_{s,u} + \pi\} = U_u + M_{s,u}, + \pi \tag{3.b}$$

which implies (2.b).

Finally, if $U_d < U_u$, and $U_d + \pi \leq U_s$, then (3.a) still holds, but $\max\{U_s + M_{s,d}, U_d + M_{s,d} + \pi, U_u\} = U_s + M_{s,d}$, and thus, (3.b) does not hold. To prove (2.b) in this case, we use the definitions of $U'_0, \ldots, U'_{n-1}$ from Figure 1 and let $U_{\max} = \max\{U'_0, \ldots, U'_{n-1}\}$. If $U_{\max} \geq U_s + M_{s,d}$, then (3.a) is enough to prove (2.a). If, on the other hand, $U_{\max} < U_s + M_{s,d}$, then

$$\phi_{s,d,u}(U_s + M_{s,d}, U_d + M_{s,d} + \pi, U_u) = U_s + M_{s,d} - \min\{U_u, U_d + M_{s,d} + \pi\}$$

$$\phi_{s,d,u}(U_s + M_{s,u}, U_d, U_u + M_{s,u} + \pi) = U_s + M_{s,u} - U_d$$

from which (2.b) can be proven for that case. ■

A load balancing strategy based on LCN assigns a new task, τ, to the processor with the minimum LCN. The processor at which τ originates, P_s, computes its LCN and advertises it in an attempt to determine the processor where τ may be executed. The search for the recipient of τ results in the beginning of a *Load Contention Phase* (LCP). During this phase, all processors that are currently holding a LCN smaller than the currently advertized LCN will express their eligibility to receive the newly generated task by advertising their own LCN. The search continues until the processor with the minimum LCN is identified. Note that the termination detection of the bidding process depends on the type of control subnet supporting the computation. In a bus based control subnet, we assume that the time is slotted. A slot is defined to be the maximum amount of time it takes a processor to successfully transmit a control message. In this environment, the bidding process continues until an empty slot is observed over the subnet. The occurrence of such a slot signals the end of the LCP and determines uniquely the processor to which the newly generated task is to be migrated. In a hypercube based configuration, the LCP consists of determining the minimum of a set of numbers, and can be achieved in $\log_2(n)$ steps.

Note that the LCN does not depend on the execution time, $\pi(\tau)$, which is usually not known at the time τ is forked. Moreover, with a reasonable estimate of the number of messages, $\nu(\tau)$, caused by the migration of τ to d, $M_{s,d}(\tau)$ may be approximated by $\nu(\tau)\mu_r\delta(s, d)$ as mentioned in Section 2. Given that a quantitative value for $\nu(\tau)$ may not be available at the time of the load balancing decision, and that for structured computations it is reasonable to assume that $\nu(\tau)$ is uniform among all tasks, we will include the effect of ν and μ_r into one factor μ. That is, we will use

$$LCN_d = U_d(t) + \mu\delta(s, d). \tag{4}$$

The load balancing strategy is to minimize $U_d(t) + \mu\delta(s, d)$, which is equivalent to minimizing $(1/\mu)U_d(t) + \delta(s, d)$. In either case, the quantity to be minimized will be referred to as $LCN_d(s)$.

From Theorem 1 and the above discussion, we conclude that if (1) the communication requirement is uniform for all the tasks, (2) the communication cost is symmetric ($M_{s,d} = M_{d,s}$) and charged to both the sender and the receiver, and (3) the communication cost is linearly dependent on the distance, then the load balancing algorithm based on minimizing the Load Contention Number defined by (4) does minimize the difference, Φ, between the most heavily loaded and the most lightly loaded processors in the system. The algorithm, however, may be applied to any system, even if the above three conditions are not satisfied. In such cases, the algorithm is expected to reduce, rather than minimize Φ.

In the formula for LCN, μ reflects the impact of the underlying communication architecture. In other words, if μ is negligible, the migration decision is hardly influenced by the communication overhead. However, if μ is large, the migration decision is mostly influenced by the communication overhead. In the next section, we show that μ can be tuned to control the relative emphasis of load and distance, resulting thereby in a spectrum of different load balancing strategies.

4. STRATEGIES FOR LOAD BALANCING

Assume that a task, τ, is generated at P_s. If $\mu = 0$, then $LCN_d(s)$, generated by a processor P_d contending for the migration of τ, reduces to $U_d(t)$. The strategy resulting from the above contention number uses the load as the only criterion to achieve load balancing [17]. On the other hand, a strategy of *no migration* regardless of the system state can be obtained by setting the value of μ to the maximum possible load achievable by a processor. This strategy attempts to maximize system performance by eliminating communication cost, without consideration for load balancing. In this case, $LCN_d(s)$ is reduced to $U_d(t) + U_{\max}\delta(s, d)$, where $U_{\max}$ is the maximum load in the system. Given that $U_s(t) < U_d(t) + U_{\max}\delta(s, d)$ for any $d \neq s$, then, irrespective of its load, the processor where the newly created task originated will always produce a smaller contention number.

In the remainder of this section, the value of μ is adjusted to reflect different load balancing strategies. The goal is to strike a balance between the load and the distance according to performance criteria prescribed by the computation environment and the communication cost.

4.1. Load Predominant Strategy

For many applications, the main objective of the load balancing algorithm is to distribute the load evenly among different processors. For this class of algorithms, the load is the predominant factor in the decision, and the distance between the sender and the receiver is only used to break ties in favor of the closest least loaded processor in the system. This strategy can be achieved by setting μ to $1/\Delta$. Consequently, the contention number reduces to $LCN_d(s) = \Delta U_d(t) + \delta(s, d)$.

For example, in Table I, we show the LCN of a processor, P_d, for different values of $U_d(t)$ and $\delta(s, d)$, assuming that $\Delta = 4$ and that a new task is generated at s. The LCN of a processor which is at a distance δ from s and which has a load U is found at the intersection of row U and column δ of the table. The table shows that the load contention number increases as the load of a processor increases. Consequently, the least loaded processor generates the smallest contention number. If more than one processor holds the smallest load, then the processor closest to P_s produces the smallest contention number.

TABLE I
LCN: Load Predominant Strategy

	$\delta(s, d)$				
Load, $U_d(t)$	0	1	2	3	4
0	0	1	2	3	4
1	4	5	6	7	8
2	8	9	10	11	12
3	12	13	14	15	16
.	.	.	.	.	.
.	.	.	.		.

TABLE II
LCN: Distance Unit ≡ 2 Load Units

	$\delta(s, d)$				
Load, $U_d(t)$	0	1	2	3	4
0	0	2	4	6	8
1	1	3	5	7	9
2	2	4	6	8	10
3	3	5	7	9	11
4	4	6	8	10	12
.	.	.	.	.	.

4.2. Strategies Combining Distance and Load

In the above scheme, the effect of the distance on the LCN is reduced to breaking ties among contending processors holding the same load. However, when communication is costly, it is desirable to add more weight to the distance factor in the migration decision. More specifically, a mechanism whereby a unit of distance equates to k units of load is required. The resulting load balancing strategy may be obtained by setting μ to k. In this case, the value of the LCN becomes $U_d(t) + k\delta(s, d)$. An example describing the resulting contention numbers when $k = 2$ is illustrated in Table II.

It is clear that as k increases, the weight of the distance in the migration decision increases, and becomes totally predominant when $k = U_{\max}$. A similar observation can be made for the lower range of the parameter k. As k decreases, the emphasis on the load increases, and becomes predominant when $k = 1/\Delta$. In this case, the load always prevails over distance. In essence, the factor k defines the *load equivalent to a unit distance*. In making migration decisions, a processor with load U at a distance δ from s is equivalent to a processor with load $U - k$ at a distance $\delta + 1$ from s.

4.3. Region and Load-Band Based Strategies

A load balancing algorithm aims at improving system throughput by avoiding idle or lightly loaded processors. However, the cost of implementing the algorithm may be excessive when all processors are involved in the bidding algorithm. This cost may be reduced by introducing the notion of *Balancing Regions* [15]. For a given processor, P_s, a balancing region is defined as the set of all processors that are at distance R from P_s. Consequently, the bidding process is restricted to the balancing region of every process. This restriction can be incorporated into LCN by setting $\mu = U_{\max}/R$, resulting in

$$LCN_d(s) = U_d(t) + \frac{\delta(s, d)}{R} U_{\max}.$$

TABLE III
LCN: Regions of Influence Based Strategy

	$\delta(s, d)$			
Load, $U_d(t)$	0	1	2	3
0	0	0	U_{max}	U_{max}
1	1	1	$U_{max} + 1$	$U_{max} + 1$
2	2	2	$U_{max} + 2$	$U_{max} + 2$
.	.	.	.	.
U_{max}	U_{max}	U_{max}	$2U_{max}$	$2U_{max}$

Note that in this scheme, the load is only important within the region. Furthermore, using $\lfloor \delta(s, d)/R \rfloor$ in $LCN_d(s)$ eliminates the effect of distance within the region. In either case, however, nodes outside the region are excluded from the migration contention. Table III shows the resulting contention numbers for $R = 2$, based on $LCN_d(s) = U_d(t) + \lfloor \delta(s, d)/R \rfloor U_{max}$.

For some applications, the profitability of migrating a task from one processor to another may be limited if the difference in the processors' loads is small. In this case, migration may only take place if the difference of the loads exceeds a certain parameter B, referred to as the *band of influence*. This strategy may be obtained by setting $\mu = B/\Delta$, resulting in

$$LCN_d(s) = \Delta \frac{U_d(t)}{B} + \delta(s, d)$$

Furthermore, using $\lfloor U_d(t)/B \rfloor$ in $LCN_d(s)$ eliminates the effect of the distance within the load band, resulting in a *threshold based strategy* [19]. An example describing the contention numbers obtained for $B = 2$ and $\Delta = 3$ is shown in Table IV. The table shows that the loads are grouped in bands of 2. Within the same band, only the distance has an effect upon the contention number.

4.4. State-Based and Node-Specified Strategies

In all of the above algorithms, the load balancing procedure is invoked every time a new task is created. In order to avoid unnecessary invocations, state-based load balancing algorithms have been proposed [14]. In these algorithms, a *threshold load* is defined to represent the load of a processor beyond which the utilization of that processor cannot be improved. Specifically, a processor may be in one of three loading states: *light*, *optimal*, or *heavy*. When the state of a processor becomes heavy, the processor seeks to transfer some of its load to a lightly loaded processor. The mechanism used to migrate the load is similar to the bidding procedure used in the basic load balancing algorithm described above. When a processor moves into a heavily loaded state, it advertises its Load Contention Number in an attempt to identify a set of lightly loaded processors to which excessive load may be transferred. Upon completion of the *Load Contention Phase*, the heavily loaded processor gradually disperses its load among the set of eligible processors, until the optimal state is reached.

In order to cast this state-based algorithm in terms of our LCN strategy, we let $L_1 - 1$ be the maximum load a lightly loaded processor may have, and $L_2 - 1$ the maximum load an optimal processor may accumulate. We then set $LCN_d(s) = \varepsilon_{1i}\Delta U_d + \alpha$, where $\alpha = \delta(s, d)\varepsilon_{1i} + (\Delta + 1)U_{max}(\varepsilon_{2i} + \varepsilon_{3i})$, ε_{ij} represent the Kronecker function, and i is determined such that $L_{i-1} \leq U_d < L_i$ ($L_0 = 0$). With this, a minimum LCN strategy selects the closest lightly loaded processor for the execution of a task generated on a heavily loaded processor. Tasks generated on optimal or lightly loaded processors are not migrated. An example for the contention numbers when $L_1 = 10$, $L_2 = 20$, $U_{max} = 30$, and $\Delta = 3$ is shown in Table V.

Other variations of the above basic scheme may be obtained based on the mechanisms described to define the region and to specify the load bands. For example, instead of fixing the values of R and B for the entire system, each node, i, may specify its own region of influence and load band (R_i and B_i). This flexibility may be used to reflect the current status of the processors.

TABLE IV
LCN: Load Band Based Strategy

	$\delta(s, d)$			
Load, $U_d(t)$	0	1	2	3
0	0	1	2	3
1	0	1	2	3
2	3	4	5	6
3	3	4	5	6
4	6	7	8	9
5	6	7	8	9
6	9	10	11	12
.	.	.	.	.

TABLE V
LCN: State Based Strategy

	$\delta(s, d)$			
Load, $U_d(t)$	0	1	2	3
0	0	1	2	3
1	3	4	5	6
2	6	7	8	9
.	.	.	.	.
10	63	63	63	33
11	63	63	63	63
12	63	63	63	63
.	.	.	.	.
20	63	63	63	63
21	63	63	63	63
22	63	63	63	63
.	.	.	.	.

4.5. *Efficiency Improvement*

The cost of the *Load Contention Phase*, *LCP*, may be prohibitive if the load changes frequently. Consequently, a mechanism to prevent *spurious* load contention phases may reduce the overall cost of the load balancing strategy. A *look forward* based approach may be used to anticipate future task forking, and determine a potential processor for the migration of newly generated tasks. During an on-going bidding phase, a processor, P_c, may *memorize* a future recipient, *recpt_id*, of "its" future load. For that, P_c needs to determine the current load of the bidding processor. This information may be easily obtained based on the advertized contention number. Let P_s be the processor that initiates the current LCP, and let P_q be the current contender. It is clear that $LCN_q(c)$ can be written as

$$\begin{aligned} LCN_q(c) &= \frac{1}{\mu} U_q + \delta(s, q) + (\delta(c, q) - \delta(s, q)) \\ &= Tr(LCN_q(s)) \\ &= LCN_q(s) + (\delta(c, q) - \delta(s, q)), \end{aligned}$$

where $Tr(\ )$ represents an operator that transforms the advertised contention number relative to s into one relative to c. During the current LCP, P_c may execute the procedure shown in Fig. 3 to determine the potential recipient, *recpt_id* $= q$, of its future load. This procedure determines whether a current bidder, P_q, qualifies to be a potential recipient, in which case *recpt_id* is memorized by P_c.

Note that if the load of *recpt_id* increases by forking a task τ_1 before P_c forks a new task, τ_2, then *recpt_id* will initiate a bidding phase to determine the recipient of τ_1. Thus, P_c will again execute the procedure of Fig. 3 and determine a new *recpt_id*. If, on the other hand, the load of *recpt_id* does not increase (no τ_1 is generated), then it will be profitable to migrate τ_2 to P_{recpt_id}. It should be noted, however, that at the time when τ_2 is forked, some processor may have an LCN lower than that of P_{recpt_id}. In order to migrate τ_2 to the processor with the minimum LCN, a new round of bidding should be initiated by P_c. However, this bidding may start with $LCN_{recpt\_id}(c)$ instead of $LCN_c(c)$. Given that $LCN_{recpt\_id}(c) < LCN_c(c)$, fewer rounds of bidding will be needed to locate the processor with minimum LCN.

```
Function Future_Recipient()
Begin
  recpt_id = c
  recpt_lcn = Tr(LCN_recpt_id(s))
  while (bidding continues)
    if (Tr(LCN_q(s)) < recpt_lcn) then
      recpt_id = q
      recpt_lcn = Tr(LCN_q(s))
End
```

FIG. 3. Determination of potential recipient of future load.

5. A SIMULATION STUDY

A simulation model was developed to examine the average task response time, T, as a function of the parameter, μ. In this model, the basic scheme discussed in this paper is applied to an n-processor system interconnected by a point-to-point communication network. Task arrivals at a processor, P_i, is based on a *Poisson* process with rate of arrival λ_i. The sizes of the generated tasks are drawn from an exponential distribution with a user specified average of ψ. The computation and communication requirements of each task are also drawn from an exponential distributions with averages ε and χ, respectively. In this model, the unfinished work, $U_s(t)$, of a processor P_s at time t is approximated by the number of tasks at P_s. The results discussed in this section are for the case where $\psi = 10^4$ bits, $\varepsilon = 0.001$ s, and $\chi = 10^3$ bits.

Task arrival rates on different processors are varied to reflect the degree of per processor load imbalance in the system [15]. This degree of imbalance is specified by a parameter ζ, $0 \le \zeta < 1$. Specifically, if the system load is specified by a parameter ρ, $0 \le \rho \le 1$, then the normalized load, ρ_i, at processor P_i is given by

$$\rho_i = \bar{\rho}(1 - \zeta)^{\sigma(i)}, \qquad i = 0, \ldots, n - 1,$$

where σ is a random permutation from $0, \ldots, n - 1$ to $0, \ldots, n - 1$. The value of $\bar{\rho}$ is obtained by solving $(1/n) \sum_{i=0}^{n-1} \rho_i = \rho$. The task arrival rate at P_i is then obtained from $\lambda_i = \rho_i/\varepsilon$. Note that for relatively large n, a small ζ causes a wide variation in ρ_i. For example, for $\rho = 0.8$ and $n = 25$, a ζ of 0.02 causes the values of ρ_i, $i = 0, \ldots, n - 1$, to vary in the range $0.62 \le \rho_i \le 1.008$.

The simulated communication network is a packet-switched 25-processor mesh architecture with a packet size of 256 bits. The communication resulting from the migration of a given task, τ, of size ψ_τ embodies the cost of sending $\lceil \psi_\tau/256 \rceil$ packets from processor P_s to processor P_d to initiate the task, and the cost of exchanging $\lceil \chi_\tau/256 \rceil$ packets between P_d and P_s, where χ_τ is the communication requirement of task τ. Store and forward routing on the mesh is simulated, thus reflecting any delay in messages due to traffic congestion. The bandwidth of the communication channel between any two processors in the system is specified by a parameter C (expressed in bits per second).

In Fig. 4, the average task response time is depicted as a function of μ. As discussed in Section 3, when $\mu = 0$, the distance between the source and the destination processors do not affect the migration decision, while when μ is large, no migration ever takes place (no load balance). The effects of the communication bandwidth, C, the system load, ρ, and the imbalance factor, ζ, are shown in three different graphs. In Fig. 4a, ρ and ζ are

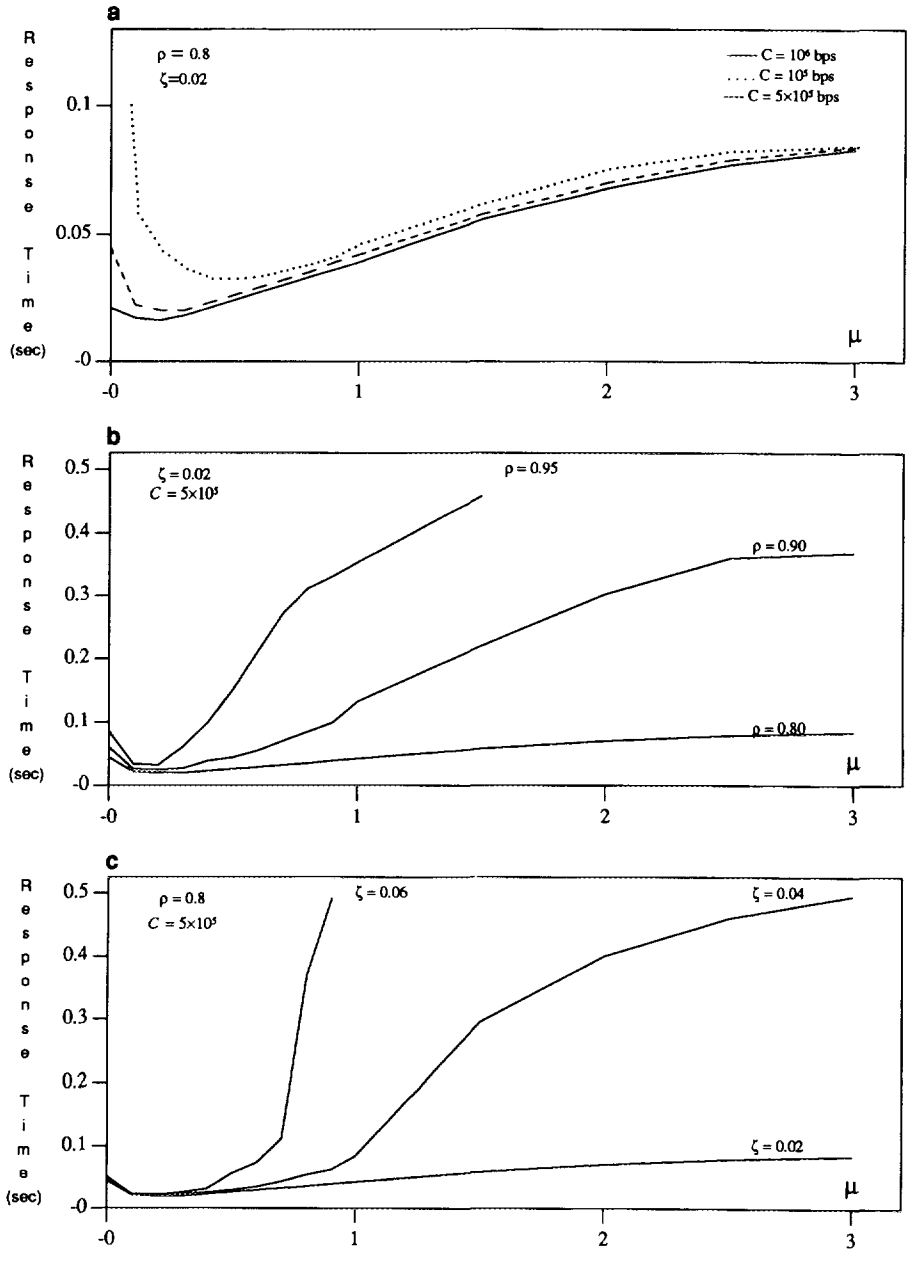

FIG. 4. Average task response time as a function of μ: (a) effect of the communication channel capacity C; (b) effect of the load ρ; (c) effect of the load imbalance ζ.

fixed at 0.8 and 0.02, respectively, and the response time is plotted for three different values of C. As expected, for large μ, the bandwidth, C, does not affect T since no migration is taking place (no load balance). To explain the behavior for small μ, note that the value of C does not affect migration decisions. Hence, when distance is not considered during load balancing ($\mu = 0$), load balancing may increase the response time, especially when communication is relatively expensive. This is observed in Fig. 4a when $C = 10^5$. However, when distance is considered during load balancing and the appropriate value of the parameter μ is used, an improvement in the response time is always observed. This improvement in response time is also observed for different values of ρ and ζ as shown in Figs. 4b and 4c.

The simulation results clearly show that the average task response time reaches a minimum for a given value of $\mu = \mu_{opt}$ which depends on the load, the imbalance factor, and the communication bandwidth. Other results, not reported here, show that the task response time depends also on the average task size ψ, the computation requirements ε, and the communication requirement χ.

TABLE VI
Strategies for Load Balancing

μ				
0	$1/\Delta$	B/Δ	U_{max}/R	U_{max}
Load based schemes	Load over distance	Band based schemes	Region based schemes	No load balancing scheme

In all cases, however, the average task response time reaches a minimum for a given μ_{opt}, for which the load balancing algorithm is optimum. The real significance of these results is that a simple policy can be developed, without explicit model assumptions, to dynamically monitor the parameter μ based the system load evolution. The monitored value can then be used to adjust the load balancing algorithm so that the task response time is minimum.

6. CONCLUSION

We have developed a general scheme for dynamic load balancing. This scheme is flexible and can be adapted to reflect the metrics of basic importance in the system. More specifically, μ, $U_d(t)$, and $\delta_s(d)$ are used to reflect the influence of the underlying architecture, the processors' loads, and the degree of task interactions, respectively. Different strategies result from specific values of the parameter μ as summarized in Table VI. Moreover, the system response time may be considered as a continuous function of μ, which exhibits a minimum at some $\mu = \mu_{opt}$. The value of μ_{opt} depends on the system load and traffic, and thus, if the average response time of the system is dynamically monitored, it is possible to update the parameter μ adaptively to minimize the response time.

The proposed load balancing algorithm is a heuristic that may be applied to a large class of multiprocessor systems with point-to-point interconnection networks. We proved that, under particular assumptions, the algorithm is optimal according to some performance metric. Moreover, simulation results for systems with nonuniform communication requirements and traffic-dependent communication cost show that the system response time is reduced when the algorithm is applied with the proper value of the parameter μ.

REFERENCES

1. G. Cybenko, Dynamic load balancing for distributed memory multiprocessors, *J. Parallel Distrib. Comput.* **7,** 2 (1989), 279–301.
2. D. Eager, E. Lazowska, and J. Zahorjan, A comparison of receiver-initiated and sender-initiated adaptive load sharing, *Sigmetrics* **13,** 12 (1985).
3. D. Eager, E. Lazowska, and J. Zahorjan, Adaptive load sharing in homogeneous distributed systems, *IEEE Trans. Software Engrg.* **12,** 5 (1986), 662–675.
4. K. Efe, Heuristic models of task assignment scheduling in distributed systems, *IEEE Comput.* **15,** 6 (1982), 50–56.
5. M. Garey and D. Johnson, in *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.
6. S. Gulati, J. Barhen, and S. Iyengar, The pebble-crunching model for fault-tolerant load balancing in hypercube ensembles, *Comput. J.* **33,** 3 (1990), 204–213.
7. C. Hsu and J. Liu, Dynamic load balancing algorithms in homogeneous distributed systems, *Proc. of the 6th Int. Conf. on Distributed Computing Systems*, 1986, pp. 216–223.
8. P. Krueger and M. Livny, A comparison of preemptive and nonpreemptive load distributing, *Proc. of the 8th International Conference on Distributed Computing Systems*, June 1988, pp. 123–130.
9. F. Lin and R. Keller, The gradient model load balancing method, *IEEE Trans. Software Engrg.* **13,** 1 (1987).
10. M. Litzkow, M. Livny, and M. Mutka, Condor—A hunter of idle workstations, *Proc. of the 8th International Conference on Distributed Computing Systems*, June 1988, pp. 104–111.
11. M. Livny, The study of load balancing algorithms for decentralized distributed processing systems, Ph.D. Dissertation, Wiezmann Institute of Science, Aug. 1983.
12. V. M. Lo, Heuristic algorithm for task assignment in distributed systems, *IEEE Trans. Comput.* **37,** 11 (1988).
13. L. Ni and K. Hwang, Optimal load balancing strategies for a multiprocessor system, *IEEE Trans. Software Engrg.* **11,** 5 (1985), 491–496.
14. L. Ni, C. Xu, and T. Gendreau, Distributed drafting algorithm for load balancing, *IEEE Trans. Software Engrg.* **11,** 10 (1985), 1153–1161.
15. J. Stankovic and I Sidhu, An adaptive bidding algorithm for processes, clusters, and distributed groups, *Proc. of the 4th Int'l Conference on Distributed Computing Systems*, 1984, pp. 49–59.
16. T. Suen and J. Wong, Efficient task migration algorithm for distributed systems, *IEEE Trans. Parallel Distribut. Systems* **3,** 4 (1992), 488–499.
17. A. Tantawi and D. Towsley, Optimal load balancing in distributed computer systems, *J. Assoc. Comput. Mech.* **32,** 2 (1985).
18. M. Theimer, K. Lantz, and D. Cheriton, Preemptible Remote Execution for the V System, *Proc. of the 10th ACM Symp. on Operating Systems Principles*, 1985, pp. 2–12.
19. M. Willebeck-LeMair and A. Reeves, Dynamic load balancing strategies for highly parallel multicomputer systems, Technical Report EE-CEG-89-14, Computer Engineering Group, Cornell University, 1989, also appeared in *Proc. of the Int. Conf. on Parallel Processing*, 1990.

TAIEB F. ZNATI obtained a Ph.D. degree in computer science at Michigan State University, East Lansing, in April 1988, and a Master of Science degree at Purdue University, West Lafayette, Indiana. In 1988, Dr. Znati joined the University of Pittsburgh, where he currently holds an associate professor position in the Department of Computer Science with a joint appointment in telecommunications in the Department of Library and Information Science. His current research interests fo-

cus on the design of network-level channel abstractions for real-time communication networks to support multimedia environments, the design and analysis of medium access control protocols to support distributed real-time systems, and the investigation of fundamental design issues related to distributed systems.

RAMI GEORGES MELHEM was born in Cairo, Egypt in 1954. He received a B.E. degree in electrical engineering from Cairo University in 1976, an M.S. degree in computer science in 1981, and a Ph.D. degree in computer science in 1983, both from the University of Pittsburgh. Since 1989, he has been an associate professor at the University of Pittsburgh. Previously, he was an assistant professor at Purdue University. He served on program committees of several conferences and workshops, and he is on the editorial board of the *IEEE Transactions on Computers*. He was Guest Editor of a special issue of the *Journal of Parallel and Distributed Computing* on optical computing and interconnection systems.

Received May 26, 1992; revised June 21, 1993; accepted July 9, 1993